Exploration of Cyber-Physical Systems for GPGPU Computer Vision-Based Detection of Biological Viruses

Dissertation

zur Erlangung des Grades eines

D o k t o r s d e r I n g e n i e u r w i s s e n s c h a f t e n

der Technischen Universität Dortmund
an der Fakultät für Informatik

von

Pascal Libuschewski

Dortmund

2017

Tag der mündlichen Prüfung:	22. März 2017
Dekan / Dekanin:	Prof. Dr.-Ing. Gernot A. Fink
Gutachter / Gutachterinnen:	Prof. Dr. Peter Marwedel
	Prof. Dr. Heinrich Müller

Dedication

To my father.

Acknowledgments

Above all, I would like to thank my supervisors Prof. Dr. Müller and Prof. Dr. Marwedel most gratefully for advice, support, inspiration, and guidance.

My very special thanks go to all my colleagues and friends at the chair of graphical systems and the design automation for embedded systems group at the TU Dortmund University. In particular I would like to thank Dr. Dominic Siedhoff and Olaf Neugebauer for all the joint research we conducted and Dr. Constantin Timm who strongly influenced the direction of my research with his thesis and advice.

Very special thanks to the collaborative research center SFB 876, the entire SFB 876-B2 team, and, particularly, to Dr. Alexander Zybin and Dr. Frank Weichert without whom the B2 project would not have been possible.

Furthermore, many thanks to all the people with whom I conducted and published research.

Finally, heartfelt thanks to all my friends and my family.

I would like to conclude these acknowledgments with a small appreciation of all life and all the decisions taken that lead to our existence today. Even the sheer number of conscious and unconscious decisions over the last 3.5 billion years is overwhelming and breathtaking. In the same manner, every decision that we take can influence numerous beings in a myriad of ways in the billions of years to come. All this comes down to each very moment. All there is, is now.

Abstract

This work presents a method for a computer vision-based detection of biological viruses in PAMONO sensor images and, related to this, methods to explore cyber-physical systems such as those consisting of the PAMONO sensor, the detection software, and processing hardware. The focus is especially on an exploration of Graphics Processing Units (GPU) hardware for “General-Purpose computing on Graphics Processing Units” (GPGPU) software and the targeted systems are high performance servers, desktop systems, mobile systems, and hand-held systems.

The first problem that is addressed and solved in this work is to automatically detect biological viruses in PAMONO sensor images. PAMONO is short for “Plasmon Assisted Microscopy Of Nano-sized Objects”. The images from the PAMONO sensor are very challenging to process. The signal magnitude and spatial extension from attaching viruses is small, and it is not visible to the human eye on raw sensor images. Compared to the signal, the noise magnitude in the images is large, resulting in a small Signal-to-Noise Ratio (SNR).

With the VirusDetectionCL method for a computer vision-based detection of viruses, presented in this work, an automatic detection and counting of individual viruses in PAMONO sensor images has been made possible. A data set of 4000 images can be evaluated in less than three minutes, whereas a manual evaluation by an expert can take up to two days. As the most important result, sensor signals with a median SNR of two can be handled. This enables the detection of particles down to 100 nm.

The VirusDetectionCL method has been realized as a GPGPU software. The PAMONO sensor, the detection software, and the processing hardware form a so called cyber-physical system. For different PAMONO scenarios, e.g., using the PAMONO sensor in laboratories, hospitals, airports, and in mobile scenarios, one or more cyber-physical systems need to be explored. Depending on the particular use case, the demands toward the cyber-physical system differ.

This leads to the second problem for which a solution is presented in this work: how can existing software with several degrees of freedom be automatically mapped to a selection of hardware architectures with several hardware configurations to fulfill the demands to the system? Answering this question is a difficult task. Especially, when several possibly conflicting objectives, e.g., quality of the results, energy consumption, and execution time have to be optimized.

An extensive exploration of different software and hardware configurations is expensive and time-consuming. Sometimes it is not even possible, e.g., if the desired architecture is not yet available on the market or the design space is too big to be explored manually in reasonable time. A Pareto optimal selection of software parameters, hardware architectures, and hardware configurations has to be found.

To achieve this, three parameter and design space exploration methods have been developed. These are named SOG-PSE, SOG-DSE, and MOGEA-DSE. MOGEA-DSE is the most advanced method of these three. It enables a multi-objective, energy-aware, measurement-based or simulation-based exploration of cyber-physical systems. This can be done in a hardware/software codesign manner. In addition, offloading of tasks to a

server and approximate computing can be taken into account. With the simulation-based exploration, systems that do not exist can be explored. This is useful if a system should be equipped, e.g., with the next generation of GPUs. Such an exploration can reveal bottlenecks of the existing software before new GPUs are bought.

With MOGEA-DSE the overall goal—to develop a method to automatically explore suitable cyber-physical systems for different PAMONO scenarios—could be achieved. As a result, a rapid, reliable detection and counting of viruses in PAMONO sensor data using high-performance, desktop, laptop, down to hand-held systems has been made possible.

The fact that this could be achieved even for a small, hand-held device is the most important result of MOGEA-DSE. With the automatic parameter and design space exploration 84 % energy could be saved on the hand-held device compared to a baseline measurement. At the same time, a speedup of four and an F_1 quality score of 0.995 could be obtained. The speedup enables live processing of the sensor data on the embedded system with a very high detection quality.

With this result, viruses can be detected and counted on a mobile, hand-held device in less than three minutes and with real-time visualization of results. This opens up completely new possibilities for biological virus detection that were not possible before.

Contents

1 Introduction **1**
1.1 Systems for Biological Virus Detection 1
1.2 Automatic System Optimization 4
1.3 Contribution of this Work 7
1.4 Organization of the Thesis 10
1.5 Author's Contribution to this Thesis 10
1.6 Acknowledgment 11

2 Biomedical PAMONO Sensor **13**
2.1 PAMONO Sensor 13
2.2 PAMONO Sensor Data 19
2.3 Summary 21

3 Related Work **23**
3.1 PAMONO Sensor 23
3.2 Image Processing, Computer Vision, and Pattern Recognition 25
3.2.1 Terminology 25
3.2.2 Image Processing 25
3.2.3 Computer Vision 26
3.2.4 Pattern Recognition 26
3.3 Random Forest and Code Generation 27
3.4 Optimization 28
3.4.1 Parameter Space Exploration 29
3.4.2 Design Space Exploration 30
3.4.3 Approximate Computing 36
3.5 Offloading 36

4 Basics **37**
4.1 General-Purpose Computing on Graphics Processing Units 37
4.1.1 OpenCL 39
4.1.2 GPGPU Notations and Definitions 41
4.1.3 Efficient Access to Global Memory 45
4.1.4 Efficient Use of Local Memory 45
4.1.5 Additional Constraints 49
4.2 Image Processing Notations and Definitions 53
4.3 Quantifying the Detection Quality 57
4.4 Evolutionary Algorithms 59
4.4.1 Motivation to Use EAs 60
4.4.2 EA Definitions 60
4.5 The SynOpSis Approach 61

5 Computer Vision-Based Detection of Biological Viruses with Virus-DetectionCL 65
5.1 Introduction . . . 66
5.1.1 Motivation . . . 67
5.1.2 Architectural Design of VirusDetectionCL . . . 67
5.2 Pre-Processing . . . 69
5.2.1 Input Conversion . . . 69
5.2.2 Brightness Correction . . . 70
5.2.3 Vibration Detection . . . 71
5.2.4 Detection of Overexposed Spots . . . 72
5.3 Signal Restoration . . . 73
5.3.1 PAMONO Signal Model . . . 73
5.3.2 Temporal Noise Reduction and Application of the Signal Model 74
5.3.3 Spatial Noise Reduction . . . 77
5.3.4 Fuzzy Spatiotemporal Noise Reduction . . . 79
5.4 Feature Extraction . . . 81
5.4.1 Spatial Feature Extraction . . . 82
5.4.2 Temporal Feature Extraction . . . 88
5.4.3 Spatiotemporal Fuzzy-Based Features . . . 94
5.4.4 Fuzzy-Based Combination of Features . . . 96
5.5 Segmentation . . . 97
5.5.1 Memory Management for Polygons . . . 98
5.5.2 Polygon Features . . . 98
5.5.3 Mapping Per-Pixel Features to Polygons . . . 100
5.6 Classification . . . 101
5.6.1 Threshold-Based Classification . . . 101
5.6.2 Random Forest-Based Classification . . . 102
5.7 Post-Processing . . . 103
5.8 Parameters . . . 104
5.9 Evaluation . . . 106
5.9.1 Evaluation of the Execution Time . . . 106
5.9.2 Evaluation of the Random Forest Classifier . . . 106
5.9.3 Evaluation of Detection Quality with the SynOpSis Approach . 107
5.10 Summary and Conclusion . . . 109

6 Automatic GPU Code Generation with RFC-Gen for the Application of Random Forest Models 111
6.1 The RFC-Gen Method . . . 111
6.2 Limitations of the RFC-Gen Method . . . 113
6.3 Summary and Conclusion . . . 113

7 Single-Objective PSE and DSE with SOG-PSE and SOG-DSE 115
7.1 Introduction to SOG-PSE and SOG-DSE . . . 116
7.1.1 Motivation . . . 117
7.1.2 Evaluated Scenarios . . . 118
7.2 General Methods for SOG-PSE and SOG-DSE . . . 119

7.2.1 Architectural Design of SOG-PSE and SOG-DSE 119
7.2.2 Parameter Space and Design Space Definition 120
7.2.3 Single-Objective Optimization 121
7.2.4 Considering Several Data Sets for Optimization 121
7.3 Single-Objective PSE with SOG-PSE 121
7.3.1 Methods 122
7.3.2 Experiments 124
7.3.3 Results 125
7.3.4 Discussion 126
7.3.5 Summary and Conclusion 128
7.4 Manually Performed DSE of Actual CPUs and GPUs 129
7.4.1 Methods 129
7.4.2 Experiments 130
7.4.3 Results and Discussion 131
7.4.4 Summary and Conclusion 133
7.5 Single-Objective Energy-Aware Hardware DSE with SOG-DSE 134
7.5.1 GPGPU-Sim Simulator 135
7.5.2 GPUWattch Simulator 136
7.5.3 Methods 138
7.5.4 Experiments 142
7.5.5 Results and Discussion 144
7.5.6 Summary and Conclusion 145
7.6 Single-Objective Power-Aware DSE of GPUs for HPC Systems with SOG-DSE 146
7.6.1 Motivation 146
7.6.2 Methods 147
7.6.3 Experiments 148
7.6.4 Results and Discussion 148
7.6.5 Summary and Conclusion 150
7.7 Limitations of SOG-PSE and SOG-DSE 151
7.8 Summary and Conclusion of the SOG-PSE and SOG-DSE Methods . . 151

8 Hybrid PSE and DSE with MOGEA-DSE **153**
8.1 Introduction to MOGEA-DSE 155
8.1.1 Motivation 155
8.1.2 Evaluated Scenarios 156
8.2 General Methods and Concepts of MOGEA-DSE 158
8.2.1 Architectural Design of MOGEA-DSE 158
8.2.2 Parameter Space and Design Space Definition 159
8.2.3 Multi-Objective Optimization with SPEA-2 and NSGA-II 164
8.2.4 GPUSimPow Simulator 166
8.2.5 Extension of GPUSimPow 167
8.2.6 Calculation of the Fitness Values 169
8.2.7 Summary and Conclusion 170
8.3 Multi-Objective Energy-Aware DSE of Simulated GPUs for Mobile, Desktop, and HPC Systems 170

8.3.1 Methods . . . 171
8.3.2 Experiments . . . 176
8.3.3 Results . . . 179
8.3.4 Discussion . . . 185
8.3.5 Summary and Conclusion . . . 187
8.4 Multi-Objective Offloading-Capable Hybrid PSE and DSE of Simulated GPUs for Mobile Devices . . . 188
8.4.1 CoPoMo LTE Simulator . . . 189
8.4.2 Methods . . . 190
8.4.3 Experiments . . . 193
8.4.4 Results . . . 194
8.4.5 Discussion . . . 196
8.4.6 Summary and Conclusion . . . 197
8.5 Multi-Objective Hybrid Hardware/Software PSE and DSE of an Embedded System . . . 198
8.5.1 Methods . . . 200
8.5.2 Experiments . . . 206
8.5.3 Measurement Setup . . . 208
8.5.4 Results . . . 209
8.5.5 Discussion . . . 223
8.5.6 Summary and Conclusion . . . 226
8.6 Summary and Conclusion of the MOGEA-DSE Method . . . 227

9 Conclusion and Future Work **231**
9.1 Conclusion . . . 231
9.1.1 Conclusion for VirusDetectionCL . . . 232
9.1.2 Conclusion for RFC-Gen . . . 232
9.1.3 Conclusion for SOG-PSE and SOG-DSE . . . 233
9.1.4 Conclusion for MOGEA-DSE . . . 233
9.1.5 Broader Context . . . 234
9.1.6 Limitations . . . 235
9.1.7 Further Use . . . 235
9.2 Future Work . . . 236
9.2.1 Proposal of the MP-MSP430-Sim Approach . . . 238
9.2.2 Proposal of the GOOM Approach . . . 239

Acronyms **243**

List of Mathematical Symbols **249**

Bibliography **253**

A Appendix **273**

CHAPTER 1

Introduction

Contents

1.1 Systems for Biological Virus Detection 1
1.2 Automatic System Optimization 4
1.3 Contribution of this Work 7
1.4 Organization of the Thesis 10
1.5 Author's Contribution to this Thesis 10
1.6 Acknowledgment 11

This work presents methods for a fast automatic computer vision-based detection and counting of biological viruses and methods of how the parameter space and design space of cyber-physical systems can be explored with the aim to find suitable systems for virus detection. The focus is on a multi-objective, energy-aware exploration of graphics processing unit architectures with suitable hardware and software parameters for biological virus detection scenarios.

1.1 Systems for Biological Virus Detection

Rapidly detecting viruses is of huge importance, "faster, portable and more accurate diagnostic methods" [MRE09] are needed. To make this possible, systems for a virus detection have to be explored and developed. Basically, a system for an automatic detection of biological viruses can be constructed from three parts: a sensor, a detection software, and a processing hardware. In this thesis, a novel virus detection sensor is considered, a novel computer vision-based detection method is developed, and suitable processing hardware is explored.

The sensor can make individual viruses indirectly visible by using only common optics and a simple setup. This sensor is named PLASMON ASSISTED MICROSCOPY OF NANO-SIZED OBJECTS (PAMONO). The computer vision-based detection method presented in this work, is able to handle the very demanding sensor data and can automatically detect and count individual viruses attaching to the PAMONO sensor. The hardware needs to be able to process the sensor images from the PAMONO sensor fast enough with the detection software. GRAPHICS PROCESSING UNITS (GPUs) have shown to be very well suited for that task as they are designed to process huge amounts of data in parallel. GPU hardware of different size and processing power is considered. To utilize the processing power of the GPU hardware, GENERAL-PURPOSE COMPUTING ON GRAPHICS PROCESSING UNITS (GPGPU) is used. With GPGPU, tasks that are usually processed on CENTRAL PROCESSING UNITS (CPUs) can be processed on GPUs. Sensor, detection

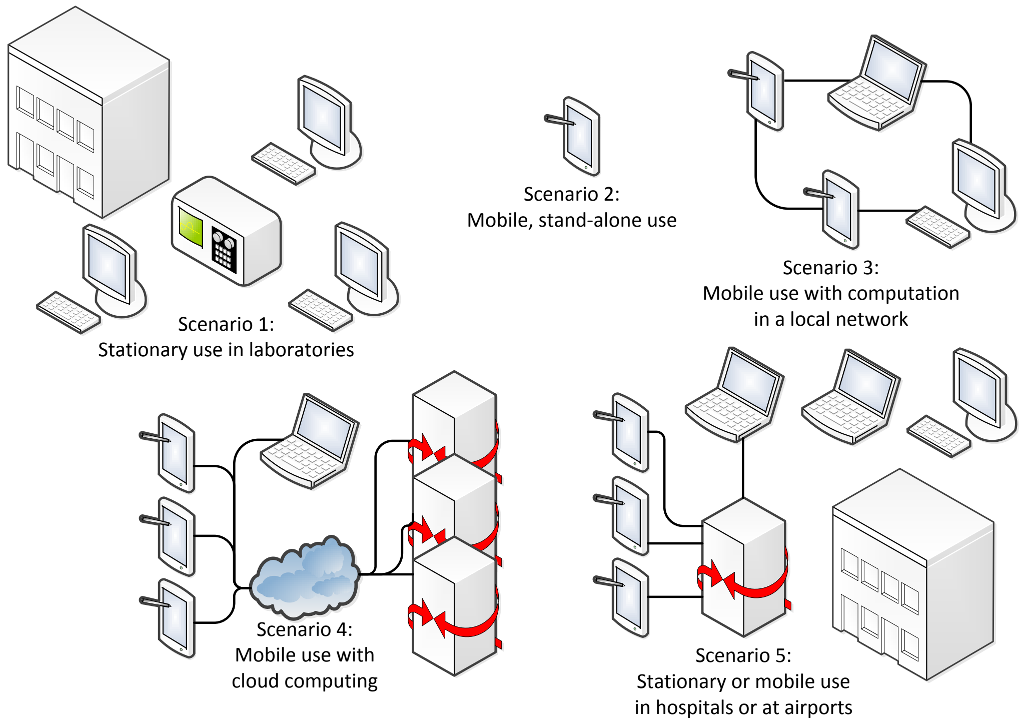

Scenario 1:
Stationary use in laboratories
Scenario 2:
Mobile, stand-alone use
Scenario 3:
Mobile use with computation in a local network
Scenario 4:
Mobile use with cloud computing
Scenario 5:
Stationary or mobile use in hospitals or at airports

Figure 1.1: Different scenarios for the detection of biological viruses.

software, and processing hardware combined form a so called Cyber-Physical System (CPS). CPSs are "integrations of computation and physical processes" [Lee07].

The Parameter Space Exploration (PSE) and Design Space Exploration (DSE) methods from this work can be used to explore these CPSs. PSE is the exploration of different parametrizations toward one or more objectives. DSE is the exploration of design alternatives, e.g., different software designs, design of new hardware, suitable existing hardware architectures, or hardware configurations toward one or more objectives.

The overall problem is to identify suitable CPSs for all scenarios of how the PAMONO sensor can be used in practice. A Pareto-optimal selection of software parameters, hardware architectures, and hardware configurations has to be found.

Therefore, the common thread running through the thesis consists of the different PAMONO scenarios. Figure 1.1 shows the five scenarios for the PAMONO sensor use case that are inspected in this work. These are named Scn1_{Lab}, $\text{Scn2}_{\text{Stand-alone}}$, $\text{Scn3}_{\text{Local}}$, $\text{Scn4}_{\text{Cloud}}$, and $\text{Scn5}_{\text{Hospital}}$.

Scenario Scn1_{Lab} shows the laboratory use case. Stationary desktop systems are used independently without need of a central server.

Scenario $\text{Scn2}_{\text{Stand-alone}}$ shows mobile processing with a single, independent hand-held device. This case can happen if a small, hand-held device is used and no connection to a server or other devices is available. Without the possibility to offload work, all calculations have to be done locally without consuming too much energy. By taking approximate computing into account, detection quality can be traded off against energy consumption.

Scenarios $\text{Scn3}_{\text{Local}}$ shows mobile processing with other available PAMONO devices in a local network. The other devices are usually battery driven but also devices with a power supply may be available. Calculations can be offloaded to other devices if the battery in the mobile devices runs low.

Scenario SCN4$_{\text{CLOUD}}$ shows mobile processing with the possibility to offload work to the cloud where the calculations are done on one or more servers. A mobile data connection like LONG TERM EVOLUTION (LTE) is used.

Finally, scenario SCN5$_{\text{HOSPITAL}}$ shows a typical setup for hospitals or airports. Several stationary or mobile devices can be used to take measurements where needed. Data can be processed locally or offloaded to a server in the local network. A server is available, e.g., to provide additional processing power for the mobile devices as needed.

For each application scenario the system has several degrees of freedom, and different, possibly conflicting requirements need to be fulfilled. The degrees of freedom and the requirements define an optimization problem.

As soon as the system exceeds a certain level of complexity, usually, no easy solution to this problem exists. Manually exploring the space of possible solutions can be very time-consuming or even infeasible. To address these issues, automatic PSE and DSE methods are examined in this work.

For the virus detection application, throughout the development process, regular changes were made on the physical setup of the sensor, on the used hardware in the sensor, on the capabilities of the detection software, and on the target hardware. Ideally, changes in one part of the system are reflected by the other parts within a short time. This requires steady changes of the system and its parametrization and an automatic evaluation of the objectives. With the PSE and DSE methods from this work, a feedback can be automatically obtained that loops back into the development process. The methods enable a hardware/software codesign and can be used in a SOFTWARE-IN-THE-LOOP (SIL), HARDWARE-IN-THE-LOOP (HIL) [Mar11], or HIL simulation [San02] manner.

Figure 1.2 shows an example of how such a hardware/software codesign with feedback might look like for the virus detection. Systems for the four considered examples are shown, i.e., systems for using the sensor in laboratories, hospitals, as a mobile device, and as a hand-held device. Each system has different demands to the sensor, needs different processing methods and parametrization and needs different processing hardware. Each line represents one CPS and its configuration. Over time the sensor and the detection software are improved. As new fields of application and use cases arise or if new hardware becomes available the target hardware changes. Even after the design of a CPS is completed the demands to the system can change dramatically depending on the use case, e.g., regarding detection quality, processing speed, energy consumption, and size of the hardware.

With the PSE and DSE methods an automatic optimization can be performed both on the software configuration and the target hardware. Additionally, bottlenecks and needed changes can be detected and treated early in the development. For example, for the virus detection use case it could be shown in this work that the noise in the sensor data is a limiting factor for detecting smaller viruses. This resulted in an improved physical setup of the PAMONO sensor with better cameras and in improved noise reduction and signal restoration methods for the detection software.

Overall, the PSE and DSE methods in this work can be used to automatically explore software configurations and hardware architectures toward several objectives. This is demonstrated on a real world use case, the detection of biological viruses.

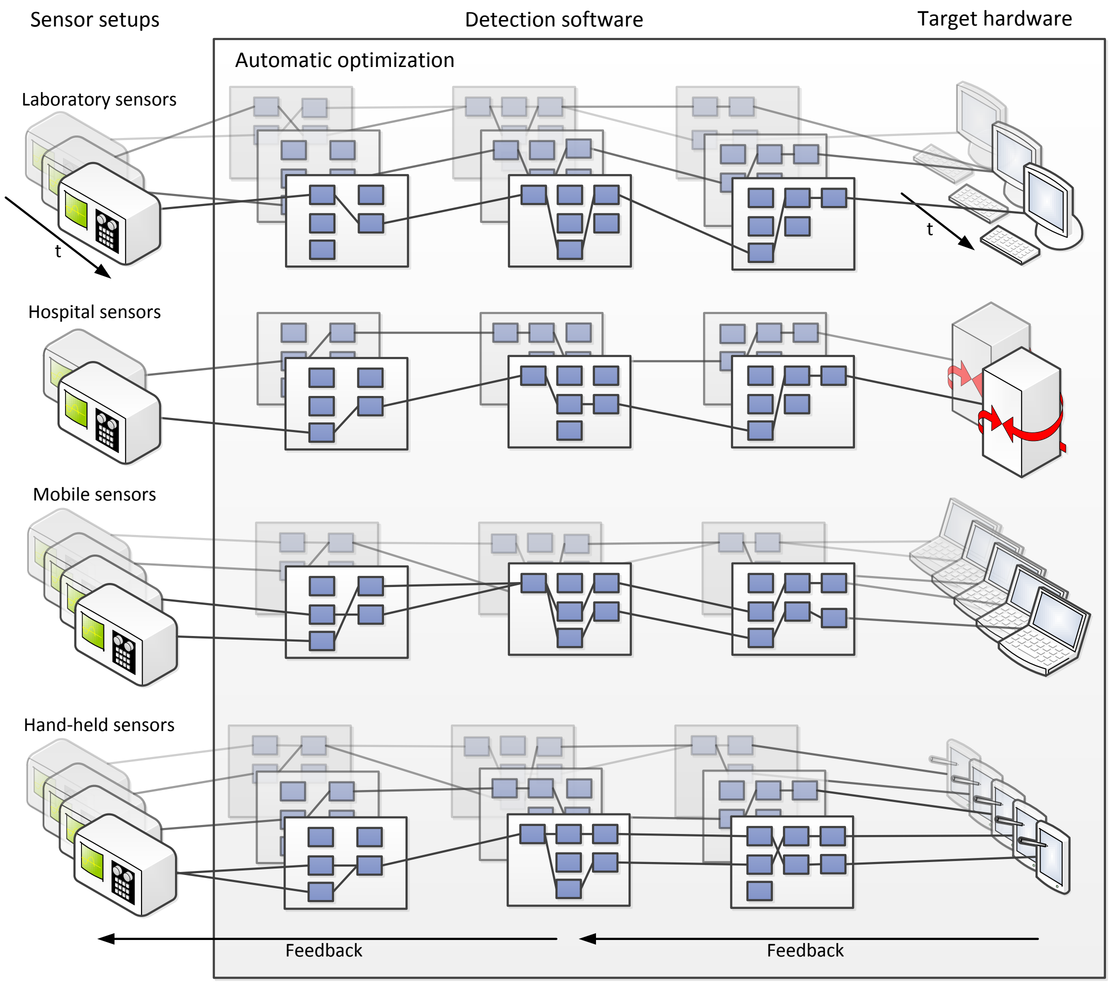


Figure 1.2: Design process for four systems for laboratory, hospital, mobile, and hand-held sensor data processing. The systems share the same processing algorithms but with different sensor setups, algorithmic configurations, and target architectures. Over time (t) the systems are changed. Every change requires that the system reflects these changes. The methods from this work can be used to automatically explore suitable configurations and suitable target hardware and can provide feedback for further improvements.

1.2 Automatic System Optimization

The PSE and DSE methods in this work can be used to explore CPSs for the virus detection use case and many other use cases. In our digitized world CPSs, which bridge the gap between the physical and digital domain, are becoming more and more important. Their importance has grown significantly in the recent years and is still increasing especially in the context of Health 3.0 [Nas08] and Industry 4.0 [BFK+14].

In this work, the focus is on heterogeneous CPSs including GPUs as target architectures. The size of the examined systems ranges from large High-Performance Computing (HPC) systems, to desktop, laptop, down to hand-held devices. Heterogeneity and concurrency play an important role for all these systems: additional powerful processors, power efficient processors, special purpose processors, or GPUs are included, e.g., for signal processing or computer vision tasks. MultiProcessor System-On-Chips (MPSoCs) and GPUs open up entirely new possibilities for CPSs. However, at the same time, the complexity to design software and to explore suitable parameters and architectures for these systems increases, especially if several possibly conflicting objectives and additional constraints need to be satisfied.

GPGPU enables the execution of all kinds of tasks on GPU hardware that was originally designed to process only graphics. With GPGPU, the processing power of GPUs is made available for various applications. To utilize the huge amount of processing cores of a GPU, concurrency is essential. As Sutter stated in 2005: "Concurrency is the next major revolution in how we write software." [Sut05] And up to now, everything indicates that he was right. Even for hand-held devices concurrency is becoming more and more important.

However, concurrency does not come for free. As an example, a simple minimum and maximum calculation on an array of numbers takes about six lines of *C* code, but an optimized minimum and maximum calculation on a GPU takes about 50 lines of GPGPU code plus about 90 lines of host code. This includes a global and local parallel reduction, efficient use of local memory, synchronization barriers, several kernel calls, and error handling. The initialization and cleanup of the GPU memory and the GPU itself is not included in this calculation.

In addition to the overhead of concurrency in the software design, GPGPU code and GPU hardware need to fit to each other. Actually, a GPGPU programming language should abstract from the hardware whereas in practice both are interdependent to a greater degree. This is especially true if the systems need to be optimized toward performance or toward a low energy consumption.

With the PSE and DSE methods the mapping to the architecture and the target architecture itself can be automatically explored and new insights to massively parallel programs can be gained.

Identifying a well suited combination of parameter values can be a difficult task. The link between parameter and design space and the objective space can be highly non-linear. Known or unknown dependencies might exist among the parameters.

A manual exploration is usually time-consuming and costly to conduct. Even if only a few variables are considered, the explored space can grow very quickly making a manual exploration infeasible. Already a few considered variables with a few possible values can span a space of thousands of points. If dependencies exist between these variables it is usually not possible to tune just one variable after the other. Instead, the depending variables have to be tuned jointly. Even if a suitable parameter set is found, small changes that affect the explored space often require that at least parts of the PSE and DSE must be repeated. If, for example, the setup, the field of use, or demands to the quality of the output change, the architecture and software need to reflect these changes. Especially if hardware and software are designed simultaneously in a hardware/software codesign manner, frequent changes with the need of repeated evaluation are common. A hardware/software codesign is often mandatory for Embedded Systems (ESs) and CPSs [Mar11].

Figure 1.3 shows a general view of what kind of systems can be explored with the PSE and DSE methods from this work. Target architectures from large and powerful down to small and mobile can be explored. For each of these architectures, an exploration

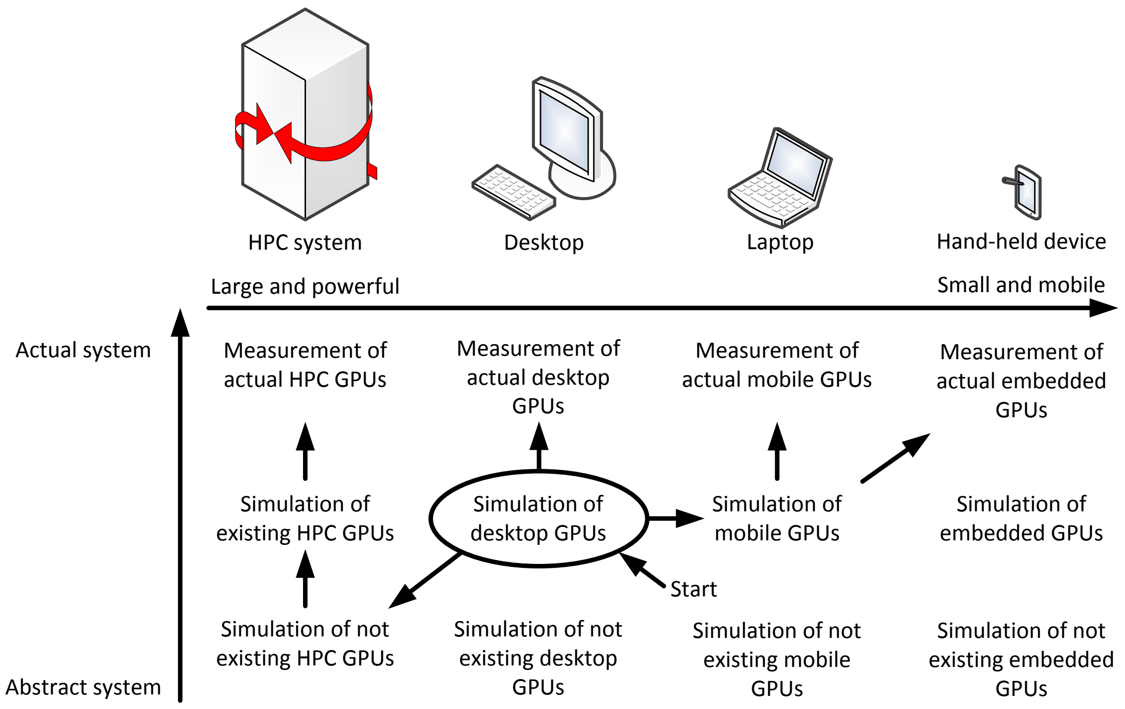

HPC system
Desktop
Laptop
Hand-held device
Large and powerful
Small and mobile
Actual system
Abstract system
Measurement of actual HPC GPUs
Measurement of actual desktop GPUs
Measurement of actual mobile GPUs
Measurement of actual embedded GPUs
Simulation of existing HPC GPUs
Simulation of desktop GPUs
Simulation of mobile GPUs
Simulation of embedded GPUs
Start
Simulation of not existing HPC GPUs
Simulation of not existing desktop GPUs
Simulation of not existing mobile GPUs
Simulation of not existing embedded GPUs

Figure 1.3: Overview of hardware systems of different size and with different abstraction levels and how they can be explored with methods from this work.

of these systems might start with a different abstraction level. For larger and more expensive architectures, the exploration might start with a high abstraction level and then continuously evolve toward the actual system. The abstraction levels might span from the simulation of not yet existing hardware, to simulation of existing hardware, and toward measurements of actual hardware. For a small and cheap architecture, the exploration might start directly with measurements of the actual device.

In addition to this separated view, an exploration might start with one system and from this basis the exploration leads toward other devices of larger and smaller sizes. This approach is indicated by the arrows in the figure with a desktop at a medium abstraction level as starting point. The size of the target architectures changes over time or if additional fields of application should be explored. Lessons learned from the exploration of architecture can be incorporated for the explorations of other architecture sizes. The systems can be explored in a hardware/software codesign manner. Software developed for one architecture can be reused and adapted to other hardware. Such a process of exploring target architectures of different size at different abstraction levels and suitable software toward actual systems can be supported with the PSE and DSE methods from this work.

As the complexity of computer systems increases, automatic methods are needed to explore these. Currently, Moore's law [Moo98] still holds: the number of transistors is raises exponentially. This trend also seems to hold for GPU systems. For example, the NVIDIA®GeForce™GTX Titan [NVI15d] has 7.1 billion transistors, twice as many as the NVIDIA®GeForce™GTX 680. With the increasing number of transistors the performance of these systems raises. However, this also leads to new challenges concerning a proper utilization of the available resources and concerning the energy consumption. Leng et al.

highlight that for the exploration of new GPU architectures, tools are needed to optimize the energy consumption of GPUs [LHE+13]. According to Rofouei et al. [RSR+08], there is “a huge potential of research in the field of energy-aware high performance computing with GPUs”. This motivates why automatic PSE and DSE methods such as these from this work are important.

1.3 Contribution of this Work

The original contribution to knowledge provided in this work consists of the five methods VirusDetectionCL, RFC-Gen, SOG-PSE, SOG-DSE, and MOGEA-DSE as explained in the following.

VirusDetectionCL

VIRUS DETECTION WITH OPENCL (VirusDetectionCL) is a method for an automatic detection and counting of individual viruses in PAMONO sensor data.

VirusDetectionCL is the first method that achieved an automatic detection and counting of individual viruses in soft real-time on small, hand-held devices. Soft real-time refers to a buffered processing of the recorded images that is on average fast enough to meet or exceed the speed of the camera.

This has been made possible by incorporating several existing and new methods for noise reduction, signal restoration, feature extraction, segmentation, and classification. The main focus of the methods is to achieve a good detection quality while at the same time the execution of VirusDetectionCL should reach or exceed the frame rate of the camera and be able to provide low energy consumption if needed.

The most important contribution with the VirusDetectionCL method is that virus infections can be analyzed and detected on a mobile, hand-held device in less than three minutes. The CPS of a mobile PAMONO sensor combined with the VirusDetectionCL method and a small, hand-held processing device opens up completely new possibilities for a virus detection that were not possible before.

The next important result is that sensor signals with a median SIGNAL-TO-NOISE RATIO (SNR) slightly below two can be handled. This enables the detection of smaller virus particles down to 100 nm. Most methods that are to some extent comparable to the VirusDetectionCL method can handle an SNR down to four [CWG01] or larger or equal to two [SLN+09].

Another important contribution is that the PAMONO sensor combined with the VirusDetectionCL approach reduces the diagnostic gap, between a sick person becoming contagious and the production of a sufficient amount antibodies, that can be detected. As soon as viruses in sufficient concentrations can be found within the sample, the PAMONO sensor can detect them, whereas with methods that detect antibodies, one has to wait until antibodies show up in sufficient concentrations within the sample. In contrast to the SURFACE PLASMON RESONANCE (SPR) sensor, which detects, like PAMONO, viruses and not antibodies, the PAMONO sensor can detect viruses in samples with lower virus concentration.

With VirusDetectionCL, an evaluation of the sensor images only takes about three minutes instead of several hours or even days, as with a manual evaluation by an expert. In result, the PAMONO sensor combined with VirusDetectionCL can detect acute infections earlier.

RFC-Gen

Random Forest Code GENeration (RFC-Gen) is a method for automatic GPU code generation for the application of Random Forest models. Instead of loading a Random Forest model to the GPU and evaluate it with fixed code, RFC-Gen generates combined GPGPU code for the model and the evaluation. Hence, the model and the evaluation become closely interwoven. Additionally, matching host code is generated, which provides the needed features to the GPGPU code.

The most important contribution for this method is that optimized GPGPU code for the evaluation of Random Forest models is generated. The RFC-Gen method is used within VirusDetectionCL.

SOG-PSE

Single-Objective GPGPU Parameter Space Exploration (SOG-PSE) is a single-objective PSE method for the exploration of parameters of given applications. A parameter space is defined by all combinations of parameter values that should be considered. The parameter space can then be explored toward an objective like quality of the result or execution time.

The SOG-PSE is the prior method of Single-Objective GPGPU Design Space Exploration (SOG-DSE) and Multi-Objective GPGPU Energy-Aware Design Space Exploration (MOGEA-DSE). Important insights could be obtained from this method. The need of a global optimization for VirusDetectionCL could be shown. This influenced the development of MOGEA-DSE. Additionally, the influence of the noise to the detection quality could be shown. This influenced both a further development of the PAMONO sensor and a further development of the noise reduction methods for VirusDetectionCL.

SOG-DSE

SOG-DSE is a single-objective DSE method for the exploration of GPU architectures for given GPGPU programs. SOG-DSE explores GPU architectures by making use of a GPU simulator and a GPU energy model. Different performance-, power-, or energy-based measures can be considered as objective.

The SOG-DSE is the prior method of MOGEA-DSE. The most important contribution for SOG-DSE is that GPU architectures can be explored automatically. A manual exploration of GPU architectures is time-consuming and expensive. With SOG-DSE, GPU architectures for laptop, desktop and HPC systems could be explored. For several GPGPU programs an optimized GPU architectures could be explored. Not only suitable GPUs could be found but also how they need to be configured. For VirusDetectionCL also an optimization of hardware related parameters was part of the evaluation.

For the exploration of HPC hardware, the power consumption has been considered. Such an exploration is important to integrate GPUs in existing HPC systems and to design new HPC systems. The power consumption of the GPUs is also important if a certain

Thermal Design Power (TDP) of a system has to be met. This led to new objectives for MOGEA-DSE.

Additionally, the need of multi-objective optimization and a speedup of the evaluation time for simulated GPUs became obvious with SOG-DSE. Both is considered in MOGEA-DSE.

MOGEA-DSE

MOGEA-DSE is a hybrid multi-objective PSE and DSE method for the simultaneous exploration of GPGPU software and GPU hardware.

The most important contributions for MOGEA-DSE are: it combines PSE and DSE to support a hardware/software codesign development process, it supports an energy-aware and multiple-objective DSE, it integrates two GPU simulators, it can take offloading of tasks into account, it integrates approximate computing, and it can perform measurements of actual ES hardware. As a unique feature, MOGEA-DSE is the first DSE method that can explore GPU architectures for HPC servers, desktop systems, mobile systems, down to ESs.

MOGEA-DSE can automatically explore hardware and software configurations toward the objectives execution time, energy consumption, and detection quality. This includes automatic measurements of the actual target hardware.

As a result, with MOGEA-DSE the soft real-time processing of VirusDetectionCL on ESs has been made possible. This enables a fast, accurate, and mobile virus detection with small, hand-held devices. A speedup of 4.1 and energy savings of 84 % could be achieved when compared to a baseline measurement that has been optimized for desktop systems. On the ES the same very good detection quality as on the desktop system could be obtained. The frame rate could be increased from 7.5 fps to 30.8 fps which is more than sufficient to enable soft real-time processing on the considered small ES.

Additionally, with MOGEA-DSE offloading of work to other devices and approximate computing can be taken into account. For offloading of work, different offloading points at different stages of the image processing pipeline of VirusDetectionCL have been examined. Some amount of work is processed on the local device on a mobile GPU and the remaining work is processed on a HPC server.

With offloading energy savings up to 95 % could be achieved. However, with full offloading the execution time increases from 0.5 s to 27.8 s. Therefore, also partial offloading has been explored. With partial offloading, 77 % energy could be saved while the execution time only increased to 0.87 s.

For approximate computing it has been examined, whether by allowing a reduced detection quality large savings in energy consumption and large speedups are possible for a CPS with an ES as target hardware. Without approximate computing with MOGEA-DSE 84 % energy could be saved and a speedup of 4.1 was possible compared to the baseline experiment. With approximate computing but still with a very good detection quality, the energy consumption could be reduced by another 50 % and a speedup of 8.1 was possible. By reducing the detection quality even further, but still reasonable for some use cases, a speedup of 12.0 could be obtained.

1.4 Organization of the Thesis

This thesis is organized as follows.

Chapter 2 introduces the PAMONO sensor. This provides the necessary biological and physical background and the application scenario for the VirusDetectionCL method.

Chapter 3 provides related work for the methods in this thesis.

Chapter 4 introduces basics for the following chapters. This includes an introduction to GPGPU, image processing notations and definitions, how the detection quality of the sensor system can be measured and quantified, an introduction to Evolutionary Algorithms (EAs), and a brief introduction to the SYNthesis/OPtimization/analySIS (SynOpSis) approach [SLW+14; Sie16].

Chapter 5 presents the VirusDetectionCL method for detecting and counting of individual viruses in PAMONO sensor data. This method is used for several experiments in the following chapters.

In Chapter 6 the RFC-Gen method is introduced. This method can be used to automatically generate GPU code for the application of Random Forest models.

Chapter 7 introduces the SOG-PSE and SOG-DSE methods. The first method is an automatic PSE for the exploration of parameters and the second method is a DSE for the exploration of GPU hardware architectures.

Chapter 8 is the most important chapter in this thesis. It introduces the MOGEA-DSE method. This method combines SOG-PSE and SOG-DSE to a hybrid multi-objective approach.

Finally, Chapter 9 provides summary, future work, and conclusion of this thesis.

1.5 Author's Contribution to this Thesis

In this section the publications of the author are presented in chronological order. For each publication the author's contributions are listed.

Parts of this work are built upon the author's diploma thesis [Lib11]. This includes parts of the description of the PAMONO sensor in Section 2.1 and a few parts of the VirusDetectionCL software in Chapter 5. Although most of the VirusDetectionCL method is new, some basic pre-processing, the marching squares algorithm, some basic polygon features, a simple threshold-based classification, the concept of the overall pipeline structure, and parts of the visualization are no new contributions.

Some preliminary work for this thesis has been done in cooperation with Timm [Tim12]. For this the author provided an early version of VirusDetectionCL as a use case. An exploration of work group sizes has been made possible in cooperation with Timm. The work from Timm clearly influenced the PSE and DSE methods in this thesis.

For the co-authored publication [SWL+11], the feature extraction methods have been developed in cooperation with Siedhoff. The classification, feature selection, and the evaluations have not been done by the author.

The publication [LWT12] is the most relevant for the SOG-PSE method. In this publication the denoising algorithms, the PSE, and the experiments have been conducted by the author.

For the publication [LST+13a], the fuzzy detection methods and the corresponding experiments have been developed and performed by the author. The other enhancement in the detection pipeline and the real time and energy measurements were done in cooperation. [LST+13b] is a German short version of [LST+13a].

For the co-authored publication [SLW+14], the SynOpSis approach was mainly devised by Siedhoff. However, the author provided the interface to VirusDetectionCL and some improvements on both SynOpSis and VirusDetectionCL are the result of this cooperation. The sensor model in [SLW+14] has been mainly developed by Siedhoff and by Zybin. The author implemented the application of the sensor model in VirusDetectionCL. This includes the parallel GPU implementation.

For the co-authored publication [SLW+14], the author only contributed to a lesser extent. The author provided VirusDetectionCL for the evaluations and implemented the sensor model.

The publications [LSW14; LMS+14; LKD+14; NLE+15] are the relevant publications for the MOGEA-DSE method. For the publication [LSW14; LMS+14], the author has contributed the DSE methods and constructed and conducted the experiments.

A cooperative paper has been done with the Communication Network Institute about offloading [LKD+14] where the author has done more than half of the work. The author integrated the LTE simulator from Dusza et al. into the MOGEA-DSE and set up and conducted all experiments.

A cooperative paper [NLE+15] has been done with Neugebauer. An equal amount of work has been contributed by the first two authors. The measurement software in Section 8.5.1 has been mostly developed by Neugebauer. The author provided interface, bug fixes, testing of the software and some small extensions and also conducted the experiments.

In the co-authored publication [STM+15] a specificity measurement of the PAMONO sensor has been conducted where the author provided measurements with the VirusDetectionCL method.

The methods for the VirusDetectionCL approach in Chapter 5 have been developed in cooperation with Siedhoff [SWL+11; LWT12; LST+13a; LST+13b; SLW+14; SFL+14]. Therefore, methods from VirusDetectionCL are also part of his Ph.D. thesis [Sie16]. The parallel design and the GPU implementations have been done solely by the author.

1.6 Acknowledgment

Part of this work has been supported by Deutsche Forschungsgemeinschaft (DFG) within the Collaborative Research Center SFB 876 “Providing Information by Resource-Constrained Analysis”, project B2.[1]

[1] http://sfb876.tu-dortmund.de

Chapter 2

Biomedical PAMONO Sensor

Contents

2.1 PAMONO Sensor 13

2.2 PAMONO Sensor Data 19

2.3 Summary 21

This chapter describes the Plasmon Assisted Microscopy Of Nano-sized Objects (PAMONO) sensor, invented by Zybin [Zyb10; Zyb13], and the PAMONO sensor data. The PAMONO sensor is a biomedical sensor that can be used to detect individual viruses. Fields of application are rapid detection of the spreading of diseases, medical diagnostics, research on viruses and antibodies, and development of new antiviral drugs.

This chapter is the basis for the automatic virus detection method in Chapter 5. Different application scenarios with the PAMONO sensor lead to the considered use cases in this thesis and are used to motivate the Parameter Space Exploration (PSE) and Design Space Exploration (DSE) methods in Chapter 7 and Chapter 8.

Parts of this introduction have already been published in the author's publications [Lib11; LMS+14] and the co-authored publication [STM+15].

2.1 PAMONO Sensor

The PAMONO sensor [ZBM+07; ZSS+16] is a biomedical sensor to detect viruses and other nano-particles in liquids. Viruses down to about 50 nm in diameter are detectable. In contrast to other bio-sensors, e.g., the Surface Plasmon Resonance (SPR) sensor, the PAMONO sensor is able to detect individual viruses. This enables the detection of viruses in samples with low virus concentration, more precise concentration measurements, and examination of binding processes in real time.

The PAMONO sensor has been developed at the Institute for Analytical Science (ISAS) by Zybin and is patented by European and American patents [Zyb10; Zyb13]. To detect individual viruses the sensor uses common optics and cameras. This is a unique feature of the PAMONO sensor. In general viruses are too small to be depicted with optical systems. How this is possible with the PAMONO sensor is stated in the following quotation. The PAMONO sensor is

> "depicting the radiation reflected by the sensor surface on the detector, characterized in that the resolution capability of the observation optics and of the detector is larger than the resolution that can be obtained by the deflection-limited radiation source." [Zyb13]

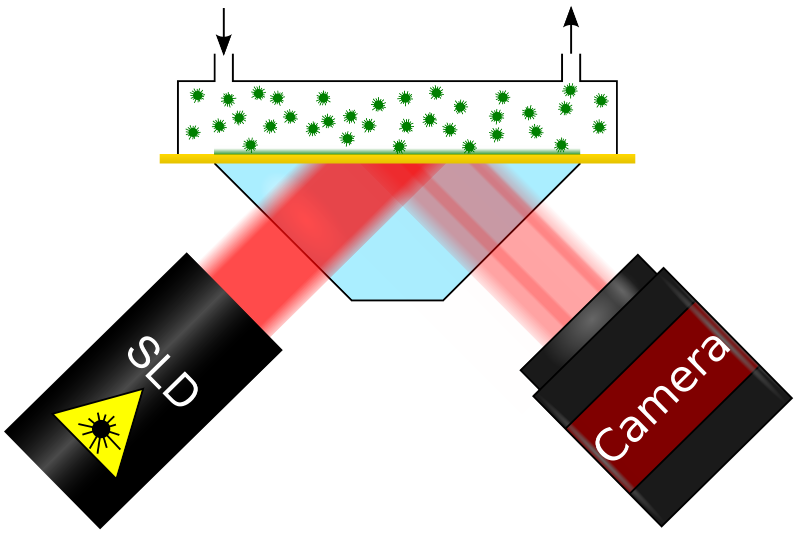


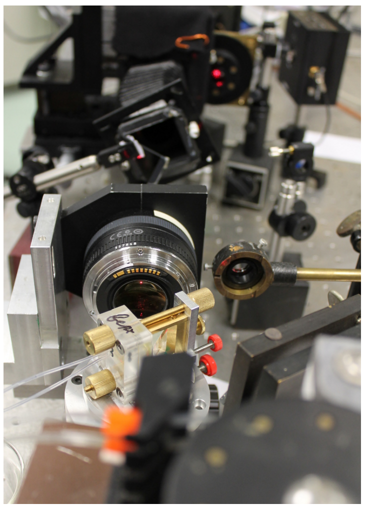
Figure 2.1: Schematic of the PAMONO sensor, with a Super-Luminescent Diode (SLD) on the left, the flow cell, gold plate, and prism in the center, and the camera on the right. Virus particles are flowing through the flow cell and possibly attach to antibodies, indicated by a thin coating on top of the gold layer. The reflected light is slightly brighter at positions where viruses have attached to antibodies. Lenses left and right of the prism have been omitted for reasons of clarity. Figure is based on [Lib11; LMS+14].

In other words, the sensor can indirectly make particles visible that are smaller than the resolution limit of the optics by making use of the secondary effect of the reflected light.

In the following the functioning of the PAMONO sensor is described. First it is described briefly and then in more detail. A schematic of the sensor is shown in Figure 2.1. The schematic shows the five main parts of the sensor: a light source, a prism, a gold plate, a flow cell, and a camera, which are also shown in the detail view. In addition to these parts, two lenses are used, which have been omitted in the schematic for reasons of clarity but are shown in Figure 2.2a. One lens is located between the light source and the prism for an even illumination of the sensor and the other between the prism and the camera for magnification and focusing.

Basically, the sensor detects viruses as follows. The gold layer is mounted on a prism, which is illuminated by the light source and reflects some light into a camera. The sample that should be tested is pumped through the flow cell on the other side of the gold layer. Viruses attach to the gold layer, which is covered with antibodies. Attaching viruses change the amount of reflected light on the other side of the gold layer. This change in intensity can be recorded by a camera. The images can be analyzed automatically by a software or manually by an expert and the signal of the attached viruses can be detected.

Figure 2.2 shows a laboratory setup of the PAMONO sensor. The individual parts of the sensor are mounted separately on a laboratory bench to support a fast reconfiguration. A mobile setup is shown in Figure 2.4. The mobile setup is much smaller. As it is not mounted on a stabilized laboratory bench, it is protected from vibrations with a stabilizing platform as shown in Figure 2.4a. The individual parts of the sensor are mounted closer


(a) An experimental PAMONO setup, which is mounted on a stabilized laboratory bench.

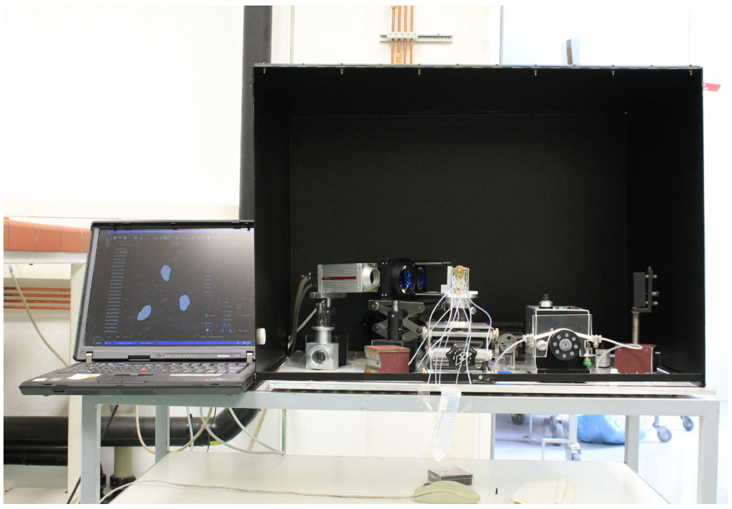
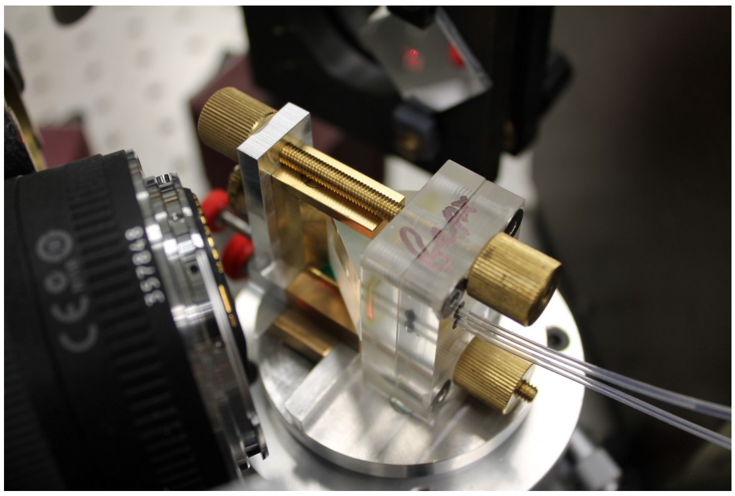

(b) First version of the mobile PAMONO setup consisting of a laptop for the analysis on the left and a (not so) mobile PAMONO sensor. The base area of this setup is 650 mm × 400 mm.

Figure 2.2: A fixed and a mobile PAMONO sensor setup. Both with SLD, mirror, rotary pump, the PAMONO sensor with the flow cell, lens, and camera.

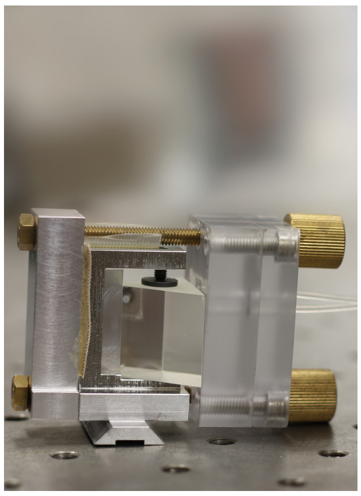
together as shown in Figure 2.4b. Because the setup is still in a prototype state, further miniaturization can be achieved easily.

In more detail, the sensor works as follows. The gold plate is usually covered with antibodies so that only desired viruses attach to the antibodies. The antibodies bound the viruses near the sensor surface, which is important for the detection. Only objects very close to the gold layer can be detected. The prism is reflecting the light into the camera. Samples are pumped through the flow cell, with 300 µl capacity [Zyb13], to the gold plate.

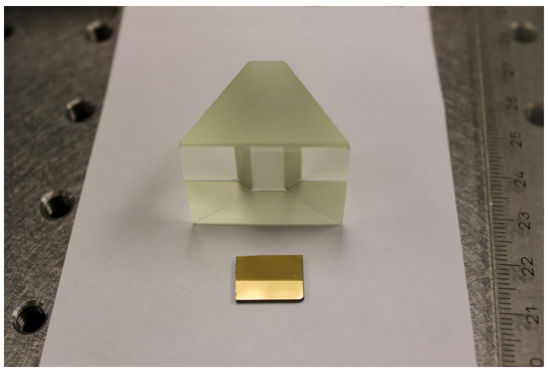
The sensor is illuminated by an SLD, see Figure 2.3d, or a laser diode. Both SLD and laser diode emit monochromatic light but an SLD causes fewer speckles in the light [Zyb13] and results in a more even illumination of the gold layer. A Kretschmann configuration is used for the arrangement of SLD, gold layer, and camera [SEM+11]. With the Kretschmann configuration the light causes so called surface plasmon waves on the surface of the gold layer to appear. Thus, some light is transformed to surface plasmon waves and is not reflected. How much light is transformed into plasmon waves and how much is reflected depends on the wavelength, thickness of the gold layer and the angle of light source and gold layer [ZSS+16]. In contrast to the classical SPR approach, the so called plasmon resonance or incident angle is not set to the minimum resonance but to a smaller angle [Zyb13]. If a virus or particle is attaching to the sensor, a shift in resonance occurs. The shift in resonance causes also a shift in resonance wavelength as shown in Figure 2.5. The wavelength can shift, e.g., from a to b. As a result, the reflected signal increases, e.g., from c to d. Although the virus is very small, for instance, only 100 nm, the change in reflected signal is on a larger scale, e.g., a few micrometers in size. This change in reflected light can then be detected with common optics. More details on the physics of the plasmon resonance effect is given in [Pat05; ZBM+07; ZSS+16].



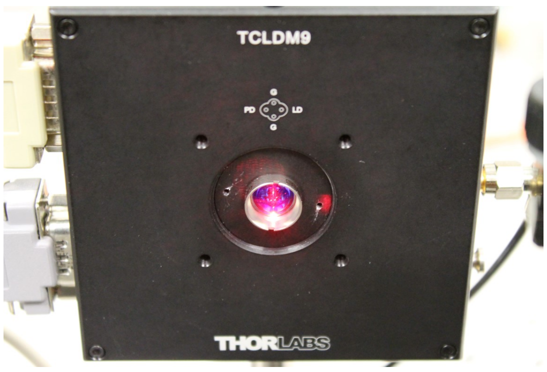
(a) Top view of the PAMONO sensor. It shows a mirror reflecting the SLD light to the sensor in the middle, a retro mounted lens for magnification on the left side, and inlet and outlet tubes on the right side.



(b) Lateral view of the PAMONO sensor with prism, gold plate (not visible), and flow cell.



(c) Gold plate and prism disassembled from the PAMONO sensor. A ruler is shown as scale.

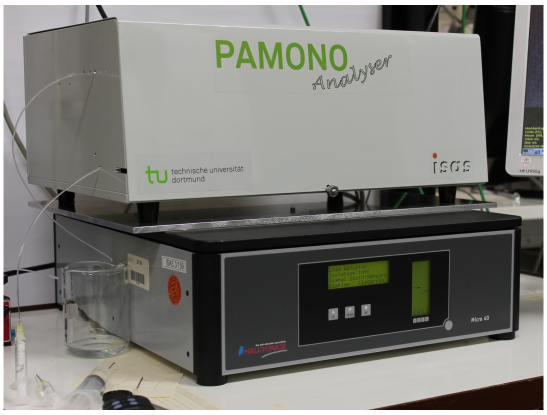


(d) A QPhotonic®QSDM-680-9 Super-Luminescent Diode in a Thorlabs®TCLDM9 cooled laser mount.

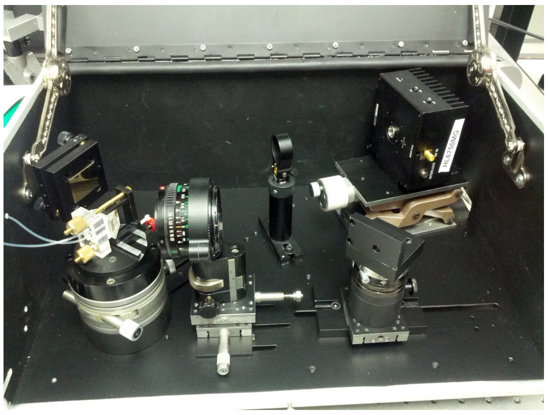
Figure 2.3: Detail views of the PAMONO sensor. Flow cell, gold plate, prism, and Super-Luminescent Diode (SLD) are shown.

Figure 2.6 shows the preparation of samples and how they are provided to the sensor. Samples can be prepared in test tubes as shown in Figure 2.6a. Afterward, they can be provided to the sensor with a rotary pump as shown in Figure 2.6b. The tubes are connected to the flow cell. If size and weight of the system matter, a syringe can be used instead of a rotary pump. However, a rotary pump has the advantage that it provides a more even flow.

The actual setup of the PAMONO sensor varied over time as it has been improved several times. The latest setup, as shown in Figure 2.4, uses a QSDM-680-9 SLD from QPhotonic® [QPh15] with a wavelength of 680 nm [Zyb13] and a power consumption of 0.64 W. A Thorlabs®LDC205C 500 mA 10 V laser diode driver [Tho15a; Tho15b], with a maximum power consumption of 30 W, operates the diode. The diode itself is fitted in a Thorlabs®TCLDM9 cooled laser mount [Tho15c; Tho15b], see Figure 2.3d.



(a) The mobile PAMONO setup placed on a stabilizing platform.



(b) Interior view of the mobile PAMONO setup.

Figure 2.4: Mobile PAMONO sensor setup. The base area is 450 mm × 300 mm.

The laser mount uses a Peltier element to keep the laser at a controlled temperature and has a maximum power consumption of 20 W. A Thorlabs®TED200C temperature controller [Tho15d; Tho15b] constantly measures the current temperature of the diode and accordingly drives the current of the Peltier element with a control range of −2 A to 2 A and a maximum output power of 12 W. The light of the SLD is altered with a zero order half wave waveplate for 670 nm [Tho15e]. A collimator lens is used to distribute the light evenly onto the sensor.

Several industrial monochrome cameras with Complementary Metal-Oxide-Semiconductor (CMOS) or Charge Coupled Device (CCD) sensor [The08] were tested and used within the PAMONO setup. The latest setup uses a The Imaging Source®DMK 23UP031 [The15] monochrome industrial camera. The camera has a 5 Mpx MT9P006 CMOS chip from Aptina™with 2.2 µm pixel size and a frame rate of 15 fps at full resolution. A second camera, currently in use, is the Allied Vision Guppy Pro F-503BC [All15a] with the same Aptina™MT9P006 CMOS chip as in the The Imaging Source®camera but with an IEEE 1394 interface. It has a frame rate of 13 fps at full resolution and a power consumption of 3 W.

Another camera, which was used for some older experiments, is the Allied Vision Prosilica GC 2450 [All15b]. The Prosilica camera has a monochrome 12 bit Sony®ICX625 CCD chip with 5 Mpx. It has a GigE interface and a frame rate of 15 fps at full resolution and a power consumption of 3.8 W.

Finally, the Kappa DX 40–1020 FW [Kap15] camera was used for early experiments, which is a 1 Mpx monochrome CCD camera with a frame rate of 16 fps at full resolution and a power consumption of 3.4 W. All three cameras can achieve a frame rate of 25 fps with a slightly reduced resolution.

The main reason to use industrial cameras is the bit depth of at least 12 bit. A 200 nm polystyrene particle causes a brightness increase of about 6 %. To detect this brightness change, a high pixel depth of at least 12 bit is recommended. With 8 bit depth as with consumer grade cameras, the greater quantization error would make a detection of a change in brightness more difficult.

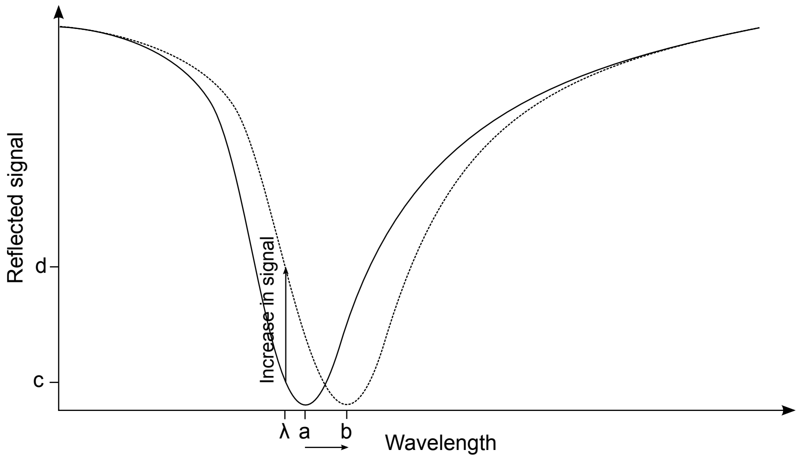


Figure 2.5: A resonance shift for an attaching virus or particle on the PAMONO sensor. The reflected signal at wavelength λ should be considered. If a virus or particle attaches to the sensor it shifts the resonance wavelength, e.g., from a to b. This causes a large increase in signal at wavelength λ, e.g., from c to d. The shift is exaggerated for clarity. Adapted from [Zyb13].

The necessary magnification of the image is achieved by a retro mounted lens, see Figure 2.3. A reversed mounted lens acts as a magnifying optics. The latest setup uses a retro mounted CANON®EF 50 mm $f/2.5$ compact macro lens [Can15], with modern anti-reflexion coatings, which makes the lens suited for the use with digital cameras. Older setups contained a retro mounted CANON®FD 24 mm $f/2.8$ lens from about 1970, see Figure 2.4b or a retro mounted MINOLTA®MD Rokkor 50 mm $f/1.7$ lens from about 1975. The anti-reflexion coatings of these two lenses were not designed to be used with digital cameras.

Figure 2.3 shows detail views of the parts of the PAMONO sensor. The core components of the sensor, prism and flow cell, are shown in Figure 2.3a and Figure 2.3b. The gold plate is mounted between prism and flow cell. A disassembled gold plate and prism are shown in Figure 2.3c.

The size for the latest mobile PAMONO setup is 450 mm × 300 mm × 200 mm, whereas older mobile setups were 650 mm × 400 mm × 500 mm in size. However, although it is already a mobile setup, the setup was designed to be used in the laboratory, in hospitals or at airports and not hand-held without any power supply nearby. Currently it contains a lot of empty space. As already mentioned, further miniaturization, toward a hand-held device, is possible.

What differs the PAMONO sensor from the SPR sensor is that for the SPR sensor only the overall brightness is measured, which is usually done with a single photo diode. As a consequence, only the overall amount of attaching viruses can be measured. For the PAMONO sensor on the other hand, the brightness changes from individual virus are measured with a camera. These changes in brightness are much smaller than the change of the overall brightness caused by thousands of viruses measured by the SPR sensor.

From the design of the PAMONO sensor several new challenges arise for the processing of the sensor signal. The main challenge is the low SIGNAL-TO-NOISE RATIO (SNR), cf. Definition 29. The signal of the viruses is only a little larger than the noise level of

(a) Samples are prepared in test tubes and can then provided to the PAMONO sensor.

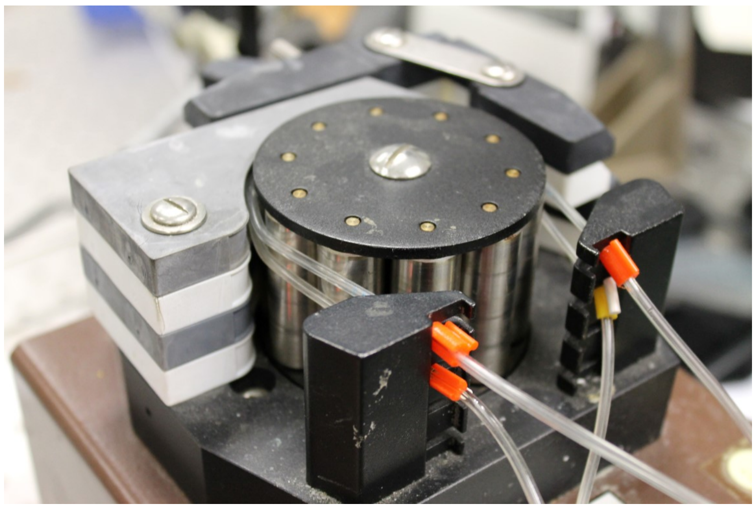

(b) The rotary pump, providing a constant flow rate of the sample to the flow cell of the PAMONO sensor.

Figure 2.6: Preparation of the samples and how they are pumped to the PAMONO sensor. The pump and test tube can be replaced by a syringe if the setup needs to be smaller.

the camera. The signal of the viruses is also much smaller than the empty background signal achieved from the sensor without any particles. However, if the challenges for the processing are solved, the PAMONO sensor can be used for a fast and simple detection and counting of individual viruses in laboratories, hospitals, at airports, and outdoors.

2.2 PAMONO Sensor Data

For the evaluation of the PAMONO sensor use case, several data sets are used. Basically, the data sets divide into signals from Virus Like Particles (VLPs), signals from polystyrene particles, and synthetic signals derived from real signals. Table 2.1 shows a list of the most important data sets. The names are used to identify the data sets in the following chapters, the type states if it is real signal recorded from the PAMONO sensor or generated synthetic signal, and the particle type describes which particles were used.

Figure 2.7 shows example images for four exemplary data sets of the list. The shown images were processed to remove noise over time and to remove the unwanted background signal to make the particles visible. A sliding window approach was used which calculates two images with reduced noise from 30 images for the background signal and 30 images for the particle signal. No spatial noise reduction has been applied. The average brightness has been increased to a medium gray level and the contrast has been enhanced with a gain factor of 15.

The data sets listed in Table 2.1 are described in the following. The data set $\text{Ds1}_{\text{HIV-VLPs}}$ contains signals from Human Immunodeficiency Virus (HIV)-VLPs. This is the most difficult data set that was analyzed. In Figure 2.7a the PAMONO signal

Table 2.1: List of PAMONO sensor data sets.

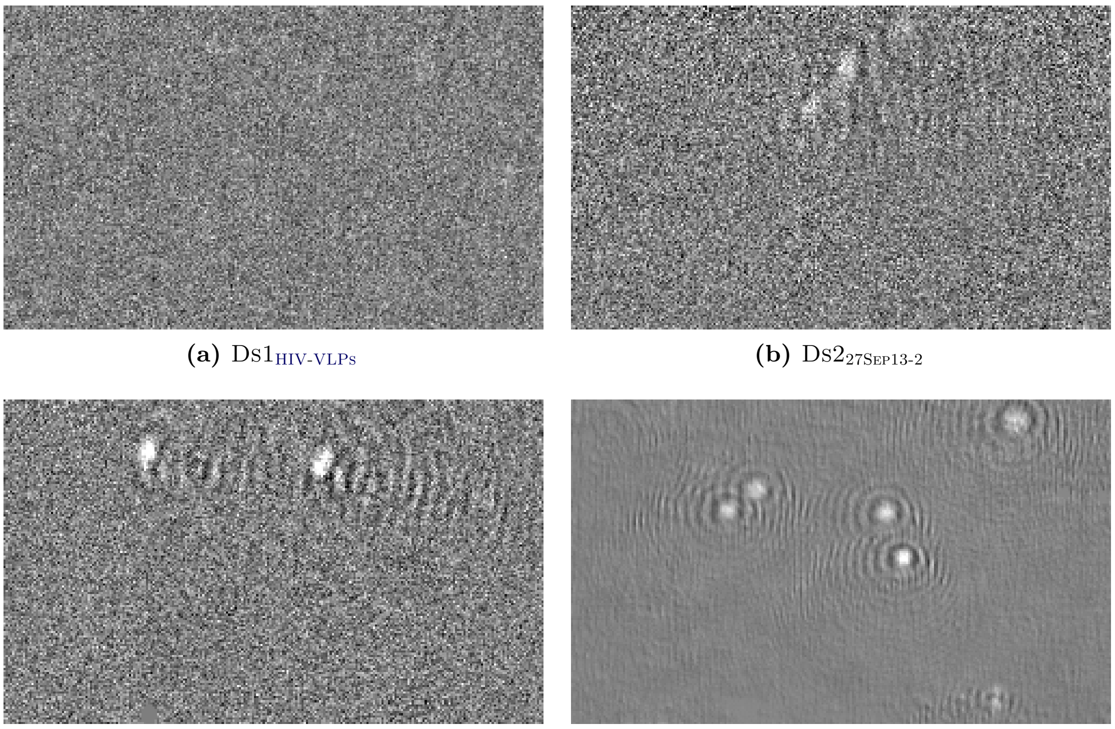
Data set name	Data set type	Particle type	Reference
Ds1$_{\text{HIV-VLPs}}$	Real	100 nm to 120 nm HIV VLPs	[ZKG+09]
Ds2$_{\text{27Sep13-2}}$	Real & synthetic	100 nm polystyrene particles	[SZS+14a]
Ds3$_{\text{27Sep13-3}}$	Real & synthetic	100 nm polystyrene particles	[SZS+14b]
Ds4$_{\text{10Apr13-1}}$	Real & synthetic	200 nm polystyrene particles	[SZS+14c]
Ds5$_{\text{11Apr13-1}}$	Real & synthetic	200 nm polystyrene particles	[SZS+14d]
Ds6$_{\text{11Apr13-2}}$	Real & synthetic	200 nm polystyrene particles	[SZS+14e]
Ds7$_{\text{Poly200nm-1}}$	Real	200 nm polystyrene particles	[LST+13a]
Ds8$_{\text{Poly200nm-2}}$	Real	200 nm polystyrene particles	-
Ds9$_{\text{Poly280nm}}$	Real	280 nm polystyrene particles	-
Ds10$_{\text{200nm-HQ}}$	Real	200 nm polystyrene particles	[Sie16]
Ds11$_{\text{200nm-MQ}}$	Real	200 nm polystyrene particles	[Sie16]
Ds12$_{\text{200nm-LQ}}$	Real	200 nm polystyrene particles	[Sie16]
Ds13$_{\text{100nm-HQ}}$	Real	100 nm polystyrene particles	[Sie16]
Ds14$_{\text{100nm-LQ}}$	Real	100 nm polystyrene particles	[Sie16]

with four very slightly visible HIV-VLPs is shown. Without further image enhancement the VLPs are very difficult to see. VLPs are used instead of real viruses because they can be handled without risk outside of a high-security laboratory. VLPs are very similar to real viruses but cannot replicate and are not pathogenic. They can attach to antibodies and PAMONO sensor signals from VLPs are indistinguishable [ZKG+09] from signal of real viruses. In [ZKG+09] further details on these VLPs can be found.

Each data set has been labeled by an expert. For every virus in the data set, a polygon is created that surrounds the virus and the frame number of the first occurrence is recorded. As the viruses are not visible on the raw sensor data, the images have to be pre-processed to make them visible.

It is very time-consuming to manually evaluate the PAMONO sensor data. Even for an experienced expert, it can take several hours to label a small data set with low virus concentration and up to one week to label a large data set with high virus concentration. If the used concentration of viruses in the sample is high, several thousand viruses attach to the sensor surface within just a few minutes. With a frame rate, e.g., of 25 fps, a 5 min recording contains 7500 images that need to be analyzed. Therefore, also synthetically generated data sets are used.

The data sets Ds2$_{\text{27Sep13-2}}$ to Ds6$_{\text{11Apr13-2}}$ contain 100 nm and 200 nm polystyrene particles. Polystyrene particles are much easier to handle as no antibodies are needed to bind them on the sensor surface. A slight electric charge of the particles is sufficient to bind them. These particles are commercially available in different sizes. Although the particles are made of a synthetic material, a resulting signal of the PAMONO sensor is almost indistinguishable from a signal of real viruses. Because of a missing antibody layer, the signal is slightly stronger than that of real viruses of the same size. Besides the directly recorded signal, these data sets also contain synthetically generated data generated by using the SYNthesis/OPtimization/analySIS (SynOpSis) approach [SLW+14], see


(a) Ds1$_{\text{HIV-VLPs}}$ **(b)** Ds2$_{\text{27Sep13-2}}$

(c) Ds5$_{\text{11Apr13-1}}$ **(d)** Ds9$_{\text{Poly280nm}}$

Figure 2.7: Example images of size 250 px × 150 px for different data sets. All data sets have been processed with the same settings to make the particles visible. In each image at least two particles are shown.

Section 4.5. For the synthetically generated signal, an accurate labeling was generated. Figure 2.7b shows four 100 nm particles from Ds2$_{\text{27Sep13-2}}$ and Figure 2.7c shows two of the 200 nm from Ds5$_{\text{11Apr13-1}}$. The data sets Ds2$_{\text{27Sep13-2}}$ to Ds6$_{\text{11Apr13-2}}$ have been made publicly available [SZS+14a; SZS+14b; SZS+14c; SZS+14d; SZS+14e] and the documentation for these data sets is also available [Sie14].

The data set Ds7$_{\text{Poly200nm-1}}$ contains 200 nm polystyrene particles. It is used to measure the influence of different image sizes to execution time and energy consumption.

Finally, two data sets with polystyrene particles Ds8$_{\text{Poly200nm-2}}$ and Ds9$_{\text{Poly280nm}}$ are used. These are the most easy data sets that are evaluated. They contain 200 nm and 280 nm polystyrene particles. For the data set Ds9$_{\text{Poly280nm}}$, a single processed image is shown in Figure 2.7d with six visible particles.

It should be especially emphasized that in the following chapters only the term virus is used, even though methods for the detection can also detect other particles like fine dust and in some experiments synthetic particles or VLPs are used. For the experiments it is made clear which data set was used.

2.3 Summary

In this chapter the PAMONO sensor has been presented. This sensor enables the detection and counting of individual viruses with common optics and in real-time. This is a unique feature of the PAMONO sensor. Viruses are usually too small to be detected optically. The PAMONO sensor, however, uses an indirect detection with help of plasmon waves and therefore making the effects of attaching viruses visible.

The PAMONO sensor has a relatively simple setup. A prism with a gold layer is the heart of the sensor. The gold layer is coated with antibodies to only bind the wanted viruses to the surface. Liquid samples with viruses are pumped through a flow cell and they can attach to the antibodies. The gold layer is illuminated with laser light, which causes plasmon waves in the gold layer. Attaching viruses disturb the plasmon waves in the gold layer. Thus, more light is reflected. Finally, the reflected light is recorded by a camera.

In this work different scenarios are inspected of how the PAMONO sensor can be used in practice. These scenarios motivate the methods that are introduced in the following chapters.

Chapter 3

Related Work

Contents

3.1 PAMONO Sensor 23
3.2 Image Processing, Computer Vision, and Pattern Recognition 25
3.2.1 Terminology 25
3.2.2 Image Processing 25
3.2.3 Computer Vision 26
3.2.4 Pattern Recognition 26
3.3 Random Forest and Code Generation 27
3.4 Optimization 28
3.4.1 Parameter Space Exploration 29
3.4.2 Design Space Exploration 30
3.4.3 Approximate Computing 36
3.5 Offloading 36

In this chapter the related work for the topics PAMONO sensor, image processing, medical image processing, computer vision, blob detection, particle tracking, Random Forest, code generation, optimization, PSE, DSE, and offloading is given. The order in which the related work is listed is the same as the topics appear in the thesis.

3.1 PAMONO Sensor

The PAMONO sensor for detection of biological viruses plays an important role throughout this thesis. This section provides the related work of the PAMONO sensor with focus on how it discriminates from other viruses sensors.

Very briefly, the PAMONO sensor [ZBM+07; ZSS+16], cf. Chapter 2, works as follows. It uses a gold surface coated with antibodies to bind viruses. The viruses are detected by indirect effects that requires no labeling of the viruses. These indirect effects are caused by plasmon resonance and enable the detection and counting of individual viruses. Details of the functioning of the PAMONO sensor are given in the patent description [Zyb10; Zyb13] and in [ZKG+09; Zyb10; GTÜ+11; STM+15; ZSS+16]. The physics of plasmon resonance is described in [Pat05; ZBM+07; ZSS+16]. Boecker et al. have described a signal model for PAMONO [BZH+07]. Binding of different particles to a spotted PAMONO sensor has been investigated by Scherbahn, Nizamov, and Mirsky [SNM16]. In the Ph.D. thesis of Siedhoff [Sie16] the PAMONO sensor is described in detail, a signal model has been designed, and methods for virus detection in PAMONO sensor images have been considered.

For the detection of viruses, other sensors than PAMONO exist. The most important sensors and methods are Surface Plasmon Resonance (SPR), Surface Plasmon Resonance Imaging (SPRi) [SS01], Polymerase Chain Reaction (PCR) [BS03], plaque assay [Dul54], and Enzyme-Linked Immuno-Sorbent Assay (ELISA) [GP13].

The SPR sensor and the SPRi sensor are the ancestors of the PAMONO sensor, with SPRi as the most similar sensor. Overall, the functioning of both sensors is likewise to the PAMONO sensor. All three sensors make use of the plasmon resonance effect and do not need labeling to make the viruses visible. A comprehensive overview about the plasmon resonance effect is given by Pattnaik [Pat05]. All three sensors can be used for concentration measurements. What discriminates the sensors is that SPR and SPRi are not able to detect individual viruses as they measure the intensity change of attaching viruses on a much larger scale than PAMONO. Basically, SPR and SPRi measure the overall intensity change caused by all attaching viruses combined. With the intensity change over the whole area, a rough assessment of the number of viruses can be obtained. In contrast to this, PAMONO measures the small intensity change of individual viruses. The image processing and detection for PAMONO is a more challenging task than for SPRi.

SPRi has found wide application and has been presented by several authors [BLB+08; CMF+04; GBH+99; NSB+03; SEM+11]. A survey on SPR-based sensors was done by Steiner and Salzer. They state [SS01] that SPRi-based methods are especially important because they show specific bounds of unlabeled molecules under in-situ conditions. The same holds for the PAMONO sensor.

Naimushin et al. have shown how an SPR sensor can be made portable [NSB+03]. They used a Peltier module with a fan and a temperature controller to keep the temperature of the sensor within a range of $23.5\,°C \pm 0.03\,°C$. Another interesting result is that they could double the binding of Staphylococcus to antibodies by increasing the temperature from $23\,°C$ to $42\,°C$. As each binding of antibody and antigen has an optimal temperature range at which it binds best, this can be part of further research. Likewise, there might be a temperature range for the PAMONO sensor at which the sensor and the detection software give faster or better results.

PCR [BS03] amplifies virus Deoxyribonucleic acid (DNA) using a polymerase enzyme. In each step the number of DNA strings that should be detected is doubled. This approach is very useful as it detects the virus DNA directly and does not rely on the detection of antibodies in the sample like in an HIV rapid antibody test. RiboNucleic Acid (RNA) can also be detected with PCR with an additional pre-processing step.

Plaque assay [Dul54] is a long established method for concentration measurement of viruses in a sample. Basically, a prepared layer of cells in a Petri dish is infected with the viruses in the sample. After about 24 h the infected cells can be seen macroscopically as plaques because surrounding cells are also infected. However, besides the time-consuming process, this is limited to virus types that cause detectable change in the infected cells. [Dul54]

ELISA [GP13] is a widely used and highly sensitive method to detect antibodies or antigens in samples. It binds the antibodies or antigens to a surface and then detects them by using an enzyme-mediated color change. For the detection of antigens, a capture antibody is used to bind the antigen, then a second, enzyme-labeled antibody is bound to the antigen, which can then be detected. For the detection of antibodies in the sample, an antigen is used to capture the antibodies in the sample. Afterward, a second, enzyme-labeled antibody is bound to the first antibody, which can then be detected. [GP13]

In summary, several other methods for the detection of viruses exist. Some have certain advantages over PAMONO. For example, PCR can detect the DNA or RNA of viruses and plaque assay can observe viruses in cells. However, no method combines the advantages of PAMONO: the fast, label-free detection of individual viruses.

3.2 Image Processing, Computer Vision, and Pattern Recognition

The VIRUS DETECTION WITH OPENCL (VirusDetectionCL) method in Chapter 5 for the automatic virus detection is mainly in the context of medical image processing, computer vision, and pattern recognition. Therefore, the focus of this related work is on medical image processing, computer vision, and blob detection. Blob detection and particle tracking methods are especially in focus as image processing methods that are most related to the virus detection task.

Target platforms for processing are GRAPHICS PROCESSING UNITS (GPUs) on HIGH-PERFORMANCE COMPUTING (HPC), desktop, laptop, and EMBEDDED SYSTEMS (ESs), which are inspected in Chapter 7 and Chapter 8.

3.2.1 Terminology

Image processing, computer vision, and pattern recognition are closely related terms that can not be strictly separated from each other. Chen and Kumar et al. discriminates the terms as follows [Che14; KSR+16].

For image processing usually the steps image enhancement, restoration, and segmentation have to be performed. Often, the goal of image processing is to extract important features from the image data. The results are then interpreted by an expert.

In computer vision also the steps image enhancement, restoration, and segmentation have to be performed. But in contrast to image processing, computer vision also interprets the results. Often, the goal of computer vision is to achieve the same results as an expert.

Pattern recognition is also closely related to image processing and computer vision. Basically, the steps in pattern recognition are feature extraction, pattern description, learning, and decision making. The main goal of pattern recognition is to make correct decisions or classifications based on the patterns and regularities in the signal. [Che14; KSR+16]

3.2.2 Image Processing

The most related image processing methods for the PAMONO virus detection are medical image processing methods. Medical image processing has been found wide application in the research community [Ban14; BF14; SG14; Sin14; TJ15; KSR+16]. The field of medical image processing is still gaining importance [EDF+13; KSR+16].

A well written survey on medical image processing on GPUs has been done by Eklund et al. [EDF+13]. They show several applications for GENERAL-PURPOSE COMPUTING ON GRAPHICS PROCESSING UNITS (GPGPU) medical image processing: optical imaging, microscopy, ultrasound, medical physics, image registration, image segmentation, image denoising, computer tomography, positron emission tomography, magnet resonance imaging, and diffusion-weighted imaging.

A survey on medical image processing with fuzzy techniques, important for Section 5.3.4 and Section 5.4.4, is presented by Chaira [Cha15].

3.2.3 Computer Vision

Same as medical image processing, computer vision has been found wide distribution and great importance in the recent years [FP03; Mor04; SHB08; Che14; Kle14] and for mobile computing computer vision is expected to be "the next big thing" [BRF15].

Image processing and computer vision using GPGPU has been presented by Fung and Mann in a survey [FM08]. They give a more broad view of the use of GPUs.

For the performance of computer vision applications on ESs, which plays an important role in this thesis, Backes, Rico, and Franke state that computer vision applications are usually very demanding for an ES [BRF15]. They have investigated different computer vision applications on ESs [BRF15]. They also inspected how Open Computing Language (OpenCL) can be used to optimize the *KinectFusion* computer vision application running on Mali GPUs of the Odroid-XU3 and an Exynos 5250 Arndale®board and on different inputs.

Sinha et al. perform GPGPU feature tracking and matching in video data [SFP+11]. Frame rates and image sizes are comparable to the PAMONO sensor images. However, the field of application is completely different, as they do feature tracking by calculating and matching of interest points in natural scenes.

3.2.4 Pattern Recognition

The most related pattern recognition methods for the PAMONO virus detection are blob detection and particle tracking methods. Both are very similar tasks to the PAMONO virus detection as small particles need to be identified in gray-scale images. In contrast to these, in PAMONO images the particles appear and then do not move, particles are not visible without any pre-processing, and they show some wave like patterns around the blob like structure in the center.

For blob detection and particle tracking, several publications exist. Comprehensive surveys on particle tracking methods have been done, for example, by Chenouard et al. [CSC+14] and Meijering, Dzyubachyk, and Smal [Mei12; MDS12].

Blob like structures that show similar characteristics to the blobs in single, preprocessed PAMONO images are those in breast ultrasound images. Moon et al. proposed a method for an automatic tumor detection. They use convolution of partial derivatives of a Gaussian distribution with the input images to build their features for blob detection [MSB+13]. Related to this is the work of Liu, White, and Summers [LWS10]. They also use derivatives of a Gaussian distribution for their blob features. They use their method to detect blobs in natural scenes and conclude that it can be used for the detection of blobs in medical images.

Blobs in images from fluorescence microscopy also show some characteristic of the blobs in PAMONO images. A survey on methods for tracking of single fluorescent particles has been done by Cheezum, Walker, and Guilford [CWG01]. They compared different tracking algorithms under realistic conditions. The methods focus on the tracking and not on the detection of the blobs. Interesting are the results for the needed SNR. SNRs of down to 1.3 have been examined but the four inspected algorithms could only successfully process images with SNRs down to four. However, for the PAMONO signal even SNRs below two need to be handled for the most difficult data sets.

Smal et al. [SLN+09] have considered different spot detection methods for live-cell fluorescence microscopy in a survey. Different supervised and unsupervised detection methods, for example, the use of Haar features [JZK+07] for a blob detection in fluorescence microscopy signal, have been evaluated on synthetic and real sensor data with different SNRs. They state that most of the classical methods break down at an SNR below four or five. For the better methods they show, that these are useful for SNRs of two. As a conclusion they recommend supervised machine learning methods for SNRs of two and lower, however, the unsupervised methods are more easy to use in practice and should also be considered. Olivo-Marin use a translation invariant wavelet transform to detect blobs of a certain size in fluorescence microscopy images [Oli02].

In general, this leads to the conclusion that, even if tracking of particles seems to be a very similar problem to the analysis of PAMONO images, particle tracking methods can not easily be transferred to the PAMONO context because the particles do not move and can have a lower SNR. However, some blob detection methods have also shown to be useful to calculate spatial features for PAMONO.

3.3 Random Forest and Code Generation

In Chapter 6 the code generation method Random Forest Code GENeration (RFC-Gen) is presented that generates GPU code for the application of Random Forest models. Therefore, this section lists the related work for Random Forest and for code generation.

Sharp [Sha08] has presented a Random Forest implementation for the GPU, using pixel shaders. A pixel-wise object class recognition task is performed. Both, training and classification have been done on the GPU. Similar to the author's approach, the forest is evaluated for every pixel position. However, a two-dimensional texture is used to store the trees, with two loops every thread is traversing all trees. As the trees are stored in global memory, this causes unnecessary global memory accesses from the different threads.

Another GPU Random Forest implementation, named CudaRF, has been presented by Grahn et al. [GLL+11]. As its name suggests, it is implemented with Compute Unified Device Architecture (CUDA). They parallelized the training and the classification phase. The main difference to the author's RFC-Gen approach is that they use one GPU thread to classify a single tree within the forest. To calculate the result for the whole forest, the classification results for the trees are downloaded to the host memory and the result is calculated on the Central Processing Unit (CPU). In the author's approach the result is calculated on the GPU and kept on the GPU. Another difference is that trees need to be uploaded to the GPU using the global memory, whereas in the author's approach trees are part of the code and fixed. As the same forest is used to classify various queries the forest does not need to be uploaded to the GPU several times nor be loaded into the local memory of the Streaming MultiProcessors (SMPs).

Code generation for GPGPU has been done by Klöckner et al. [KPL+12]. It is a scripting approach, which uses a high-level scripting language to generate CUDA and

OpenCL code, named PyCUDA and PyOpenCL. Another approach for the generation of GPGPU code is the SAGE approach by Samadi et al. [SLJ+14]. They automatically generate CUDA code with various levels of approximation. To speed up the calculation by scarifying Quality Of Result (QoR), their method stores input buffers with fewer bits, skips threads and interpolates the results from these threads, and selectively skips certain operations. As a result, QoR and performance can be traded against each other by the user.

3.4 Optimization

This section provides related work for optimization in general and in particular for the Parameter Space Exploration (PSE) and Design Space Exploration (DSE) methods Single-Objective GPGPU Parameter Space Exploration (SOG-PSE), Single-Objective GPGPU Design Space Exploration (SOG-DSE), and Multi-Objective GPGPU Energy-Aware Design Space Exploration (MOGEA-DSE) in Chapter 7 and Chapter 8. It should be noted that PSE is often called parameter tuning in the literature. In this work the term PSE is preferred in accordance to the term DSE.

Optimization methods divide into exact and heuristic approaches. Here, heuristic approaches are considered, which in turn divide into problem dependent heuristic and metaheuristic methods. As the PSE and DSE methods in this work should be applicable to a wide variety of problems, metaheuristic methods are most suited in this context.

Optimization methods also divide into model-based and model-free methods depending of whether they use meta models or surrogate models. Additionally, they can also be divided in three categories [MA04] depending on if preferences for the optimization are specified a priori, a posteriori, or not at all. The methods in this thesis are mainly in the second category but a combined objective can also be formulated a priori.

Metaheuristics are the main field of stochastic optimization. They can find good solutions even it is unknown how an optimal solution might look like. A comprehensive overview about all kind of metaheuristics has been done by Luke [Luk13]. This mainly covers but is not limited to evolution-based methods.

Detailed surveys of optimization methods are provided by several authors [MA04; Meh05; CP07]. Marler and Arora, for example, compare several optimization methods like weighted global criterion, weighted sum, lexicographic method, goal programming methods, bounded objective function method, physical programming, normal boundary intersection method, normal constrain method, or several genetic algorithms. The authors conclude that Genetic Algorithms (GAs) can be effective in obtaining the Pareto-optimal set.

In this work the focus of single-objective and multi-objective optimization is on Evolutionary Algorithms (EAs). This set of algorithms is called Multi-Objective Evolutionary Algorithmss (MOEAs). Coello, Deb as well as Konak, Coit, and Smith provide a comprehensive description of multi-objective optimization with EAs and GAs [Coe99; Deb02; Coe06; KCS06]. Strength Pareto Evolutionary Algorithm 2 (SPEA-2) and Non-dominated Sorting Genetic Algorithm-II (NSGA-II) are both EA-based methods and are used for multi-objective optimization throughout this work. SPEA-2 has been developed by Zitzler, Laumanns, and Thiele [ZLT01] and NSGA-II by Deb et al. [DAP+00]. For NSGA-II also the extension Non-dominated Sorting

Genetic Algorithm-III (NSGA-III) [DJ14] is available. The motivation to use EAs is discussed in Section 4.4.1.

A survey on optimization of image processing is provided by Siarry [Sia09]. It covers optimization with GAs, EAs, particle swarms, artificial ants, Bayesian inference, Markov models, and several other methods.

Beume, Naujoks, and Emmerich have developed the S-Metric-Selection Evolutionary Multi-objective Optimization Algorithm (EMOA) (SMS-EMOA) approach [BNE07], which uses an S-metric to steer the direction of the multi-objective optimization. The approach is especially suited for more than three objectives and has the advantage that the population is guaranteed to convert toward the actual Pareto front. However, it has the drawback that the calculation of the S-metric is expensive. If this drawback is of any importance in practice, highly depends on how long the fitness evaluations take.

Deb, Mohan, and Mishra have proposed ε-MOEA [DMM03]. The ε-MOEA makes use of the ε-dominance. They divide the search space into hyper-boxes and each hyper-box can only be occupied by one individual, which ensures diversity.

Smith has studied multi-objective simulated annealing techniques in his Ph.D. thesis [Smi06]. The simulated annealing method is compared to SPEA-2 and NSGA-II. Bergstra and Bengio have presented a method for hyper-parameter optimization that uses random search [BB12]. As an interesting detail, they have shown that Grid-Search is less efficient than random sampling.

Nam and Park have compared simulated annealing with EAs [NP00]. The authors have shown that for some problems simulated annealing performs better than EAs and for other problems it is the other way round. They also state that simulated annealing has often the drawback of a long evaluation time. In conclusion, they state that simulated annealing is a powerful algorithm with good properties to solve multi-objective optimization problems.

3.4.1 Parameter Space Exploration

A PSE is a special case of an optimization task. With a PSE a parameter space is explored in order to improve the QoR. Eiben and Smith provide an introduction to parameter tuning methods [ES15]. They compared parameter tuning methods Evolutionary Strategy with Co-variance Matrix Adaptation (CMA-ES), Sequential Parameter Optimization (SPO), Relevance Estimation and VAlue Calibration on parameters (REVAC) for tuning of EA parameters. In general, all PSE methods work according to the principle of generate and test. New parameters are generated and then tested according to their quality. For each test the parameters are set and remain fixed during the run. This distinguishes parameter tuning from parameter control. With parameter control the parameters are tuned during the run.

Mantovani et al. compared different tuning algorithms for the optimization of Support Vector Machine (SVM) parameters [MRV+15]. This includes random search, grid search, GAs, particle swarm optimization, and estimation of distribution algorithms. Two parameters of an SVM have been tuned. They concluded that random search can provide good results, similar to the other methods and that no significant differences between GAs, particle swarm optimization, and estimation of distribution algorithms could be found. However, the problem of tuning two SVM parameters might be just to simple to determine differences in the methods.

A meta-learning PSE has been developed by Molina et al. [MVR+12]. They optimized two parameters of a tree-based learning algorithm on several data sets. They evaluated the accuracy of several meta-learning algorithms. However, only a single step in the PSE is evaluated. Their study is more a comparison of different learning algorithms than a PSE.

Akay and Karaboga used an artificial bee colony algorithm for a PSE [AK12]. They extended an existing artificial bee colony algorithm and compared their method to several other approaches like differential evolution, particle swarm optimization, EAs, and GAs.

Simulated annealing is also a well established method for parameter optimization [Smi06; BSM+08; LL08]. Simulated annealing mimics the physical process of cooling. New individuals are generated and accepted with a probability that is based on the fitness and the current temperature. Eiben and Smith have shown that simulated annealing can also be seen as an evolutionary process [ES15].

An online parameter tuning method has been developed by Chau et al. [CTB+14]. In an offline step they optimize parameters for an object tracking algorithm with exhaustive search, enumerative search, or with a GA. Afterward, their approach uses, in an online step, information from the tracking context to select a previously tuned parameter set that matches the conditions.

Model-based optimization [JSW98; RKB+16] is another interesting method for PSE. It is especially suited for expensive evaluations. It is an iterative optimization that uses a regression model to steer the direction of the optimization. However, this method does not benefit as much as other methods from a higher degree of parallelization.

A sequential parameter optimization toolbox has been developed by [BLP10]. They use meta models to predict new points in the search space. The approach is mainly used to tune optimization methods like hidden Markov models or EAs.

The SynOpSis approach [Sie16; SLW+14], developed by Siedhoff, is a multi-objective PSE with different detection quality measures as objectives. It can simulate the considered sensor to synthetically generate training and testing data. The SynOpSis approach is also suitable for a PSE of VirusDetectionCL as has been shown in [Sie16] and the co-authored publication [SLW+14]. The SynOpSis approach is further described in Section 4.5.

3.4.2 Design Space Exploration

In contrast to PSE a DSE explores design alternatives and not parameters. In this work the focus is in particular on energy-aware DSE of GPU architectures and Cyber-Physical Systems (CPSs) in a hardware/software codesign manner.

DSE in General

In general, DSE methods divide into simulation-based and measurement-based approaches. In this thesis both simulation- and measurement-based approaches considered. A survey of different simulation-based and measurement-based approaches can be found in [AR12]. Ahmad and Ranka surveyed energy-aware computing in general.

Calborean and Vintan proposed Framework for Automatic Design Space Exploration (FADSE), a framework for DSE of multi-core architectures [CV10]. In

FADSE they integrated different multi-objective optimization methods and three simulators for simulation of multi-core architectures. They argued that automatic DSE is becoming more and more important if the number of cores in architectures rises. At the same time the complexity of the simulators and the simulation time rises. They also highlighted the importance of hardware/software codesign and that the processor design methods need to be adapted to the emerging challenges.

McIntosh-Smith et al. have observed that with the number of transistors also the power consumption of CPUs and GPUs increased exponentially until about 2004 [MWI+12]. From this date on, the power consumption was getting close to an upper bound. As the number of transistors is still increasing exponentially, a lower voltage was the main measure to keep the power consumption down to a reasonable level. However, as they stated the voltage is reaching a lower bound, too, so that the consideration of power consumption is increasingly challenging. [MWI+12]

Chis, Vintan, and Vintan compared different optimization methods for multi-objective DSE [CVV13]. The methods include SPEA-2, NSGA-II, a particle swarm optimization, and FADSE. FADSE includes the methods Controlled Elitist Non-dominated Sorting Genetic Algorithm-II (CNSGA-II) and Multi-Objective Cross generational elitist selection, Heterogeneous recombination and Cataclysmic mutation (MOCHC) and allows a distributed evaluation of individuals. The DSE was performed for a grid Arithmetic Logic Unit (ALU) processor, which enables asynchronous execution on a grid of ALUs. Five parameters of the grid ALU processor have been optimized and as objectives they chose clocks per instruction and hardware complexity.

Field, Anderson, and Eder have developed the Energy Aware COmputing Framework (EACOF) for energy-aware software design [FAE14]. With EACOF, several energy measurements can be gathered and combined and then provided to several consumers. How an energy measurement is performed is encapsulated in a provider, which reads and processes the energy information from a data source.

Dosanjh et al. considered DSE of exascale systems in a hardware/software codesign manner to achieve the next generation of supercomputing [DBD+14]. They explored the best CPU and memory for given applications. They conclude that validated hardware/-software codesign tools are needed and that simulations can provide valuable insights regarding the best configurations.

Ferrandi et al. use Ant Colony Optimization (ACO) for a DSE of heterogeneous ESs [FLP+10]. They considered mapping, scheduling, and communication of different applications. A model of the target architecture must be provided as input. As only objective, they focused on overall execution time. An interesting aspect of their work is that they used a model of a MultiProcessor System-On-Chip (MPSoC) developed on a Field Programmable Gate Array (FPGA).

To speed up a DSE, for example, the following techniques are feasible. Jahr et al. developed a DSE framework [JCV+12] that is based on NSGA-II and includes additional fuzzy rules to guide the optimization. With the fuzzy rules, expert knowledge can be incorporated, which can lead to a speedup. Ozisikyilmaz, Memik, and Choudhary use machine learning techniques to speed up their DSE [OMC08]. Deb, Zope, and Jain have shown how distributed computing can be used to calculate Pareto-optimal solutions with EAs [DZJ03]. Distributed optimization is important if a local evaluation is not feasible, e.g., if the local computer does not provide enough processing power. They have proposed a method of how several processors can calculate only parts of the Pareto solution, with only a minimum amount of overlap and without missing points by the partitioning.

Fei et al. compare multi-objective optimization methods for wireless sensor networks [FLY+16]. The most important objectives for wireless sensor networks are energy efficiency, network lifetime, and reliability. The authors state that in the past often only one objective has been optimized and the others have been treated as additional constraints. This may be unfair or unrealistic. Instead, multi-objective optimization should be preferred.

DSE for GPUs

For DSEs of GPUs, performance is often the only objective. However, several authors [RSR+08; MWI+12; LHE+13] emphasize that power or energy consumption should also be taken into account as this is becoming more and more important. Therefore, in this work power and energy consumption are used as additional objectives alongside with performance.

For the simulation-based approaches for GPUs, GPGPU-Sim [BYF+09] by Bakhoda et al., GPUWattch [LHE+13] by Leng et al., and GPUSimPow [LLA+13] by Lucas et al. play an important role. The GPGPU-Sim simulator is a simulator for GPGPU. It is clearly the most influential simulator in the field in the recent years. Bakhoda et al. use their GPGPU-Sim simulator to explore different GPU design parameters but only targeting execution time, not energy consumption. Both, Leng et al. [LHE+13] and Lucas et al. [LLA+13] extended GPGPU-Sim with a model for the power consumption. The first mentioned is called GPUWattch and the second GPUSimPow. However, these approaches are no automatic DSEs. An exploration of different target platforms has to be performed fully manual. All three methods, GPGPU-Sim, GPUWattch, and GPUSimPow, are used within the automatic, simulation-based DSE in this thesis.

Another power and performance model for GPUs has been developed by Hong and Kim [HK10]. Their method is called Integrated Power and Performance (IPP). In contrast to GPGPU-Sim, GPUWattch and GPUSimPow it does not rely on cycle accurate simulation to estimate the execution time of a GPU program but rather predicts the execution time with a timing model. They varied the number of active SMPs to inspect how this influences the power and energy consumption. They showed that the performance is saturated for some applications at a certain number of SMPs.

Kasichayanula et al. also developed a power model for GPUs [KTL+12]. They considered the power for the micro-architectures on the GPU. They built their models by measuring actual GPUs with micro benchmarks, which is similar to what has been done for GPUWattch and GPUSimPow.

A regression-based GPU DSE was performed by Jia, Shaw, and Martonosi in [JSM12]. Their approach is named Stargazer. Randomly selected GPUs from the design space are sampled, and from these samples a model of the execution time is constructed using step wise regression. An interesting statement from them is that a DSE for CPUs is usually inefficient or unsuitable for a DSE of GPUs. Mirsoleimani, Karami, and Khunjush [MKK14] also employ a linear regression model, which was used to find the most important GPU parameters affecting the run time of different GPGPU programs. Jooya, Baniasadi, and Dimopoulos [JBD13] presented an approach to find the GPU configuration that delivers the best execution time under a given transistor budget. It uses the Plackett-Burman design to set up the experiments and solves a knapsack problem. The most important difference between these three publications and the DSE in this thesis is that they do not considered an evaluation of energy consumption.

Mittal and Vetter provide a detailed survey [MV14] on methods that consider energy efficiency of GPUs in general. They state that for several applications the use of GPUs is more energy efficient than the use of CPUs or FPGAs and that considering energy efficiency of GPUs is very important. They categorize the methods into Dynamic Voltage and Frequency Scaling (DVFS) techniques, CPU-GPU workload division techniques, architectural techniques, techniques that exploit workload variations, and application specific and programming level techniques for improving the energy efficiency of GPUs. The SOG-DSE and MOGEA-DSE methods from this work belong to the categories architectural and application specific techniques.

Park et al. performed a power and energy-aware DSE for GPUs [PKS+13]. Their approach is measurement-based. They considered a simple GPGPU application as input and explored different mapping of work to the processing cores of one target GPU. This is a very simple DSE as only two hardware related parameters were optimized and the whole design space was sampled.

Liu et al. considered energy-aware task mapping to large scale GPU clusters [LDX+11]. They use a simple GPU energy model to estimate the energy consumption of the tasks before they are mapped to the GPU hardware. They optimize three hardware related parameters, i.e., dynamic voltage scaling, dynamic resource scaling, and partial execution of GPU sub-tasks on the CPU. Their objective is to save as much energy as possible while still meeting all given deadlines for the tasks. Compared to MOGEA-DSE this method is more a scheduling method.

An energy-aware optimization of streaming tasks on GPUs has been done by Maghazeh et al. [MBH+14]. They considered different mappings of tasks to the cores on the GPU. They optimized the task size and the scheduling of the tasks. For the optimization they use a heuristic that includes a data stream model and a simple energy model of the GPU. They evaluated two small GPU programs, an encryption program and a convolution, on an ES. Although it is an interesting method, the energy model is too simple to be useful for real world applications. It simply assumes that the energy consumption of the GPU decreases with increasing job size down until a certain level is reached. This works for their small programs because the programs do not utilize the GPU very well. The authors only considered job sizes of 10 kB for the encryption and 20 px × 20 px for the convolution and varied the number of jobs that are passed to the GPU from 1 to 100.

Cebrián, Guerrero, and García state that there is a need for new designs and optimizations to make GPGPU more energy efficient, especially for mobile GPUs as target platform [CGG12]. In their work they run eleven GPGPU programs on two NVIDIA®GPUs with different core clock frequencies. They measured the power consumption of the GPUs. A drawback of their work is that they measured only desktop GPUs although they stated that mobile GPUs should be especially in focus. Additionally, the limitations of a measurement-based DSE becomes obvious. Measurements are time-consuming and buying several architectures is also expensive. They only measured two GPUs and evaluated only a few, small programs.

Yi et al. [YTW+10] optimized a single application toward a given HPC GPU architecture. Their approach is measurement-based and they considered execution time and energy consumption as objectives. For their GPGPU application they achieved a speedup of 8.7 and could save 83 % energy compared to a parallel CPU implementation. However, all optimizations were performed manually. Therefore, this is more a case study on optimization techniques and not an automatic DSE.

Timm has presented a multi-objective DSE for an energy-efficient mapping of existing GPU code to a given GPU hardware platform [Tim12]. Two DSE methods have been presented Multi-OBjective Local Instruction Scheduling (MOBLIS) and Feedback-based and memory-Aware gLobal Instruction Scheduling (FALIS). GPU code is optimized with local and global instruction scheduling for the objectives energy consumption and execution time. Optimizations are performed with SPEA-2 and NSGA-II on the code level and evaluated with actual measurements of GPUs. A testbed has been developed. The explored GPUs need to be switched manually. Additionally, two manual explorations have been performed. The first manual exploration considered different work group sizes for a given GPU application on selected GPUs. The second manual exploration considered the power consumption of running programs either on the CPU or the GPU of a low power PC. What discriminates the MOGEA-DSE from the MOBLIS and FALIS methods is that with MOBLIS and FALIS GPU code is optimized and not the parameters or the GPUs. The considered hardware/software codesign is actually an automatic platform-based design with manual switching of target architectures.

Overall, MOGEA-DSE is the only GPU DSE method for the multi-objective exploration of CPSs from large HPC servers down to small, hand-held ESs. Additionally, MOGEA-DSE offers simulation-based and measurement-based evaluation, can integrate offloading strategies, and integrates approximate computing.

DSE for CPSs and ESs

For a DSE of CPSs and ESs, often more than one objective has to be optimized and additional constraints have to be met. This makes a DSE for these systems particularly challenging.

A DSE for CPSs has been done by several authors. Canedo and Richter developed a design automation tool for CPSs in the automotive sector [CR14]. They combine different simulation components to explore different architectures. Green computing solutions for distributed CPSs has been examined by Abbasi et al. [AJB+13]. A multi-objective rule-based DSE has been proposed by Abdeen et al. [AVS+14]. The authors also make use of NSGA-II to guide the optimization. They optimize a CPS for a smart building application. The authors integrate multi-objective optimization into a model-driven rule-based DSE framework.

A multi-objective DSE for System-On-a-Chips (SoCs) with GAs has been performed by Palesi and Givargis [PG02]. They map software to parameterizable SoCs to meet timing requirements and a power budget. To speed up the evaluation, clusters of interdependent parameters are explored and independently optimized using GAs or exhaustive search. In a subsequent step the results are merged and again optimized using GAs or exhaustive search. This is somehow similar to the two stage optimization in Section 7.3 but different to how the parameter dependencies are handled in Section 8.2.2. An application specific DSE for SoCs has been done by Ascia et al. [ACD+07]. They use EAs to explore configurations of a SoC platform. The presented approach can be on multiple objectives and includes software and hardware parameters in the optimization. Also, pruning of the design space and other speedup techniques. A multi-objective DSE of CPUs of SoCs combined with source code transformations has been done by Agosta, Palermo, and Silvano [APS04]. The energy consumption and performance approximation is done with the WATTCH simulator.

Pham et al. proposed a framework for DSE of MPSoCs [PSK+13]. They considered energy and throughput as objectives to map applications to MPSoCs. Their approach also include an online step where the generated mappings are used and refined at runtime. This approach only considers mapping and no hardware/software parameters. Shao et al. also proposed a framework for DSE of MPSoCs [SRW+14] but on a much lower level. They use a simulation-based approach to explore accelerator and memory hardware for given programs. They consider performance, power, and area as objectives.

An EA-based allocation and scheduling for an XMOS®MPSoC has been examined by Banković and López-García [BL15]. A more general approach has been done by Erbas, Cerav-Erbas, and Pimentel. They used EAs to map applications to MPSoCs [ECP06]. Castrillón et al. map Kahn process networks to MPSoCs [CTL+12]. Another DSE for MPSoCs has been done by Angiolini et al. [ACL+06]. Even wider, Cannella et al. compose MPSoCs according to multiple objectives by using EAs [CDF+11].

The Odroid-XU3 MPSoC, cf. Section 8.5, has also been used by Gensh et al. as a hardware platform for energy and performance measures [GAR+15] and by Backes, Rico, and Franke as a platform for computer vision applications for performance measures [BRF15].

Mariani et al. use additional meta models to predict which individuals should be evaluated [MPS+09; MBP+10]. They use NSGA-II for a multi-objective DSE of MPSoCs. An artificial neural network is included in the optimization to speed up the evaluation by building a meta model. An evolutionary strategy is used to decide, based on the output of the meta model, if a simulation should be performed or not. Another model-based DSE approach has been proposed by Basten et al. [BBG+10]. They map different tasks to CPU, GPU, FPGA, and one or more memories. They use throughput as objective. However, as a major drawback they only abstractly model the mapping on the hardware components. For every task the needed resources and dependencies are described in textual form. The hardware is also only described in textual form. The DSE is then performed on this abstract description.

Mohanty et al. presented a method for DSE of heterogeneous ESs [MPN+02]. Their method is a hierarchical approach from a very abstract DSE on a large design space down to a very detailed DSE on a smaller design space. Constraints are described with the OBJECT CONSTRAINT LANGUAGE (OCL). On the coarsest level of the DSE functional simulators are used to verify the functionality of the application. Then, high level power and performance estimators are used to reduce the design space. Finally, cycle accurate simulation is used on the most detailed level.

A DSE of ESs especially for image processing applications is examined by Keinert and Teich [KT11]. For a given image processing application, their approach works as follows: First, the data flow of the application is specified. Second, the system is verified. Third, the automatic DSE is performed and the memory is analyzed on a system level. Fourth, based on the previous steps a system is synthesized. Finally, the complete ES is constructed.

Although GPUs are very suited for many kinds of image processing applications, the authors did not consider GPUs on ESs as target architectures.

In contrast to the existing approaches, here, a multi-objective PSE and DSE is conducted, cf. Chapter 8, which can be used for different hardware and software configurations. Multiple and possibly conflicting objectives can be chosen, such as energy consumption, power consumption, number of core cycles, run time or the energy delay product. The presented optimization is done in a hardware/software codesign to identify hardware and software configurations that meet the constraints.

3.4.3 Approximate Computing

With approximate computing, two or more objectives, usually quality and performance, can be traded against each other by using more or less accurate computation. Typical approximate computing methods are presented in a detailed survey by Mittal [Mit16]. The author examined several methods: precision scaling, loop perforation, load value approximation, memorization, skipping tasks, skipping memory accesses, multiple inexact program versions, inexact or faulty hardware, voltage scaling, reducing branch divergence in Single Instruction Multiple Data (SIMD) architectures, and neural network-based acceleration.

Approximated computing with the objective of saving energy has been investigated by Baek and Chilimbi [BC10]. Based on an annotated source code of an algorithm and additional calibration data a quality model is built and a solution that meets the desired quality is provided to the user. They use loop and function approximation as methods for approximated computing.

3.5 Offloading

Offloading approaches for mobile devices are quite common. Kumar and Lu [KL10] have shown that cloud computing can save energy on hand-held devices, but not all applications are suitable saving energy using the cloud. They conclude that offloading approaches for mobile devices have to provide energy savings and this has to include also the overhead for privacy, security, and reliability.

Li, Wang, and Xu [LWX01] have shown how computation offloading can be used on hand-held devices that are connected via Wireless Local Access Network (W-LAN) to a server. An adaptive computation offloading for battery powered devices has been shown by Xian, Lu, and Li [XLL07].

Toma and Chen [TC13] have shown computation offloading for real-time tasks. Rudenko et al. [RRP+99] describe a framework to automatically migrate tasks to a server to receive the results. An introduction into how offloading decisions can be made on bandwidth data have been done by Wolski et al. [WGK+08]. They formulated computation offloading as a statistical decision problem, which incorporates different network bandwidth predictors. However, the local and remote execution time of tasks is supposed to be known by the scheduler.

Chapter 4
Basics

Contents

4.1 General-Purpose Computing on Graphics Processing Units . . 37
4.1.1 OpenCL 39
4.1.2 GPGPU Notations and Definitions 41
4.1.3 Efficient Access to Global Memory 45
4.1.4 Efficient Use of Local Memory 45
4.1.5 Additional Constraints 49
4.2 Image Processing Notations and Definitions 53
4.3 Quantifying the Detection Quality 57
4.4 Evolutionary Algorithms 59
4.4.1 Motivation to Use EAs 60
4.4.2 EA Definitions 60
4.5 The SynOpSis Approach 61

In this chapter basic explanations and definitions are provided, which are used in the following chapters. It is left to the reader whether this chapter is read beforehand or only upon demand as references are provided in the following chapters toward this chapter.

The structure of this chapter is as follows. In Section 4.1 an introduction GPGPU is given. This includes several definitions of terminology, an introduction to efficient processing on GPUs and also special characteristics of the GPU hardware. In Section 4.2 all needed definitions for the image processing methods in Chapter 5 are given. In Section 4.3 measures of detection quality for virus detection methods on PAMONO sensor data is introduced. In Section 4.4 an introduction to EAs and in Section 4.5 a brief introduction to SynOpSis approach is given.

4.1 General-Purpose Computing on Graphics Processing Units

General-Purpose computing on Graphics Processing Units (GPGPU) has become more and more important in the last years. With GPGPU, tasks that are usually processed on CPUs can be processed on GPUs. However, to utilize all cores of a GPU is a complex task. Modern desktop CPUs are equipped with up to ten CPU cores, with up to ten threads or 20 hyper threads, while modern GPUs are equipped with hundreds or thousands cores that can hold thousands of active threads. The cores in GPUs are called Streaming Processors (SPs), lightweight processing units. For example, the Intel®Core™i7-6950X [Int16], released 2016, is currently the only desktop CPU from Intel®with ten or

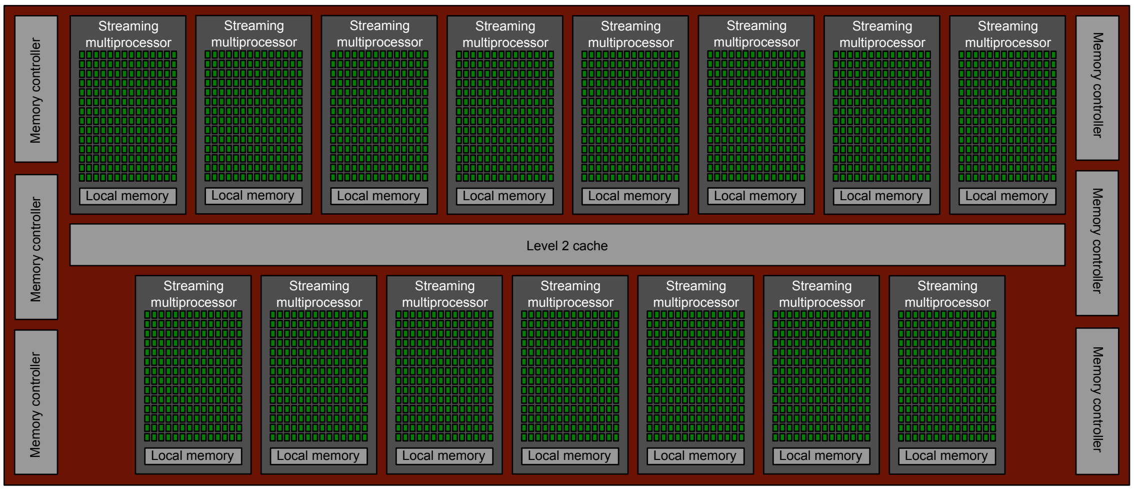


Figure 4.1: The NVIDIA®GK-110B KEPLER™architecture. A GK-110B GPU consists of 15 SMPs, each with 192 SPs and a local memory. For example, the NVIDIA®GEFORCE™GTX Titan Black is based on the GK-110B architecture. Adapted from [NVI12a].

more cores and the NVIDIA®GEFORCE™GTX Titan Black GPU has 2880 SPs [NVI15e; NVI12a]. Each SP is processing one lightweight GPU thread. SPs are grouped into SMPs, 15 SMPs in the example of the GTX Titan. SMPs are physical units not just logical units. Within one SMP, all SPs share a local memory, some constant memory, and a program counter.

An example of an NVIDIA®GK-110B KEPLER™architecture [NVI12a] is shown in Figure 4.1. This architecture, as used in the NVIDIA®GEFORCE™GTX Titan Black, consists of 15 SMPs, each with 192 SPs and a shared local memory. In total, 2880 SPs are available for processing.

A GPU architecture like this is especially suited to work with the SIMD programming paradigm. The same instruction is performed on different parts of the data. For example, for image processing each image can be partitioned into rectangular parts and each part can be mapped and processed on different SMPs. On the SMPs the image can be partitioned again and mapped to the SPs. On the SPs one thread processes the assigned chunk of data. Each of the threads executes the same code, for example a convolution, but on different sets of input and output data.

Several challenges for GPGPU exist [NVI09b; NVI15b]:

- The algorithms need to be parallelized in order to utilize the SPs.
- It needs to be specified how the work is distributed, an optimal distribution might differ on different hardware.
- The available hardware structures, for example the local memory, should be utilized if possible.
- As global memory accesses need to be minimized as they are expensive, with respect to needed cycles and energy consumption.
- Access patterns to global memory need to match underlying hardware.
- Threads within one work group may need to be synchronized, especially if local memory is used.

- Usually global memory cannot be synchronized within one kernel call. If a global synchronization is needed, the kernel needs to be divided and two kernel calls have to be scheduled one after the other.

This is only a small excerpt of existing challenges for GPGPU, several other constraints need to be fulfilled which are not listed here. To mention another example, threads on NVIDIA®GPUs are organized in so called warps and half warps. The threads within one half warp can work concurrently without synchronization. Memory accesses can be executed combined into one single operation if all threads within one half warp access the memory in the right manner. As memory accesses are executed in aligned segments of size 64 B, 128 B, or 256 B on modern GPUs, the threads within one half warp should access the memory in such a way that the least amount of segments are needed [NVI15b]. All memory operations that fall into the same segment can be executed coalesced. In Chapter 5 some solutions to these problems are presented. The following sections introduce OpenCL, needed GPGPU definitions, the OpenCL memory hierarchy, how it is used efficiently, and what other constraints exist in the OpenCL context.

This section is based on the OpenCL specification [Khr12] and on the OpenCL best practice guide [NVI09b].

4.1.1 OpenCL

This section provides an overview of Open Computing Language (OpenCL) and to which details particular attention should be paid. OpenCL is an open, portable, vendor independent parallel programming language specified by the Khronos™group [Khr09; Khr11a; Khr12; Khr14]. OpenCL programs can be executed, without modification, on GPUs, CPUs, Digital Signal Processors (DSPs), FPGAs, and some other accelerator hardware for desktop, mobile, and ES hardware. All GPU applications presented in Chapter 5 and Section 6.1 are solely written in OpenCL and are therefore portable to a huge amount of devices.

In the following the platform model, memory model, execution model, and programming model of OpenCL are introduced based on the OpenCL specification [Khr12].

Platform Model

The platform model of OpenCL consists of a host and one or several compute devices. A single compute device contains one or more compute units and each compute unit contains one or more processing elements. An OpenCL application runs on a host, which submits work to the devices which then execute the work on the processing elements. Single Instruction Multiple Data (SIMD) is the main programming concept of OpenCL, all processing elements execute the same instruction on different data. However, it is possible that different threads execute different paths. The second programming concept of OpenCL is Single Program Multiple Data (SPMD) where a single program is executed independently on different data on several processing elements.

Memory Model

The memory model of OpenCL consists of different layers: global memory, constant memory, local memory, and private memory. The first layer in this hierarchy consists of the global and constant memory accessible by all processing elements of the device. Usually, they are located outside of the compute units on the device, e.g., in form of Dynamic Random Access Memory (DRAM). The difference between the global and constant memory is that the constant memory is initialized by the host and only read access is possible by the processing elements. The constant memory may be part of the global memory and both might be cached by the device. If a cache should be used, can be decided by the vendor of the hardware.

The second layer is the local memory. It is shared among all processing elements within one compute unit. Usually it is located on the compute units.

The third layer is the private memory. It is only accessible by one processing element and is usually located on the processing elements, e.g., in form of registers.

In general, the global and constant memory are the largest but slowest memory within the memory hierarchy. Constant memory may be faster than the global memory. Typical sizes are 256 MB to 4 GB. The local memory is usually much faster than the global and constant memory and ranges from 16 kB to 64 kB, which has to be shared among several threads. Finally, the private memory is usually faster than the local memory and typically ranges from 4 kB to 64 kB. However, the size and speed of each layer of the memory hierarchy is not part of the OpenCL specification. The vendor of the device can decide how it is realized in hardware.

Execution Model

The host is used for initialization and clean up of the OpenCL context and for submitting work to the devices execute the actual program logic.

A function in OpenCL is called *kernel*. The programming language for the kernels is a reduced set of *C-99* without pointers and dynamic memory allocation and with some extensions for the parallel programming, e.g., for identifying the current thread and for synchronization.

A global index space has to be specified to execute an OpenCL kernel. The kernel is instantiated for every index in the global index space. This instance is called global work item. The index space in OpenCL can be one-, two-, or three-dimensional. As in Chapter 5 only two-dimensional image data is processed, in the following the focus is only on two-dimensional index spaces. The index space is evenly subdivided into so called work groups. Each global work item is usually executed by a thread on a processing element. Each work item is assigned to a work group and can be identified by the global index or by the index of the work group and the local index within the work group.

If the kernel is ready for execution it is submitted by the host to the command queue of the device. Also, other commands, e.g., for allocating memory on the device, are submitted to the command queue. For each device one or more command queue can be manged so that several programs can be executed in parallel. The device executes the commands in the command queues. For each queue it can be specified if it should execute the commands in order or out of order. A scheduling of the commands in the queue and a mapping of tasks to different SMPs is done by the device.

Programming Model

The programming model of OpenCL data parallel and task parallel execution with data parallel execution as the main programming model. Results in memory can be synchronized across processing elements in one compute unit or across all processing elements. For example, threads within one work group can be used to load a sub-image into the local memory that can afterward be read by all threads in the work group. Then a synchronization barrier has to be set to assure that all threads have finished loading parts of the sub-image into the local memory. With a barrier, threads in one work group need wait until all threads in the work group have reached the barrier before any thread continues. This prevents that threads read random memory values from memory that has not been loaded.

It is important to note that a global synchronization across all threads or between threads in different work groups is only possible by finishing a kernel call and submitting another kernel. It is not possible that threads in one work group can access results in the global memory, produced by threads in another work group within one kernel call even if global memory barriers are used. The global memory barriers only assure that the global memory is synced across the threads in the work group. Because the execution sequence of work groups is scheduled by the GPU and cannot be influenced by the programmer, it can not be guaranteed that a particular work group has written its results. Only if a kernel is finished the global memory is synchronized across all threads.

A synchronization of commands within the command queues is also possible either by barriers or by signals. These synchronizations are useful if kernels are executed out of order. Barriers assure that all kernels are finished at the barrier. While the barriers can only be used to synchronize commands within one command queue, the signals can also synchronize commands in different command queues as one queue can wait until an event from another queue is received.

In the following section the terms from this section are properly defined and the mathematical notation is introduced.

4.1.2 GPGPU Notations and Definitions

This section defines OpenCL and GPGPU terms, which have been introduced informally in Section 4.1.1. Additionally, some GPGPU notations are introduced to keep them consistent throughout this work. All OpenCL specific names, descriptions, and definitions are based on the OpenCL specification [Khr12]. They are introduced without further reference to the OpenCL specification.

The parallelism of OpenCL is based on a few concepts. Briefly, an OpenCL kernel is a function that can be executed on the OpenCL device. It describes the program code for a single thread. Different data can be processed for each thread. The overall index space defines how many threads should process the data. The index space is partitioned into work groups and the work groups are partitioned into single threads called work items. Each work item executes one instance of the OpenCL kernel. The work items are grouped into work items and each work item within a work group executes the same command. As every work item can be identified within the kernel, it can access for example different data in an SIMD manner.

Notation 1 (Width and Height)**.** The width and height of the input images shall be denoted as w and h. For reasons of simplicity, the width and height of the input images shall be implicitly given in the following.

Definition 1 (OpenCL Kernel)**.** An OpenCL kernel is a function that can be executed on an OpenCL device.

In the following the terms OpenCL kernel, kernel code, or simply kernel refer to OpenCL device code. OpenCL host code is explicitly labeled.

If an OpenCL kernel should be executed, a global index space needs to be defined. This provides also the number of global work items.

Definition 2 (Global Index Space and Global Work Item)**.** A two-dimensional *global index space* is defined by the width $w_{\mathrm{g}} \in \mathbb{N}$ and the height $h_{\mathrm{g}} \in \mathbb{N}$. For each (valid) index in the global index space, a *global work item* is created. A global work item is defined as an instance of an OpenCL kernel. The global work item at index (x, y) is denoted as $i(x, y)$. The set of all global work items in a global index space of width w_{g} and the height h_{g} can be described by the set S:

$$S := \{i(x, y) \mid x \in \{0, \ldots, w_{\mathrm{g}} - 1\} \text{ and } y \in \{0, \ldots, h_{\mathrm{g}} - 1\}\}. \tag{4.1}$$

For reasons of simplicity, w_{g} and h_{g} shall be implicitly given throughout this work.

In addition to the definition of a global index space, a subdivision of the global index has to be provided in order to execute an OpenCL kernel. With this subdivision the global work items are grouped to work groups. All work items within one work group execute the same code and share a local memory and the work group barriers for synchronization purposes. On a GPU they are executed on the same SP.

Notation 2 (Width and Height of a Work Group)**.** The width and height of a work group is denoted as w_{l} and h_{l}, with $w_{\mathrm{l}}, h_{\mathrm{l}} \in \mathbb{N}$. For reasons of simplicity, w_{l} and h_{l} should be implicitly given throughout this work.

Definition 3 (Two-Dimensional Work Group and Local Work Item)**.** For a given subdivision $w_{\mathrm{l}} \in \mathbb{N}$ and $h_{\mathrm{l}} \in \mathbb{N}$ with $h_{\mathrm{g}} \mod h_{\mathrm{l}} = 0$ and $h_{\mathrm{g}} \mod h_{\mathrm{l}} = 0$, the two-dimensional global index space can be equally divided. All global work items that are in the same subdivision form a so called *work group*. A work group is denoted as $W_{(t,u)}$ with $t \in \{0, \ldots, \lfloor w_{\mathrm{g}}/w_{\mathrm{l}} \rfloor - 1\}$ and $u \in \{0, \ldots, \lfloor h_{\mathrm{g}}/h_{\mathrm{l}} \rfloor - 1\}$. All work groups for the given subdivision are of the same size. The global work items within a work group are named *local work items*. The local work item at local index $(x_{\mathrm{l}}, y_{\mathrm{l}})$ of the work group (t, u) is denoted as $i_{(t,u)}(x_{\mathrm{l}}, y_{\mathrm{l}})$, with $x_{\mathrm{l}} \in \{0, \ldots, w_{\mathrm{l}} - 1\}$ and $y_{\mathrm{l}} \in \{0, \ldots, h_{\mathrm{l}} - 1\}$. A two-dimensional work group defined as the set $W_{(t,u)}$:

$$W_{(t,u)} := \{i_{(t,u)}(x', y') \mid x' \in \{0, \ldots, w_{\mathrm{l}} - 1\} \text{ and } y' \in \{0, \ldots, h_{\mathrm{l}} - 1\}\}. \tag{4.2}$$

The size of the set $W_{(t,u)}$ is $w_{\mathrm{l}} \cdot h_{\mathrm{l}}$.

Each global work item can either be accessed with the global index or with the work group index and the local index. This identifies the same thread, although it is called global work item and local work item. The indices of the local and global work items that are assigned to the same thread can be mapped to each other according to the following lemma.

Lemma 1 (Local Work Item to Global Work Item)**.** A local work item $i_{(t,u)}(x_\mathrm{l}, y_\mathrm{l})$ of the work group (t, u) with a width of w_l and a height of h_l can be mapped to the corresponding global work item $i(x, y)$ by setting

$$x := t \cdot w_\mathrm{l} + x_\mathrm{l} \text{ and} \tag{4.3}$$

$$y := u \cdot h_\mathrm{l} + y_\mathrm{l}. \tag{4.4}$$

Proof. The work group index (t, u) for the global work item (x, y) can be described as

$$t = \lfloor x/w_\mathrm{l} \rfloor \tag{4.5}$$

$$\Leftrightarrow t \cdot w_\mathrm{l} = \lfloor x/w_\mathrm{l} \rfloor \cdot w_\mathrm{l} \tag{4.6}$$

and

$$u = \lfloor y/h_\mathrm{l} \rfloor \tag{4.7}$$

$$\Leftrightarrow u \cdot h_\mathrm{l} = \lfloor y/h_\mathrm{l} \rfloor \cdot h_\mathrm{l}. \tag{4.8}$$

The local work group item $(x_\mathrm{l}, y_\mathrm{l})$ for the global work item (x, y) can be described as

$$x_\mathrm{l} = (x \mod w_\mathrm{l}) \text{ and} \tag{4.9}$$

$$y_\mathrm{l} = (y \mod h_\mathrm{l}). \tag{4.10}$$

The global work item (x, y) can also be expressed as

$$x := \lfloor x/w_\mathrm{l} \rfloor \cdot w_\mathrm{l} + (x \mod w_\mathrm{l}) \text{ and} \tag{4.11}$$

$$y := \lfloor y/h_\mathrm{l} \rfloor \cdot h_\mathrm{l} + (y \mod h_\mathrm{l}). \tag{4.12}$$

Replacing $\lfloor x/w_\mathrm{l} \rfloor \cdot w_\mathrm{l}$ with the left side of Equation (4.6) and replacing $(x \mod w_\mathrm{l})$ with the left side of Equation (4.9) results in Equation (4.3). Replacing $\lfloor y/h_\mathrm{l} \rfloor \cdot h_\mathrm{l}$ with the left side of Equation (4.8) and replacing $(y \mod h_\mathrm{l})$ with the left side of Equation (4.10) results in Equation (4.4). □

Definition 4 (Global Memory)**.** The *global memory* is a memory that can be accessed by all work items in all work groups. It may be a cached memory.

Definition 5 (Constant Memory)**.** The *constant memory* is initialized by the host and remains constant throughout the execution of an OpenCL kernel.

Definition 6 (Local Memory)**.** The *local memory* is a memory that can be shared among all work items within a work group. It can not be accessed by work items from other work groups.

Definition 7 (Private Memory)**.** The *private memory* can only accessed by a single work item.

Notation 3 (Allocated Global GPU Memory)**.** An allocated two-dimensional global memory is denoted as $G(x, y)$, with $G : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{R}$, $x \in \{0, \dots, w_\mathrm{m} - 1\}$, $y \in \{0, \dots, h_\mathrm{m} - 1\}$, width $w_\mathrm{m} \in \mathbb{N}$, and height $h_\mathrm{m} \in \mathbb{N}$. In this work, the global memory size is for most cases chosen equal to the size of the global index space, with $w_\mathrm{m} = w_\mathrm{g}$ and $h_\mathrm{m} = h_\mathrm{g}$.

Clamp to edge is a widely used technique [Khr12] to cover edge cases.

Definition 8 (Clamping)**.** A clamp function clamp : $\mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ to clamp a value x to the range $[a, b]$ is defined as

$$\operatorname{clamp}(x, a, b) := \min\big(\max(x, a), b\big), \tag{4.13}$$

with $a \leq b$.

Definition 9 (Clamped Image)**.** The clamped image $I_t^{\text{clamp}}(x, y) : \mathbb{Z} \times \mathbb{Z} \to \mathbb{R}$, where every coordinate is clamped to be within the image dimensions, is defined as

$$I_t^{\text{clamp}}(x, y) := I_t\big(\operatorname{clamp}(x, 0, w - 1), \operatorname{clamp}(y, 0, h - 1)\big). \tag{4.14}$$

Although memory can be considered as two- or three-dimensional, it is stored in a one-dimensional array in the GPU memory. Two- or three-dimensional coordinates have to be mapped to the one-dimensional array. For example, the two-dimensional coordinates (x, y) can be mapped to a one-dimensional coordinate with help of the global ID function.

Definition 10 (Global ID)**.** The global ID function gid : $\mathbb{N}_0 \times \mathbb{N}_0 \times \mathbb{N} \to \mathbb{N}_0$ maps a two-dimensional coordinate to a one-dimensional array and is defined as

$$\operatorname{gid}(x, y, w_{\text{g}}) := y \cdot w_{\text{g}} + x, \tag{4.15}$$

with w_{g} as the width of the two-dimensional memory.

For a memory of size $w_{\text{g}} \times h_{\text{g}}$, the indices x and y are only valid if they are in the interval $\{0, \ldots, w - 1\}$ and $\{0, \ldots, h - 1\}$. A valid ID can be calculated by combining the clamp function in Definition 8 with the calculation of the global ID in Equation (4.15):

Definition 11 (Global ID Clamped to Edge (gidcte))**.** The global ID with invalid indices clamped to the nearest edge is given by the function gidcte : $\mathbb{Z} \times \mathbb{Z} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N} \to \mathbb{N}_0$ with

$$\operatorname{gidcte}(x, y, w_{\text{g}}, w, h) := \operatorname{clamp}(y, 0, h - 1) \cdot w_{\text{g}} + \operatorname{clamp}(x, 0, w - 1). \tag{4.16}$$

For faster processing the local memory can be used. Threads in one work group can share the local memory. Often, the size of the allocated local memory is chosen equal or larger than the number of threads in the work group.

Notation 4 (Allocated Local GPU Memory)**.** An allocated two-dimensional local memory is denoted as $L(x, y)$, with $L : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{R}$, $x \in \{0, \ldots, w_{\text{pl}} - 1\}$, $y \in \{0, \ldots, h_{\text{pl}} - 1\}$, padded width $w_{\text{pl}} \in \mathbb{N}$, and padded height $h_{\text{pl}} \in \mathbb{N}$.

The padding region of the local memory is usually used to store some additional input data in the local memory for fast access. For example, for image processing tasks often some neighboring pixels are needed for the calculation of the output. If these pixels are needed by more than one thread, it can be beneficial to store them in the local memory.

4.1.3 Efficient Access to Global Memory

Although OpenCL has been designed as a hardware independent GPGPU programming language, the best performance can be achieved if the underlying hardware is considered during the design phase. In this section an overview is given of how efficient access to the global GPU memory can be achieved.

In general, accesses to global memory have to be prevented if possible. This includes the host to device transfer, device to host transfer, and read/write access to the global memory throughout the execution of an OpenCL kernel.

For example, if for an image processing pipeline GPGPU should be used, it would be inefficient to switch between GPU and CPU processing of the images several times throughout the pipeline, because this requires the transfer of memory if the input of one GPU kernel depends on the output of a previous CPU computation step or vice versa. In the best case, the entire processing is done exclusively on the GPU even if this requires some calculation overhead. In the best case, the only host to GPU transfer is to upload new images to the GPU and, if required, to download the processed images afterward.

Further, within a kernel only the required global memory should be loaded and it should be reused if possible. Also, it should be shared with help of the local memory if it is needed by several work items in a work group. On NVIDIA®GPUs local memory has a latency that is about 100 times [NVI09b] lower than for the global memory.

For an efficient access, the load and store of values in the global GPU memory should be done coalesced. If memory accesses are done properly, coalescing combines several memory accesses to one transaction. In more detail, with a coalesced memory access, up to half warp many threads in one work group can load up to a half warp many words in one transaction. A half warp is usually 16 or 32 threads depending on the GPU hardware architecture. A word can be up to 128 bit of data, e.g., one 32 bit float value. For 32 bit values, segments of 128 B are considered, with the first memory address of the segment properly aligned by the segment size. All threads that have scheduled a transaction within this segment can be processed with a single memory transaction. A transaction can either be 128 B, 64 B, or 32 B. The transaction size is automatically adapted to operate as many threads within the 128 B segment as possible but not larger than needed.

For example, gray-scale images are usually stored in the GPU memory as one-dimensional arrays, one row of the image after the other and one pixel after the other. The easiest and most efficient way to access these values with coalesced accesses is with successive threads that access one value one after the other. The first row of work items in the work group accesses work group width many pixels of a row of the image, the next row of work items accesses work group width many pixels the next row of the image, and so on. All work group widths should be chosen as a multiple of a half warp.

To further minimize the access to global memory, the local memory can be used. As the same values from the global memory are sometimes used by several work items, the local memory can be used to share these values among the work items. How the local memory can be used efficiently is shown in the following section.

4.1.4 Efficient Use of Local Memory

The use of local memory is a key factor for fast and energy efficient GPU code. The local memory is a small amount of memory, e.g., 16 kB, available on each of the SMPs and shared amount all threads within one work group. It can be seen as a scratchpad memory and has to be manually managed by the application designer.

Global memory accesses can have a latency of several hundred clock cycles [NVI09b]. Although, this latency can be hidden by other operations on the same SMP, global memory accesses should be minimized. The same is true with regard to energy consumption. To minimize global memory accesses, values from global memory or intermediate calculation results can be stored in the local memory and used by different threads from the same work group. Also, a non-coalesced memory access can be avoided with help of the local memory: Instead of accessing the global memory directly with a non-coalesced access pattern, global memory can be loaded with coalesced memory access into the local memory. Then values in the local memory can be accessed more or less arbitrarily.

However, for the local memory also a few hardware details have to be taken into account in order to realize the full potential. On NVIDIA®GPUs threads on an SMP access the local memory by so called memory banks. For example, if each bank is 32 bit wide, a 32 bit word can be loaded with one bank. Words from different banks can be loaded in parallel. For NVIDIA®GPUs it takes two cycles for a transfer to complete. As an example, if successive words are in different banks, number of banks many successive words can be loaded in just two cycles. In contrast, if two words are assigned to the same bank, these can only be loaded sequentially, which is called bank conflict and should be avoided. The worst case would be if several words from the same bank should be loaded at the same time. To sum up, if a GPU has, e.g., 16 banks and each bank is 32 bit wide, up to 16 times 32 bit can be loaded or stored in parallel. However, only if the application designer ensured that no bank conflicts occur and if the available bank width is fully utilized.

In the following, access pattern are investigated of how threads of a $w_\mathrm{l} \times h_\mathrm{l}$ sized work group can fill a local memory of size $w_\mathrm{pl} \times h_\mathrm{pl}$. The considered local memory is larger than the work group in both directions because the case where width and height of the local memory are both equal or smaller than width and height of the work group is more or less trivial as every thread only need load at most one element.

In the image processing domain often a neighborhood around each pixel is considered for the calculation of the output pixel. As an example, a convolution with a $u \times v$ filter kernel in the image domain requires $u \cdot v$ input pixels for the calculation of every output pixel. Typical sizes for $u \times v$ range from 3×3 to 9×9.

The $u \times v$ neighborhood is centered around each pixel. Neighboring threads in a work group process neighboring pixels in the image. If for every output pixel all input pixels are loaded from global memory, this would be very inefficient. Some of the $u \times v$ pixels can be shared within a work group as they are also needed by other work items. For example, two threads one pixel apart in u-direction share $(u-1) \cdot v$ input pixels. Here, the use of shared local memory is beneficial. If the $u \times v$ neighborhood should be available in local memory for every thread in the work group, the local memory needs to be $u-1$ pixels larger in width and $v-1$ pixels larger in height. For u and v as odd integer numbers, this would result in a local memory $\frac{u}{2}-1$ pixels larger in both x-directions and $\frac{v}{2}-1$ larger in both y-directions.

To load global memory into the local memory, efficient access patterns are needed for the work items within a work group. The access pattern needs to maximize coalesced memory access and needs to minimize warp divergences and bank conflicts. These objectives are potentially contrary to each other.

For the following experiments, a neighborhood, or filter size, of 5×5 and a work group a size of 8×8 is considered. For each pixel of the input image, the neighborhood around these pixels needs to be loaded and processed. This results in a local memory size of 12×12. All three sizes are common for practical problems.

Three access patterns are chosen of how a work group of 8×8 threads can be used to fill a 12×12 local memory. The threads are used to efficiently load values from the global memory into the local memory. Figure 4.2 shows these access patterns. With the three 8×8 arrays, a representation of the threads are given. Each thread is marked with one to four symbols, indicating how many memory operations a thread performs. With the three 12×12 arrays, the local memory is shown, each entry is loaded by one thread as indicated by the symbols. Areas of threads with one symbol are loading areas of local memory marked with the same symbol. Threads with the same symbol can be executed in the same warp.

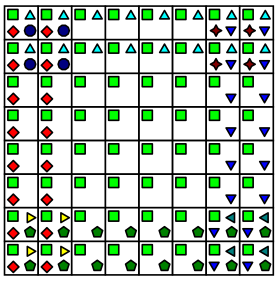
For the experiments the half warp size and the bank size are 16. The local memory has been created as a single one-dimensional array, one row after the other, beginning with the upper left element. Successive elements in the one-dimensional local memory are assigned to different banks. Every 16 entries the same bank is reached again. To avoid bank conflicts each 16 thread accesses should be properly aligned with the banks. However, as the local memory width is 12 and not 16, bank conflicts can not be avoided easily.

Coalesced global memory access is possible for successive entries in each row of the local memory that are executed within the same half warp. In addition, the accesses need to be aligned to the segments. Here, a segment size of 64 B is considered.

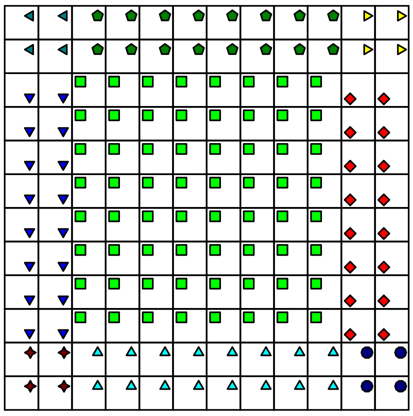
Figure 4.2a and Figure 4.2b show the first access pattern, which properly aligns global memory accesses to the segments. It is the most simple access pattern, as no index shifting is required. The work group sized center part of the local memory is loaded by all threads in parallel. Then the borders are loaded one after the other and finally the corners are loaded. The threads are properly aligned to the segments of the global memory. For the local memory, 10 bank conflicts appear. However, this pattern is not very efficient because it has eight branches where the threads diverge. In the four branches that fill the corners only four threads are active.

Figure 4.2c and Figure 4.2d show the second access pattern. The thread block of 8×8 threads is shifted up in y-direction and left in x-direction. Only three more branches are needed to load all values of the local memory. It has the least amount of branch divergences and the most threads cooperating. Also, the number of bank conflicts is low. Although more threads cooperate and only 9 bank conflicts appear, it is less efficient, due to a misalignment of the threads to the global memory segments. As the 8×8 center block of the local memory is aligned to match the segments of the global memory, the shift of the threads in x-direction misaligns the threads. It is possible, to increase the number of coalesced memory accesses by aligning the local memory consistent with the segments. However, this alignment depends on the neighborhood size and needs to be adapted if another neighborhood size should be used.

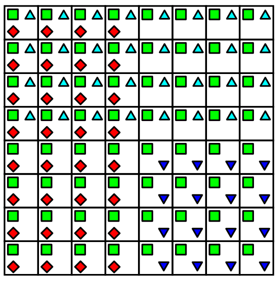
Figure 4.2e and Figure 4.2f show the third and best access pattern as used by NVIDIA®in their examples [NVI09b]. This pattern has fewer branch divergences than the first one, only five instead of eight. Like in the first pattern, also 10 bank conflicts appear.



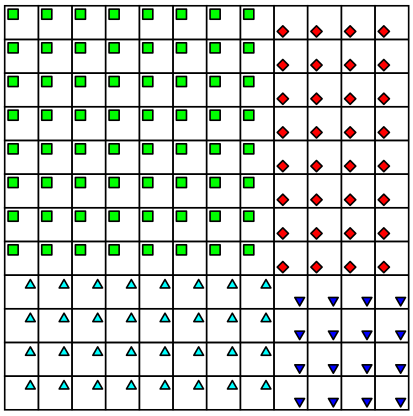
(a) Pattern 1 for a work group of 8×8 threads.



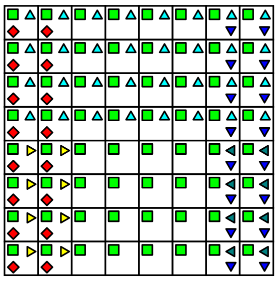
(b) Corresponding pattern 1 for a local memory of 12 × 12 entries.



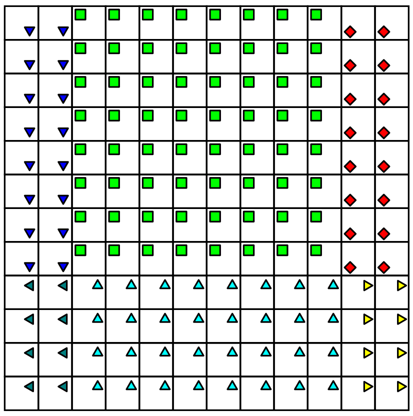
(c) Pattern 2 for a work group of 8×8 threads.



(d) Corresponding pattern 2 for a local memory of 12 × 12 entries.



(e) Pattern 3 for a work group of 8×8 threads.



(f) Corresponding pattern 3 for a local memory of 12 × 12 entries.

Figure 4.2: Three different access patterns of how a work group of 8×8 threads can efficiently initialize a local memory of size 12×12. Each thread loads one to four global memory elements into the local memory as indicated by the symbols. The third access pattern is based on [NVI09b].

The key feature is, that first the 8×8 threads load a block in the local memory that is shifted up in y-direction but not shifted in x-direction in contrast to the second pattern. With the shift in y-direction, the number of branch divergences is reduced. By waiving a shift in x-direction, the alignment to the segments is not changed.

It should be noted that all three access patterns are limited to a neighborhood, or filter size, not larger than the size of the work group plus one in x- and y-direction. If a larger neighborhood should be used, e.g., of size 11×11 for a work group of size 8×8, additional branches have to be added to fill the 13×13 sized local memory, resulting in a less efficient use of the local memory.

To conclude, if the local memory should be used efficiently, global memory accesses should be aligned to the GPU segments, local memory accesses should be aligned to the SMP banks, and the number of branches should be minimized. The third access pattern has shown to be the best, although it does not minimize the branch divergences and has some bank conflicts. It reduces the branch divergences compared to the first pattern and increases the coalesced memory accesses compared to the second pattern. Both of these two optimizations are important. The number of bank conflicts has shown to be less important, each bank conflict adds a delay of only two cycles.

A specific example of how the access pattern to the local memory can be used is provided in Section 5.4.3. The amount of saved global memory operations by using the third access pattern is calculated for a typical input buffer and a 7×7 neighborhood.

4.1.5 Additional Constraints

Based on the considerations for an efficient memory use in Section 4.1.3 and Section 4.1.4 additional soft and hard constraints can be formulated for size of the global index space, size of allocated memory, and size of the work groups. Also, the limitations of the hardware causes some additional hard constraints. These soft and hard constraints are introduced in the following, based on the definitions and notations as introduced in Section 4.1.2.

As explained in Section 4.1.3 a proper alignment of the memory to the work items or the other way round is crucial for a coalesced memory access. The proper alignment depends on the access pattern, the data type, the offset of the work items to the accessed memory blocks, the work group size, the size of the global index space, and the hardware architecture of the OpenCL device. Besides the access pattern and the underlying hardware, the width of the work groups is usually the most important parameter with which a coalesced memory access can be influenced.

Randomly accessing the pixels of an image in global memory is very inefficient. A simple but efficient access pattern can be achieved, for example, if the image width is divisible without remainder by the half warp size. Then the half warp size can be chosen as the work group width and the global index space is chosen as the image dimensions. The first global index is mapped to the upper left pixel in the image and the last global index to the bottom right pixel. Every work item accesses the mapped pixel of the image. As a result, work items in each line of a work group access a properly aligned partial line of pixels of the image in the global memory. Hence, every half warp of work items can perform a coalesced memory access.

The first two hard constraints that should be introduced are that the global width w_g and height h_g of the index space must be divisible without remainder by the work group width and height w_l and h_l:

$$w_\mathrm{g} \mod w_\mathrm{l} = 0 \tag{4.17}$$

$$h_\mathrm{g} \mod h_\mathrm{l} = 0. \tag{4.18}$$

These conditions hold trivial for $w_\mathrm{l} = 1, w_\mathrm{l} = 1$ and for $w_\mathrm{g} = w_\mathrm{l}, h_\mathrm{g} = h_\mathrm{l}$ but these solutions are in practice considered as inefficient and slow or even not possible due to limitations of the hardware. As an example, if the w_g and height h_g are prime numbers and too large to be chosen as work group width and height a work group of size one is the only possible solution. Each SMP would only execute a single thread at once, most SPs would remain unused and the threads can neither use coalesced memory accesses nor make efficiently use of shared local memory.

Usually w_l and h_l have to satisfy more conditions to result in an efficient OpenCL program. On NVIDIA®GPUs each work group is further divided into half warps as they are executed on the GPU. The threads within one half warp can work concurrently, as explained in Section 4.1.3. A half warp is 16 or 32 in current NVIDIA®GPUs. Therefore, the work group size $w_\mathrm{l} \cdot h_\mathrm{l}$ should be divisible without remainder by the half warp size s. This leads to the following soft constraint:

$$(w_\mathrm{l} \cdot h_\mathrm{l}) \mod s = 0. \tag{4.19}$$

In the best case, w_l is also divisible without remainder by the half warp size because then the memory accesses can be more easily be coalesced and bank conflicts are avoided, formulated in the next soft constraint:

$$w_\mathrm{l} \mod s = 0. \tag{4.20}$$

Another soft constraint is that the work group size should be large enough to utilize all SPs on an SMP

$$w_\mathrm{l} \cdot h_\mathrm{l} \geq \frac{\#SPs}{\#SMPs}, \tag{4.21}$$

with $\#SPs$ as the overall number of SPs and $\#SMPs$ as the overall number of SMPs.

As all work items in a work group need to be processed, the work group size should also be an integer multiply of the number of SPs per SMP, which prevents that some SPs on an SMP have no work item to process:

$$(w_\mathrm{l} \cdot h_\mathrm{l}) \mod \frac{\#SPs}{\#SMPs} = 0. \tag{4.22}$$

Also, it should be possible to distribute the work groups to the SMPs without SMPs not being utilized, which can be achieved with

$$\left(\frac{w_\mathrm{g}}{w_\mathrm{l}} \cdot \frac{h_\mathrm{g}}{h_\mathrm{l}}\right) \mod \#SMPs = 0, \tag{4.23}$$

with $\#SMPs$ as the number of SMPs.

Besides these soft constraints for efficiency, additional hard constraints are caused by the capabilities of the OpenCL hardware device. The capabilities can be read out with `clGetDeviceInfo()`. The most important constraints for the work group size are given in the values `CL_DEVICE_MAX_WORK_GROUP_SIZE`, `CL_DEVICE_MAX_WORK_ITEM_SIZES`, and `CL_DEVICE_LOCAL_MEM_SIZE`. With `CL_DEVICE_MAX_WORK_GROUP_SIZE` the overall work group size is given, with `CL_DEVICE_MAX_WORK_ITEM_SIZES[0]` the maximum work group width, and with `CL_DEVICE_MAX_WORK_ITEM_SIZES[1]` the maximum work group height. These values limit the work group size as following:

$$w_\mathrm{l} \cdot h_\mathrm{l} \leq \texttt{CL\_DEVICE\_MAX\_WORK\_GROUP\_SIZE} \tag{4.24}$$
$$w_\mathrm{l} \leq \texttt{CL\_DEVICE\_MAX\_WORK\_ITEM\_SIZES[0]} \tag{4.25}$$
$$h_\mathrm{l} \leq \texttt{CL\_DEVICE\_MAX\_WORK\_ITEM\_SIZES[1]}. \tag{4.26}$$

The overall available memory per work group in byte is given with `CL_DEVICE_LOCAL_-MEM_SIZE`. If every work item consumes m bytes local memory, the following equation has to be fulfilled:

$$m \cdot (w_\mathrm{l} \cdot h_\mathrm{l}) \leq \texttt{CL\_DEVICE\_LOCAL\_MEM\_SIZE}. \tag{4.27}$$

The minimum value for `CL_DEVICE_LOCAL_MEM_SIZE` that is guaranteed to be supported by every OpenCL device is 16 kB [Khr12]. The constant and private memory is also limited on an OpenCL device. These values can also influence the maximal possible work group size. However, they are usually not the limiting factor.

All these soft and hard constraints should be or must be considered by an application designer in order to achieve an efficient OpenCL program. However, it might not be clear which soft constraints are especially important and which can be ignored.

As shown in Equation (4.17) and Equation (4.18), the size of the index space strongly restricts the size of the work groups. To make the choice of the work group size less dependent of the index space size, some padding can be added to the width and height of the index space so that the padded width and height is divisible without remainder. The same can be done for the input or output data that should be processed. For example, if one work item solely processes one element in the memory, in a one to one mapping of work items to memory, the memory can also be padded with the same padding as the global index space.

As more limitations are given for the work group size than to the global index size it is more easy to set w_l and h_l to desired values that fulfill the dependencies and then adapt w_g and h_g to the chosen values as the other way round. Instead of choosing w_l according to w_g, w_l is chosen as the desired value and w_g is adapted to the smallest integer multiply of w_l that is greater or equal to w and analogously for h_g. This is expressed as

$$w_\mathrm{g} = \min\left\{ w + w' \;\middle|\; w' \in \mathbb{N}_0 \text{ and } \left((w + w') \mod w_\mathrm{l} \right) = 0 \right\} \tag{4.28}$$
$$h_\mathrm{g} = \min\left\{ h + h' \;\middle|\; h' \in \mathbb{N}_0 \text{ and } \left((h + h') \mod h_\mathrm{l} \right) = 0 \right\}, \tag{4.29}$$

for given $w_\mathrm{l}, h_\mathrm{l}, w, h \in \mathbb{N}$. In other words, some amount of padding is added to the width and height so that the global index space can easily be tiled into work groups.

For given w_l and h_l, the padded size of the index space can be automatically determined as follows, adapted from [Ash10]:

$$w_\mathrm{g} = (w + w_\mathrm{l} - 1) - \left((w + w_\mathrm{l} - 1) \mod w_\mathrm{l} \right) \text{ and} \tag{4.30}$$
$$h_\mathrm{g} = (h + h_\mathrm{l} - 1) - \left((h + h_\mathrm{l} - 1) \mod h_\mathrm{l} \right). \tag{4.31}$$

And the needed padding $w' \in \{0, \dots, w_\mathrm{l} - 1\}$ and $h' \in \{0, \dots, 0, h_\mathrm{l} - 1\}$ in Equation (4.28) and Equation (4.29) can be calculated with Equation (4.30) and Equation (4.31) as

$$w' = (w_\mathrm{l} - 1) - \big((w + w_\mathrm{l} - 1) \mod w_\mathrm{l}\big) \text{ and} \tag{4.32}$$
$$h' = (h_\mathrm{l} - 1) - \big((h + h_\mathrm{l} - 1) \mod h_\mathrm{l}\big). \tag{4.33}$$

Sometimes, different work group sizes are chosen for different OpenCL kernels working on the same input data. As an example, most kernels might use a work group size of 16×16 but for some kernels this size may be too big because if they use more local memory the GPU would run out of local memory. Choosing different padded width and height of the global index space or the memory would cause some additional overhead. As a solution, if the work group sizes are chosen properly, a single padded width and height can be set, which is divisible without remainder by any of the work groups, according to the following lemma:

Lemma 2. If all work group widths are set as a power of two and w_l is set as the maximum of all work group widths then w_g can be set according to Equation (4.30) and as a result w_g is divisible without remainder by any of the work group widths. The same applies to h_g in Equation (4.31) with h_l set as the maximum of all work group heights.

Lemma 2 is true, as w_g is divisible without remainder by w_l. w_l is the maximum of all work group widths. Because all work group width are chosen as a power of two, w_l is divisible without remainder by any of the work group widths. Finally, because w_g is divisible without remainder by w_l and w_l is divisible without remainder by any of the work group width, w_g is also divisible without remainder by any of the work group widths. This analogously applies to h_g.

However, padding introduces some overhead. The relative overhead o can be expressed as

$$o = \frac{w_\mathrm{g} \cdot h_\mathrm{g} - w \cdot h}{w \cdot h}, \tag{4.34}$$

with w_g and h_g as the padded width and height and w and h as the width and height without padding.

If the global index space is padded, not needed work items are added. Depending on the implementation they participate on the processing or not. In either way they usually do not produce useful output. If the memory is padded, more memory is consumed. Additionally, if the padded part of the memory is used for calculation, these parts have to be loaded or stored which takes time and consumes energy.

For an actual implementation some decisions can be made. Either only the global index space can be padded or global index space and memory. Padding only the memory is not useful for most applications. Also, additional code can be added to save calculations or memory operations, which can save execution time and/or energy. In the easiest case both global index space and memory are padded, the padding is treated as normal input or output and no additional code is added. If only the global index space is padded, additional code has to be added to prevent invalid memory accesses. A simple check if the work item is within the padded area and a `return` statement is usually enough. However, this simple check requires that the actual width and height is known. As OpenCL does not provide these parameters, two additional parameters have to be provided. These parameters have to be checked also by work items that are not assigned to the padded area.

Another problem is that OpenCL barriers require that every work item in a work group reaches the barrier before a function is allowed to continue. If some work items finish before the barrier is reached, e.g., by a conditional `return` statement for some threads, the whole function would lock up. In all cases where synchronization is needed, e.g., if some local memory is filled and shared by several work items, work items can finish only after the barriers, not before. Then, additional code has to be added to prevent that work items assigned to the padded area access invalid memory. This can reduce the readability and maintainability of the code and slows down the execution of all work items. As a solution, in this work, if barriers are used, the global index space and the memory is padded and no additional code is used for filling the local memory. Simply, all work items participate in filling the local memory even if they load not needed chunks of the global memory into the local memory. However, if neighborhood information is used, at some point after the barriers additional checks need to be added to prevent the processing of invalid memory.

In general, all the decisions made for the work group sizes, the global index space, the memory allocation, and how the data is processed influence how fast an application runs and how much energy it consumes. A generally applicable statement of how these decision should be taken can not be given as they also strongly depend, e.g., on which hardware the application runs. This motivates the use of automatic PSE and DSE methods that can help to make these decisions.

4.2 Image Processing Notations and Definitions

For the image processing methods in Chapter 5, some notations and definitions are introduced in the following.

The PAMONO sensor produces a stream of images. The raw sensor images usually have a bit depth of either 12 bit, 14 bit, or 16 bit depending on the used camera. Some cameras also output 16 bit images although they record with depths of 12 bit or 14 bit. The width w and height h of the image, cf. Notation 1, depend on the used camera in the sensor setup and are fixed during an experiment.

Individual images can be uniquely identified by the time t, also referred as the frame number. A single image instance is then be given by Notation 5.

Notation 5 (Image)**.** A gray-scale image at time t with width $w \in \mathbb{N}$ and height $h \in \mathbb{N}$ is denoted as $I_t(x, y)$. A single pixel of this image at position (x, y) is denoted as $I_t(x, y)$, with $I_t : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{R}$, $x \in \{0, \dots, w-1\}$, $y \in \{0, \dots, h-1\}$, and $t \in \mathbb{N}_0$.

For example, $I_0(0, 0)$ is the upper left pixel value of the first image and $I_{t-1}(w-1, h-1)$ is the bottom right pixel value of the t-th image.

To remove a dependency of the bit depth of the camera, all values are mapped to the interval $[0, 1]$ within the image processing software. The interval $[0, 1]$ is chosen because the floating point numbers in the interval $[0, 1]$ are represented on a GPU with a high precision and can be processed efficiently.

Additionally, the set of images I shall also be implicitly given to not always provide images as argument.

Notation 6 (Set of Images)**.** An unsorted set of all gray-scale images $\{I_0, I_1, I_2, \dots\}$ is denoted as I. All images in I have the same size $w \times h$.

Typically, I contains all images of a single measurement with the PAMONO sensor.

Definition 12 (Image Queues)**.** An ordered queue of $t_2 - t_1 + 1$ gray-scale images is defined as $I_{[t_1,t_2]} : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{R}^{(t_2-t_1+1)}$ with

$$I_{[t_1,t_2]}(x,y) := \left(I_{t_1}(x,y), I_{t_1+1}(x,y), \dots, I_{t_2}(x,y)\right)^\top \tag{4.35}$$

within the time interval $\{t_1, \dots, t_2\}$ and $x \in \{0, \dots, w-1\}$, $y \in \{0, \dots, h-1\}$, $t_1, t_2 \in \mathbb{N}$, and $t_1 \leq t_2$.

To describe and apply a PAMONO signal model, in the following the terms background signal, virus signal, artifact signal, and noise signal are defined.

First, the constant background signal B_{constant} of the PAMONO sensor is introduced as:

Definition 13 (Constant Background Signal)**.** The constant background signal is caused by the structure of the gold layer in the PAMONO sensor. A constant background signal with width w and height h is denoted as $B_{\text{constant}}(x,y)$, with $B_{\text{constant}} : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{R}$, $x \in \{0, \dots, w-1\}$, and $y \in \{0, \dots, h-1\}$.

The virus signal holds the signal caused by virus adhesions to the PAMONO sensor. For reasons of simplicity, the name and notation of the virus signal is also used for other, non-virus, particles attaching to the sensor that should be detected.

Definition 14 (Virus Signal)**.** The signal solely caused by viruses that are currently attached to the PAMONO sensor is called virus signal. The virus signal at time t with width w and height h is denoted as $V_t(x,y)$, with $V_t : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{R}$, $x \in \{0, \dots, w-1\}$, $y \in \{0, \dots, h-1\}$, and $t \in \mathbb{N}_0$.

The artifact signal holds all kind of unwanted signal, except for background and noise signal.

Definition 15 (Artifact Signal)**.** The artifact signal contains all kind of distortions in the sensor signal except background and noise signal. This includes signals from air bubbles, dust, cells, unspecific virus bindings, detaching viruses, and signal caused by movements and unwanted vibrations of the sensor system. Some part of the virus signal might also be considered as artifact signal. For example, some outer parts of the virus signal or old virus signal might be considered as unwanted signal and therefore as artifact signal. An artifact signal at time t with width w and height h is denoted as $A_t(x,y)$, with $A_t : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{R}$, $x \in \{0, \dots, w-1\}$, $y \in \{0, \dots, h-1\}$, and $t \in \mathbb{N}_0$.

The constant background signal combined with the artifact signal is called background signal.

Definition 16 (Background Signal)**.** The background signal at time t with width w and height h is denoted as $B_t(x,y)$ and consists of the constant background signal B_{constant}, cf. Definition 13, multiplied [SLW+14] with the artifact signal A_t, cf. Definition 15. It is defined as $B_t : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{R}$ with [SLW+14]

$$B_t(x,y) := B_{\text{constant}}(x,y) \cdot A_t(x,y) \tag{4.36}$$

and with $x \in \{0,\dots,w-1\}$, $y \in \{0,\dots,h-1\}$, and $t \in \mathbb{N}_0$.

The noise signal is defined as follows.

Definition 17 (Noise Signal)**.** The noise signal consists of the noise caused by the PAMONO sensor and the camera. A noise signal at time t with width w and height h is denoted as $N_t(x,y)$, with $N_t : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{R}$, $x \in \{0,\dots,w-1\}$, $y \in \{0,\dots,h-1\}$, and $t \in \mathbb{N}_0$.

To reference the neighborhood around a given position in a stream of images, the 3D neighborhood multiset N is introduced.

Definition 18 (3D Neighborhood)**.** The 3D neighborhood $N^{x,y,t}_{w_\text{n} \times h_\text{n} \times h_\text{t}}$ is a multiset of spatiotemporal coordinates of a given volume of size $w_\text{n} \times h_\text{n} \times t_\text{n}$ around a given spatiotemporal coordinate (x,y,t). It is defined as [LST+13a]

$$\begin{aligned} N^{x,y,t}_{w_\text{n} \times h_\text{n} \times t_\text{n}} := \Big\{ & \big(\text{clamp}(x+\hat{x},0,w-1),\ \text{clamp}(y+\hat{y},0,h-1),\ \max(t+\hat{t},0)\big) \in {\mathbb{N}_0}^3 \\ & \Big|\ \hat{x} \in \big\{\lceil(w_\text{n}-1)/2\rceil,\dots,\lfloor(w_\text{n}-1)/2\rfloor\big\}, \\ & \hat{y} \in \big\{\lceil(h_\text{n}-1)/2\rceil,\dots,\lfloor(h_\text{n}-1)/2\rfloor\big\}, \text{ and} \\ & \hat{t} \in \big\{\lceil(t_\text{n}-1)/2\rceil,\dots,\lfloor(t_\text{n}-1)/2\rfloor\big\}\Big\}, \end{aligned} \tag{4.37}$$

with the outermost $\{\dots\}$ denoting a multiset, $x,y,t \in \mathbb{N}_0$, and $w_\text{n}, h_\text{n}, t_\text{n} \in \mathbb{N}$. 3D neighborhood multiset N contains all spatiotemporal coordinates within a volume of width w_n, height h_n, and depth t_n around the input spatiotemporal coordinate (x,y,t). The coordinates in the multiset are clamped to edge with the clamp function, cf. Definition 8, to prevent invalid coordinates in the multiset.

Definition 19 (Average)**.** The *average* of a distribution is denoted as μ, and the average approximation $\hat{\mu}$ over $n \in \mathbb{N}$ values $x_1,\dots,x_n \in \mathbb{R}$ is calculated as [DK11]

$$\hat{\mu} := \frac{\sum_{i=1}^{n} x_i}{n}. \tag{4.38}$$

Definition 20 (Standard Deviation)**.** The *standard deviation* of a distribution is denoted as σ, and the standard deviation approximation $\hat{\sigma}$ over $n \in \mathbb{N}$ values $x_1,\dots,x_n \in \mathbb{R}$ is calculated as [DK11]

$$\hat{\sigma} := \sqrt{\frac{\sum_{i=1}^{n}(x_i-\hat{\mu})^2}{n-1}}, \tag{4.39}$$

with $\hat{\mu}$ as the average approximation from Equation (4.38). The x_i is a single realization and $n \in \mathbb{N}$ as the number of realizations.

Definition 21 (Variance)**.** The *variance* of a distribution is given by the squared standard deviation σ^2, and the variance approximation is given by the squared standard deviation approximation $\hat{\sigma}^2$ [DK11].

Definition 22 (Gaussian Distribution)**.** The *Gaussian distribution* $g : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is defined as [DK11]

$$g(x, \sigma) := \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-(x-\mu)^2}{2\sigma^2}}. \tag{4.40}$$

Definition 23 (Two-dimensional Gaussian Distribution)**.** The two-dimensional Gaussian distribution $g_\sigma : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ with zero mean is defined as [DK11]

$$g_\sigma(x, y) := \frac{1}{2\pi\sigma^2} e^{\frac{-(x-\mu_x)^2-(y-\mu_y)^2}{2\sigma^2}}. \tag{4.41}$$

Definition 24 (Poisson Distribution)**.** The Poisson distribution $p : \mathbb{N}_0 \to \mathbb{R}$ is defined as [DK11]

$$p(x) = \frac{\lambda^x}{x!} e^{-\lambda}, \tag{4.42}$$

with the distribution parameter $\lambda \in \mathbb{R}^+$, $x!$ as the factorial of x, and e as Euler's number. For a given sample, $\hat{\lambda}$ is used instead of λ and $\hat{\lambda}$ is calculated as [DK11]

$$\hat{\lambda} = \frac{\sum_{i=1}^{n} x_i}{n}, \tag{4.43}$$

with x_i as a single realization and $n \in \mathbb{N}$ as the number of realizations.

In order to determine the k-th largest element of a multiset, the rank function r is introduced.

Definition 25 (Rank)**.** The k-th largest element of a multiset $J = \{j_i \mid i \in \{1, \ldots, n\}, n \in \mathbb{N}\}$, with $k \in \{1, \ldots, |J|\}$ is given by the rank function $r(k, J)$ and is defined as [LST+13a]

$$r(k, J) := j_k \text{ where } j_1 \geq j_2 \geq \ldots \geq j_k \geq \ldots \geq j_{|J|}. \tag{4.44}$$

Based on this definition, the median can be defined:

Definition 26 (Median)**.** The *median* of a multiset $J = \{j_i \mid i \in \{1, \ldots, n\}, n \in \mathbb{N}\}$ is denoted as $\mathrm{median}(J)$ and is calculated as

$$\mathrm{median}(J) := r\left(\left\lfloor \frac{|J|+1}{2} \right\rfloor, J\right), \tag{4.45}$$

with r as the rank function according to Equation (4.44) and $|J|$ as the number of elements in the set J.

A key concept of GPGPU is a partitioning of data so that it can be efficiently processed in parallel. An image can be partitioned into tiles. Then, the tiles can be processed by the SMPs on the GPU. The tiles might overlap.

Definition 27 (Image Tile)**.** An image tile $T_{I_t(x,y)}(x', y')$ of size $w_{\text{tile}} \times w_{\text{tile}}$ from a gray-scale image I_t is defined as $T_{I_t(x,y)} : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{R}$ with

$$T_{I_t(x,y)}(x', y') := I_t(\mathrm{clamp}(x + x', 0, w - 1), \mathrm{clamp}(y + y', 0, h - 1)), \tag{4.46}$$

with the clamp function as defined in Definition 8, $x' \in \{0, \ldots, w_{\text{tile}}-1\}$, $y' \in \{0, \ldots, w_{\text{tile}}-1\}$, $w_{\text{tile}}, w_{\text{tile}} \in \mathbb{N}$, and $t \in \mathbb{N}_0$. In the following, the width w_{tile} and height w_{tile} of image tiles shall be implicitly given.

The definition for an image tile is also used to define a template image—a cutout of a virus signal. For the template image, the virus signal V_t is used in Equation (4.46) instead of I_t.

To obtain a partitioning of an image instance into not overlapping image tiles, the following set over tiles can be used.

Definition 28 (Set of Image Tiles or Partitioning of an Image)**.** A set of not overlapping image tiles is defined as

$$\begin{aligned}\big\{T_{I_t(x,y)}(x',y') \mid\ & x \in \{0\cdot w_{\text{tile}}, 1\cdot w_{\text{tile}}, 2\cdot w_{\text{tile}}, \dots, w_{\text{g}} - w_{\text{tile}} - 1\} \text{ and} \\ & y \in \{0\cdot w_{\text{tile}}, 1\cdot w_{\text{tile}}, 2\cdot w_{\text{tile}}, \dots, h_{\text{g}} - w_{\text{tile}} - 1\}\big\},\end{aligned} \tag{4.47}$$

with w_{g} as the padded width of the memory and h_{g} as the padded height.

Finally, the signal and noise can be set in relation with the SNR.

Definition 29 (Signal to Noise Ratio)**.** The estimated Signal-to-Noise Ratio (SNR) for an object O and a background B is defined as [CWG01]

$$\text{SNR}(O,B) := \frac{|\hat{\mu}_O - \hat{\mu}_B|}{\hat{\sigma}_O}, \tag{4.48}$$

with O as a multiset of object values, B as a multiset of background values, $\hat{\mu}_O$ as average of O, cf. Equation (4.38), $\hat{\mu}_B$ as average of B, and $\hat{\sigma}_O$ as the standard deviation of O, cf. Equation (4.39).

To calculate the noise approximation $\hat{\sigma}_O$ only over the object O, results in a more accurate estimate [CWG01] of the SNR. The SNR is useful as quality measure of data. Data with a high SNR is more easy to process. An SNR below four is considered as difficult.

4.3 Quantifying the Detection Quality

To measure the detection quality of a bio-medical sensor, such as the PAMONO sensor, several quality measures can be used. Which quality measures should be used, strongly depends on the use case. For the PAMONO sensor different use cases are possible. The sensor can be used for a simple (virus) test, which just measures if an analyte molecule is present in the sample or not. It can be used to determine the exact concentration of analyte molecules in the sample, which is the main use case. Also, binding speed, binding rate over time or the ratio of specific to unspecific bindings can be measured. Real-time analysis of the binding process is another use case.

For the PAMONO use case, individual viruses can be indirectly detected. The detection can be provided automatically by a software or manually by an expert. In each case the provided detections consists of positions and frame numbers where and when in the sensor image viruses have been attached to the sensor. These detections can be checked against a ground truth. The numbers of True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN) can be counted as defined in the following.

Definition 30 (True Positives)**.** True Positives (TP) in the context of PAMONO is defined as the number of particles found by the detector for which a corresponding particle in the ground truth exists.

Definition 31 (True Negatives)**.** For a true negative no particle is found by the detector and also no corresponding particle exist in the ground truth. As in the PAMONO context the true negatives cannot be counted, True Negatives (TN) is supposed to be zero.

Definition 32 (False Positives)**.** False Positives (FP) in the context of PAMONO is defined as the number of particles detected by the detector with no corresponding particle in the ground truth.

Definition 33 (False Negatives)**.** False Negatives (FN) in the context of PAMONO is defined as the number of particles in the ground truth for which no corresponding particle was detected by the detector in the sensor images.

An overview if a single event should be counted as TP, TN, FP, or FN can be found in the confusion matrix [DG06] in Table 4.1.

Table 4.1: Confusion Matrix

	Particle detected	No particle detected
Particle present	TP	FN
No particle present	FP	TN

More details on how TP, TN, and FP are calculated for the PAMONO sensor use case is given in Section 7.3.1.

Based on TP, TN, FP, and FN, several performance measures can be defined. In the following, definitions for accuracy, precision, recall, F_1 score, and Matthews correlation coefficient are given.

Definition 34 (Accuracy)**.** *Accuracy* is defined as [DG06]

$$\text{accuracy} := \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{FP} + \mathrm{FN} + \mathrm{TN}}. \tag{4.49}$$

Definition 35 (Precision)**.** *Precision* is defined as [DG06]

$$\text{precision} := \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}. \tag{4.50}$$

Definition 36 (Recall)**.** *Recall* is defined as [DG06]

$$\text{recall} := \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}. \tag{4.51}$$

Definition 37 (F_1 score)**.** The F_1 score, also named F-Score and F-Measure, is defined as a balance between precision and recall [CS93]:

$$\mathrm{F}_1 := \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}. \tag{4.52}$$

Definition 38 (Positive Agreement)**.** *Positive agreement* PA is defined as [KP03]

$$\mathrm{PA} := \frac{2 \cdot \mathrm{TP}}{2 \cdot \mathrm{TP} + \mathrm{FP} + \mathrm{FN}}. \tag{4.53}$$

Lemma 3. F_1 score and positive agreement PA are equal.

Proof.

$$\begin{aligned}
\mathrm{F}_1 &= \frac{2 \cdot \mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} && (4.54)\\
&= \frac{2 \cdot \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}} \cdot \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}}{\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}} + \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}} && (4.55)\\
&= \frac{2\mathrm{TP}^2}{(\mathrm{TP}+\mathrm{FP})(\mathrm{TP}+\mathrm{FN})(\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}} + \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}})} && (4.56)\\
&= \frac{2 \cdot \mathrm{TP}}{2 \cdot \mathrm{TP} + \mathrm{FP} + \mathrm{FN}} = \mathrm{PA}. && (4.57)
\end{aligned}$$

□

Definition 39 (Matthews correlation coefficient)**.** The *Matthews correlation coefficient* mcc is defined as [Mat75]

$$\mathrm{mcc} := \frac{\mathrm{TP} \cdot \mathrm{TN} - \mathrm{FP} \cdot \mathrm{FN}}{\sqrt{(\mathrm{TP}+\mathrm{FP}) \cdot (\mathrm{TP}+\mathrm{FN}) \cdot (\mathrm{TN}+\mathrm{FP}) \cdot (\mathrm{TN}+\mathrm{FN})}}. \tag{4.58}$$

4.4 Evolutionary Algorithms

In this section an overview of Evolutionary Algorithms (EAs) is given based on [Luk13]. EAs play an important role in Chapter 7 and Chapter 8 where they are used for single- and multi-objective optimization. They are metaheuristic algorithms that are inspired by evolution that can be used for black box optimization.

To put EAs into context, they are classified as optimization method in the most general class and stochastic optimization in the next sub-class. For stochastic optimization some degree of randomness is used to optimize. A subclass of stochastic optimization are metaheuristics. Metaheuristics describes the field of optimization methods than can find good solutions although it is unknown how an optimal solution might look like but where some kind of quality measure can be assigned to each solution that is found. The next class within metaheuristics toward EAs is Evolutionary Computation (EC), population-based optimization methods that make use of concepts from the evolution. EC covers all kind of evolution-based methods. Finally, each algorithm from the class EC is called EA. [Luk13]

In general, EAs are population-based optimization algorithms. Several candidate solution called individuals form the population. For each individual in a population, a fitness is calculated. Based on the fitness values, a new generation is formed by breeding and joining. Typically, the breeding consist of selection, mutation, and recombination of individuals. This is repeated for several generations. As an example, two common types of EAs are Genetic Algorithms (GAs) and evolution strategies.

In GAs gene like structures are used to code the parameters of individuals. A recombination is done by combining genes of two parents. For evolution strategies the breeding process only includes selection and mutation but no recombination. The input of GAs is a description of the genes, a fitness function, and a breeding function. The output is the best solution, found by the GA. [Luk13]

4.4.1 Motivation to Use EAs

In this work, EAs are chosen for optimization as a good starting point to gain insights into the inspected optimization problems and research issues. They are used to optimize GPU hardware and software and hardware configurations for GPGPU tasks.

EAs have some benefits that may not be present in other optimization methods: they are flexible, can handle a large input space, can be parallelized and solved with distributed computing, can be used for single- and multi-objective optimization, do not require gradient information, and can be easily adapted and extended. The main reasons EAs were chosen is that EAs are able to search even highly non-linear search spaces and do not make assumption on the characteristics of the search spaces.

It should be especially emphasized that this thesis is not a comparative study for different optimization techniques. The focus is on the fitness evaluation and not on the optimization. Also, the optimization concepts and methods that are introduced in this work are not limited to the use of EAs. They can be replaced by other methods for multi-objective optimization.

The main use case that is examined with EAs throughout this thesis is VirusDetectionCL, cf. Chapter 5 and Chapter 7. This program has several parameters that can affect the detection quality. With manual parameterization an optimal configuration is hard to find, especially as some parameters have strong dependencies to each other. For example, noise reduction strongly affects the segmentation thresholds, these affect the segmentation, the segmentation affects the classification, and this affects the QoR. Additionally, there are hidden side effects, which even an expert might not be aware of. For example, an improved detection of the borders of the virus adhesions can also increase the number of false positives, as more artifacts are considered as a possible virus adhesion.

In Chapter 8 more than one objective is considered for the exploration of CPSs with VirusDetectionCL and different target GPU architectures. In addition to the QoR, energy consumption and execution time are considered. This increases the difficulty of finding good solutions in the parameter and design space. The optimization task becomes even more difficult. Additional non-linear relationships have to be handled by the optimization. EAs are able to cope with such challenges.

For example, Timm has shown that EAs work well on optimizing GPU code toward the objectives execution time and energy consumption [Tim12]. This motivates the use of EAs for the hardware exploration tasks in this work. Overall, the decision to use EAs is somehow conservative but well founded.

4.4.2 EA Definitions

To define the terms in the context of EAs, the well formulated definitions from Luke [Luk13] are used in this work.

Definition 40 (Individual)**.** An *individual* is a candidate solution. [Luk13]

Definition 41 (Child and Parent)**.** If a new, modified individual is created from other individuals, the new individual is called *child* and the other individuals are called *parents*. [Luk13]

Definition 42 (Population)**.** All individuals form a *population*. [Luk13]

Definition 43 (Fitness and Fitness Evaluation)**.** The *fitness* of an individual is used to describe the quality of this individual. It is calculated by the *fitness evaluation*. [Luk13]

Definition 44 (Genotype and Genome)**.** The data structure of an individual is called *genotype* or *genome*, which is used during breeding. [Luk13]

Definition 45 (Chromosomes and Genes)**.** *Chromosomes* describe the genotype. A chromosome consists of one or more *genes*. [Luk13]

Definition 46 (Allele)**.** A particular expression of genes is called *allele*. [Luk13]

Definition 47 (Evolutionary Cycle)**.** A typical evolutionary cycle to generate new individuals from the current population consists of the steps: *evaluation*, *selection*, *recombination*, and *mutation*. Applying one of these cycles leads to a new *generation* of the population. The selection is done by picking a number of individuals based on their fitness. Recombination or crossover combines the genome of two parents to produce new children. In the mutation step the genes are altered. [Luk13]

Definition 48 (Tournament Selection)**.** Tournament selection selects the best individual out of a number of $t \in \mathbb{N}$ randomly picked individuals from the population. The t is typically chosen as 2. This step is repeated for the number of needed individuals. Tournament selection is the most used selection operation for EAs. [Luk13]

Definition 49 (Dominated Individual)**.** An individual i_1 is dominated by the individual i_2 if i_2 is in at least one objective better than i_1 and in all other objectives at least as good as i_1. [Luk13]

Definition 50 (Non-dominated Individual)**.** An individual is not dominated if no other individual exists that dominates it. [Luk13]

Definition 51 (Pareto Front)**.** A *Pareto front* consists of all non-dominated individuals. [Luk13]

4.5 The SynOpSis Approach

In this section the SYNTHESIS/OPTIMIZATION/ANALYSIS (SynOpSis) approach [SLW+14; Sie16], developed by Siedhoff, is presented. SynOpSis can be used to generate synthetic PAMONO sensor data and to optimize program parameters. The generated synthetic sensor data is important for several experiments. For the co-authored publication [SLW+14] and for the publication [Sie16], the VirusDetectionCL method, which is introduced in Chapter 5, has been provided as the most important use case. Also, the interface of VirusDetectionCL has been extended for the use with SynOpSis. In addition, several

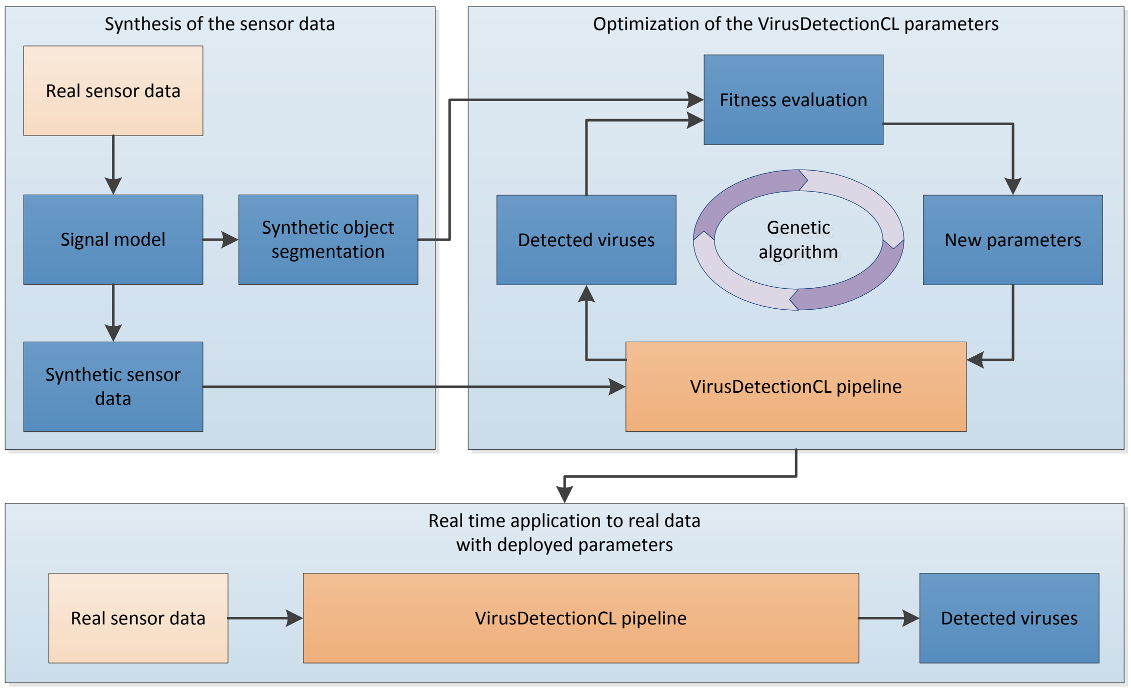

Synthesis of the sensor data
Real sensor data
Signal model
Synthetic object segmentation
Synthetic sensor data
Optimization of the VirusDetectionCL parameters
Fitness evaluation
Detected viruses
Genetic algorithm
New parameters
VirusDetectionCL pipeline
Real time application to real data with deployed parameters
Real sensor data
VirusDetectionCL pipeline
Detected viruses

Figure 4.3: The SynOpSis approach with a real data-based synthesis of training and testing data on the left, an automatic parameter optimization on the right and the real time application of the optimized parameters on real sensor data on the bottom. Adapted from [SLW+14].

improvements on both SynOpSis and VirusDetectionCL are the result of this cooperation. Therefore, SynOpSis is briefly presented in this work although it is no contribution of the author.

As the name suggests, SynOpSis contains three stages: a synthesis stage, where new synthetic data can be generated, an optimization stage, where parameters can be automatically optimized, and an analysis stage, where the optimized parameters are used to analyze real sensor data.

Figure 4.3 shows the synthesis stage on the left. The input is real sensor data with and without viruses, a manual segmentation of viruses for the sensor data with viruses, and the signal model. The sensor data without viruses can easily be provided by recording images with the PAMONO sensor while only buffer solution, e.g. salt water, is pumped through the flow cell of the sensor. The recording of the sensor data with viruses is done after a sample with viruses has been inserted in the sensor. Subsequently to the recording, a manual segmentation needs to be done by experts for a sufficient amount of viruses. SynOpSis then generates synthetic training and testing data, with a synthetic object segmentation. The signal of the segmented viruses is extracted and then synthetically injected into the empty sensor data using the signal model. The first output of the synthesis stage is a synthetic data set where the viruses are injected at random but known positions. The second output is a synthetic object segmentation with a polygon segmentation and frame number of every virus appearing in the synthetic data set. [SLW+14]

The figure also shows the optimization stage on the right, where the automatic optimization of program parameters can be done. Here, the VirusDetectionCL program, which is introduced in Chapter 5, is optimized. As input, the synthetic object segmentation and synthetic sensor data from the synthesis stage is used. The optimization stage makes use of a GA to optimize the parameters on the synthetic sensor data. As the synthetic object segmentation is also available, the fitness evaluation can easily be performed. As output, an optimized parameter set for the program is provided. [SLW+14]

Finally, the figure shows the analysis stage with the application of optimized parameters on real sensor data. In this stage new samples can be analyzed. The application of the optimized parameters is usually fast, whereas the optimization of parameters may take several hours and is done offline. [SLW+14]

In this work the synthesis stage of the SynOpSis approach is used for a Hardware-In-the-Loop (HIL) simulation. Instead of first process samples in the PAMONO sensor, then evaluate them manually, and finally analyze them with the detection software, SynOpSis can be used to synthetically generate annotated data.

CHAPTER 5

Computer Vision-Based Detection of Biological Viruses with VirusDetectionCL

Contents

5.1 Introduction **66**
5.1.1 Motivation 67
5.1.2 Architectural Design of VirusDetectionCL 67
5.2 Pre-Processing **69**
5.2.1 Input Conversion 69
5.2.2 Brightness Correction 70
5.2.3 Vibration Detection 71
5.2.4 Detection of Overexposed Spots 72
5.3 Signal Restoration **73**
5.3.1 PAMONO Signal Model 73
5.3.2 Temporal Noise Reduction and Application of the Signal Model . 74
5.3.3 Spatial Noise Reduction 77
5.3.4 Fuzzy Spatiotemporal Noise Reduction 79
5.4 Feature Extraction **81**
5.4.1 Spatial Feature Extraction 82
5.4.2 Temporal Feature Extraction 88
5.4.3 Spatiotemporal Fuzzy-Based Features 94
5.4.4 Fuzzy-Based Combination of Features 96
5.5 Segmentation **97**
5.5.1 Memory Management for Polygons 98
5.5.2 Polygon Features 98
5.5.3 Mapping Per-Pixel Features to Polygons 100
5.6 Classification **101**
5.6.1 Threshold-Based Classification 101
5.6.2 Random Forest-Based Classification 102
5.7 Post-Processing **103**
5.8 Parameters **104**
5.9 Evaluation **106**
5.9.1 Evaluation of the Execution Time 106
5.9.2 Evaluation of the Random Forest Classifier 106

5.9.3 Evaluation of Detection Quality with the SynOpSis Approach . . 107
5.10 Summary and Conclusion 109

In this chapter a detection method for biological viruses is developed. The approach is named VIRUS DETECTION WITH OPENCL (VirusDetectionCL). It is a computer vision-based approach, which can automatically detect and count individual viruses in PAMONO sensor images.

This chapter is based on the author's publications [LWT12; LST+13a; LST+13b; LKD+14] and the co-authored publications [SWL+11; SLW+14; NLE+15; STM+15].

5.1 Introduction

The VirusDetectionCL approach consists of methods for signal restoration, feature extraction, segmentation, and classification. With these methods the sensor data can be processed and signals from individual viruses can be detected and counted.

The input for VirusDetectionCL is a stream of PAMONO sensor images and a parameter set. The output is a list of the detected viruses including a segmentation and features for every virus. Features are intended to create the segmentation and for the classification of detected particles to sort out FP responses.

The PAMONO sensor produces images where the signal from viruses is not visible to the human eye. The viruses are attaching to antibodies on a thin gold layer while the signal is recorded by a camera viewing only the other side of this gold layer. Thus, with the VirusDetectionCL approach signal needs to be analyzed from the other side of the gold layer needs to be analyzed to "see" what is happening on top. That this is possible in general is because with a light source plasmon waves are excited in the gold layer, the viruses change the reflectivity of the gold layer, and this changes the amount of light recorded by the camera at an area around an attached virus that is larger than a virus itself as explained in Section 2.1. As a result, the existence of individual viruses in the sample can be proven by detecting the indirect effects they cause.

However, several challenges for processing PAMONO sensor data exist that are handled with the VirusDetectionCL method. The most important challenge is the very low SNR for all particles below 200 nm. Even a median SNR below two is possible [Sie16] and needs to be handled. Related methods for blob detection and cell tracking can only handle an SNR down to four [CSG05] or down to two [SLN+09], cf. Section 3.2.4.

Other important challenges are:

- small spatial extension of signal
- wave like structures around the center of the signal
- overlapping signals from different particles
- artifacts similar to the actual signal
- irregularities in thickness of the gold layer
- out of focus areas of the image
- overexposed spots
- only 40 ms processing time per frame
- heating of the sensor surface

- vibrations
- the energy consumption of the system

All methods from this chapter are implemented as GPGPU kernels using OpenCL. To emphasize the complexity of the VirusDetectionCL methods, it should be highlighted that the GPGPU application consists of 14 000 source lines of *C* code and 4000 source lines of OpenCL code.

The VirusDetectionCL GPGPU application is used as the main use case for the PSE and DSE approaches in Chapter 7 and Chapter 8, e.g., for a combined software and hardware exploration of VirusDetectionCL and different GPU hardware.

5.1.1 Motivation

Without automatic approaches an expert needs up to two days [Sie16] to evaluate a single PAMONO data set with 4000 images, which is recorded in 160 s. Although the recording is fast, the long evaluation time makes the use of the PAMONO sensor as a fast virus detection device infeasible. Two days evaluation time are too long for most of the use cases, e.g., for an everyday use in a hospital. This limits the use of the PAMONO sensor to cases where and the time and the costs for a two day evaluations are feasible. This is the case for a laboratory use, e.g., regarding life threatening diseases or to develop new antiviral drugs.

In contrast, with the VirusDetectionCL method the evaluation of 4000 images takes less than three minutes on appropriate hardware. It is even possible to evaluate a data set on-the-fly while it is recorded from the camera in the PAMONO sensor, which keeps the overall evaluation time also below three minutes. This opens up entirely new prospects and opportunities. By evaluating samples in parallel, an evaluation of several hundred samples is possible within a few minutes. This might be useful to test samples from all air passengers at a high risk airport, e.g., during a virus outbreak, just within the time between security check and boarding, or to prophylactically test samples on a regular basis in hospitals or doctor's offices.

Additionally, with VirusDetectionCL a fast evaluation on small hand-held devices is possible. This enables a mobile virus detection, e.g., to control disease outbreaks. If the source of an outbreak can be identified, the surrounding population can be vaccinated or a quarantine area can be established. Speed and detection quality are the most important factors to make this work. Therefore, the focus of the VirusDetectionCL method is on these two objectives.

5.1.2 Architectural Design of VirusDetectionCL

The VirusDetectionCL approach is designed as a streaming application with a pipeline structure. The VirusDetectionCL pipeline is shown in Figure 5.1 in a simplified version. Each new image, recorded by the camera in the PAMONO sensor setup, is streamed through this pipeline. At any point only a limited number of previous frames is available.

Briefly, the VirusDetectionCL pipeline structure is as follows: First, a pre-processing is applied. Then, the signal model is applied and noise is removed. On the enhanced images several features are calculated. The parts of the image that show a possible virus attachment are segmented and features are aggregated. Within the classification stage

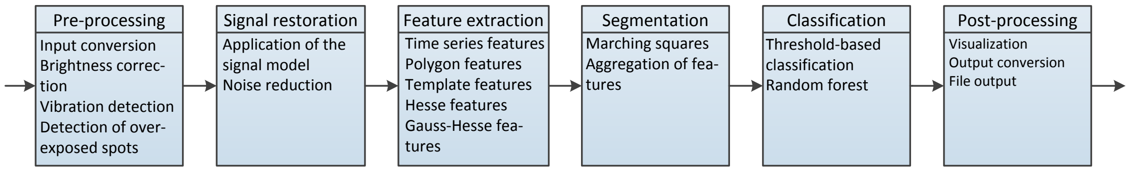


Figure 5.1: Simplified VirusDetectionCL pipeline. The sensor images are pre-processed in the first pipeline stage. In the second pipeline stage the signal model is applied and noise is removed. Afterward, features are calculated. The parts of the image that possibly show a new virus attachment are segmented and features are aggregated. Within the classification stage generated polygons are classified with the calculated features. Finally, within the post-processing stage the images are optionally processed to be visualized and saved to disk.

generated polygons are classified with the calculated features. The post-processing stage can then prepare the images that should be saved or visualized.

This VirusDetectionCL pipeline is encapsulated as a GStreamer [GSt16] plug-in, which makes it very flexible to use in different systems and on different operating systems with many kinds of input sources. GStreamer provides the input stream to the pipeline and handles the output stream. With additional GStreamer plug-ins, the recording, loading, saving, decoding, and encoding of the images is done.

In more detail, the pre-processing stage consists of conversion steps to prepare the input signal for the processing on the GPU. This includes uploading the images to the GPU, scaling of the range of the images, and optionally a brightness correction and a vibration detection.

The signal restoration stage is a very important part of the pipeline. As the SNR of PAMONO signal can be very small, the noise must be handled and the actual virus signal has to be extracted from the images to make the virus detection more robust.

The feature extraction stage is the most important part of the pipeline. In this stage good features for the virus signal need to be extracted. With the features the virus signal is distinguishable from the non-virus signal.

The segmentation stage combines per-pixel classifications to polygons that cover all pixels of a possible virus detection. The segmentation stage is not clearly separated from the feature extraction as some features require the segmentation and also per-pixel features are combined to per polygon features. It is also not clearly separated from the classification stage as a per-pixel classification is needed to create the polygons.

In the classification stage the actual decision is made which parts of the images contain virus signal and which not. In the most simple design this is done by thresholds on the feature values. In the most complex design this is done by a Random Forest.

Finally, the post-processing stage can prepare the image buffer to the desired output and display format and optionally visualizes them or saves the images on the hard drive for future use.

The overall philosophy of the design is a strictly streaming-based approach, with detection quality and processing performance as the main objectives. All pipeline stages use SIMD processing with OpenCL and the pipeline can be executed entirely on the GPU. Beside desktop and HPC systems, also mobile and ESs are the target devices. For all target devices the soft real-time processing of the PAMONO sensor data should be possible.

The frame rate of the used cameras in the PAMONO sensor varies from 5 fps to 40 fps. Consequentially, for a soft real-time processing, each frame has to be processed in 200 ms to 25 ms. VirusDetectionCL is able to achieve a so called interactive frame rate if these limits are met. As for most data sets, a camera frame rate of 25 fps was used, a processing time of 40 ms per frame is assumed as soft real-time limit for processing. This constraint influenced almost every design decision of VirusDetectionCL. Especially, if VirusDetectionCL should be executed on an embedded device, 25 ms per frame is not much time. In consequence, the number of pixels, accessed in every frame, needs to be small and also the use of complex algorithms needs to be limited to enable a processing of the images as fast as they are received from the PAMONO sensor. Hence, for VirusDetectionCL, only algorithms with a complexity of $\mathcal{O}(n)$ in Bachmann-Landau notation, with n as the number of pixels in a single PAMONO sensor image, are used.

The structure of this chapter is oriented toward the pipeline structure of VirusDetectionCL. In Section 5.2 the pre-processing is explained, followed by the signal restoration in Section 5.3, which includes noise reduction and the application of the signal model. In Section 5.4 the different features on a pixel level are explained followed by the segmentation in Section 5.5 and classification in Section 5.6. The last pipeline stage is the post-processing, which is explained in Section 5.7. Afterward, the parameter of VirusDetectionCL are listed in Section 5.8 and the chapter finishes with a short overview of the performed evaluations in Section 5.9 and a summary in Section 5.10.

5.2 Pre-Processing

The pre-processing of the PAMONO sensor data consists of the steps input conversion, brightness correction, vibration correction, and detection of overexposed spots as explained in the following.

Every received image from the camera or from image files is uploaded to the GPU memory. All succeeding steps are performed on the GPU.

5.2.1 Input Conversion

Within the first pre-processing step, the raw values from the PAMONO sensor images are converted. The raw values are provided as packed integer values and are converted to normalized float values. The conversion handles 8 bit, 16 bit, 24 bit and 32 bit depths with both big endian and little endian byte order in the packed array.

Bit depths of 16 bit are processed, by accessing two successive bytes in the packed byte array and shifting one of the bytes, depending if the bytes are packed big or little endian, by one byte to the left and combine this two byte value with the not shifted byte. The two single bytes then result in one 16 bit integer.

By dividing the resulting value by $2^{16}-1$, the input is mapped from the set $\{0, \ldots, 2^{16}-1\}$ to the interval $[0, 1]$. Many industrial cameras use a depth of 12 bit internally, stretch these values evenly to 16 bit and output 16 bit values. If however 12 bit values are stored without adaption of the range in 16 bit values, the input has to be divided by $2^{12} - 1$ to get the correct intensity values.

5.2.2 Brightness Correction

During a measurement the PAMONO sensor is illuminated by monochromatic light of a SLD or a laser diode (cf. Section 2.1). The model assumptions of the PAMONO sensor are that the intensity of the light should stay constant during a measurement and that the sensor contains no moving parts. Therefore, the resulting image should stay constant if no particles attach. However, this is not true in practice. Four main effects can influence the resulting image. First, constant illumination of the sensor causes the gold layer to heat up, which causes stronger plasmon waves, which then causes a drift in intensity toward a brighter image. If measurements are short, e.g., one minute, an increased brightness level does not affect the detection quality. For longer measurements an increased brightness level has to be compensated. Second, the camera sensor and the camera electronic also heat up and produce more noise. Third, a temporal misalignment of the sensor, e.g., through vibration, can cause that the sensor is no longer within the required surface plasmon resonance angle, resulting in an increase or decrease of the overall brightness, cf. Section 5.2.3. Fourth, it might occur that air bubbles within flow cell and tubes attach to the gold layer and cause the image to be purely white.

To make longer measurements more accurate, e.g., longer than 5 min and to avoid false positive detections at sudden brightness changes, a brightness correction was implemented. The input of this algorithm is an image and a reference brightness. The output is a corrected image with a brightness level same as the reference brightness.

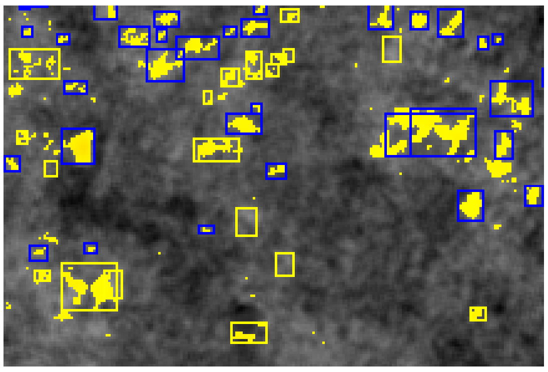
To detect and correct the drift in intensity, the brightness of the input images needs to be determined. However, the calculation of the actual brightness of an image is expensive. Instead of calculating the actual brightness, a brightness estimation is used. This reduces the number of pixel values that need to be loaded from the global memory.

The current brightness estimation $\tilde{b}_t$ for the t-th image I_t is calculated by sampling the image at work group size, cf. Definition 3, many points, summing up the values and normalizing by the number of sampled points. To speed up calculation, only a work group size as a power of two is valid. All threads within the work group load one sample point of the global image. A parallel reduction [NVI12b; NBG+08] is then used on the local memory to calculate the brightness.

It is worth highlighting that no reduction on the global memory is used, the global brightness estimation is fully calculated by each work group. This introduces overhead, as the global brightness of the image is the same for every work group and needs to be calculated again for every work group but has shown to be fast. The complexity of this approach is $\mathcal{O}(n)$ in Bachmann-Landau notation, with n as the number of pixels in the input image. Only a single kernel call is needed. If, on the other hand, a parallel reduction would be used to calculate the brightness of the image, the complexity would be $\mathcal{O}(n\log(n))$ and would require $\mathcal{O}(\log(n))$ many kernel calls, with additional overhead for every kernel call.

The local memory $L_t(x_l, y_l)$, with $x_l \in \{0,\dots,w_\mathrm{l}-1\}$ and $y_l \in \{0,\dots,h_\mathrm{l}-1\}$, is filled with values from the image I_t as

$$L_t(x_l, y_l) \coloneqq I_t\Bigg(\min\left(\left\lfloor x_l \cdot \frac{w_\mathrm{g}}{w_\mathrm{l}} + \frac{1}{2}\cdot\frac{w_\mathrm{g}}{w_\mathrm{l}}\right\rfloor, w_\mathrm{g}\right), \min\left(\left\lfloor y_l \cdot \frac{h_\mathrm{g}}{h_\mathrm{l}} + \frac{1}{2}\cdot\frac{h_\mathrm{g}}{h_\mathrm{l}}\right\rfloor, h_\mathrm{g}\right)\Bigg). \tag{5.1}$$


(a) Without brightness correction

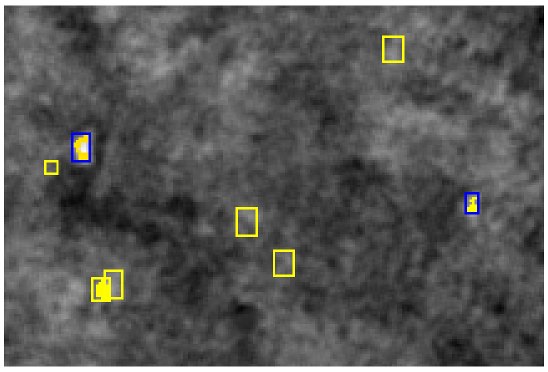

(b) With brightness correction

Figure 5.2: A brightness correction example. Without brightness correction (a), a sudden brightness increase can result in false positive detections, which are difficult to filter out. With brightness correction (b), an increase in intensity is detected and corrected, resulting in fewer false positives. The example (b) shows no false positives and no false negatives.

After synchronizing the local memory, the brightness estimation $\tilde{b}_t$ can be calculated with a local reduction as

$$\tilde{b}_t := \frac{1}{w_l h_l} \sum_{x_l=0}^{w_l-1} \sum_{y_l=0}^{h_l-1} L_t(x_l, y_l). \tag{5.2}$$

The brightness corrected image $\hat{I} : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{R}$ is then calculated by

$$\hat{I}_t(x,y) := \begin{cases} \frac{\tilde{b}_0 \cdot I_t(x,y)}{\tilde{b}_t} & \tilde{b}_t \neq 0 \\ I_t(x,y) & \text{else,} \end{cases} \tag{5.3}$$

with $\tilde{b}_0 \neq 0$.

For the concrete implementation, memory pointers of $\hat{I}$ and I are swapped, after the brightness correction, so that following filters can just use I and no memory need to be copied.

An example of how the brightness correction can result in better detection quality, can be seen in Figure 5.2. In Figure 5.2a, an image is shown, where a sudden increase in brightness occurred a few frames before. Several false positive responses are marked on the image in addition to true positive detections. With brightness correction enabled, as shown in Figure 5.2b, only true positive responses, cf. Definition 30, are detected and no false positives, cf. Definition 32, appear.

5.2.3 Vibration Detection

A problem for the PAMONO sensor is the sensitivity to vibration. As background signal is far greater than virus signal, even slight movements of parts in the system, can cause an offset in the current background signal. A translation of the background signal of a fraction of a pixel size can be enough to disturb the detection. As the background signal also contains structures that are visually very similar to the virus signal, mostly small variations in thickness of the gold layer, vibrations can cause virus like signals appearing in the images. Also, the time series show step like structures, similar to steps caused by virus adhesions.

In the laboratory the PAMONO sensor is mounted on heavy laboratory benches as shown in Figure 2.2a, which are decoupled from the floor. If, however, a mobile setup should be used, this is not so easy to achieve. The mobile setup in Figure 2.4 is put onto an anti-vibration device, which actively reduces vibrations transmitted from the floor to the PAMONO sensor. In other setups visco-elastic Sorbothane feet AV2 from THORLABS®are used, which passively dampens vibration. This reduces vibrations, transmitted from floor or table into the PAMONO sensor. However, if the sensor itself vibrates, e.g., if it is touched or if attached cables are transmitting vibrations into the system, neither the active nor the passive vibration control are sufficient to prevent all vibrations. These disturbances should to be detected.

To detect major disturbances of the sensor system, an OpenCL kernel has been implemented. This kernel simply checks if the number of detected particles and artifacts is within a reasonable range. As soon as the number exceeds a certain threshold, a number of frames before and after the detection are marked as invalid. Particles and artifacts appearing within this range are removed. Against expectations, even this simple kernel is not easy to implement on a GPU. Simple operations in a single threaded program, like calculating the minimum and maximum over an array, are much more complicated on a GPU.

To count the number of viruses and artifacts in the current frame, a parallel reduction [NVI12b] is used. The complexity of this approach is $\mathcal{O}(n\log(n))$ operations in Bachmann-Landau notation, with n as the number of pixels in the input image. As in each reduction step the remaining input is halved, $\log(n)$ reduction steps are needed. As this exceeds the desired complexity of $\mathcal{O}(n)$ for VirusDetectionCL, it should only be used if needed. Alternatively, the complexity can be reduced to $\mathcal{O}(n)$ if the detection is done in an offline step after the processing.

Additionally, an anti-vibration control and not only a vibration detection might be useful especially for hand-held devices. An anti-vibration control can be realized either in software or hardware and might be able to reduce the influence of vibration to the images. It might also be possible to prevent movements within the PAMONO sensor, as only vibrations that affect the image need to be avoided. If the physical setup of SLD, prism, gold layer, and camera is rigid, movements of the whole sensor setup do not influence the sensor image quality. Because the current laboratory setup can be used without vibrations in the system and a hand-held device might not need a vibration control if it is build more rigid, a vibration control was saved for future work.

5.2.4 Detection of Overexposed Spots

To enhance the SNR in the images the signal, intensity of the SLD or laser diode, should be as high as possible without losing information of the attaching particles. The maximum brightness of the pixels in the camera has to be considered. The signal should be near the maximum brightness but far enough away prevent the pixel values to be saturated if a virus attaches.

If a few bright spots are on the image, e.g., caused by irregularities in the gold layer, there are two possibilities to handle these: First, the overall intensity can be lowered such that the brightness of these spots is within the desired range. Second, the few bright spots can be ignored and the brightness can be set higher to be more suitable for most of the pixels.

In the first case, the overall SNR is lower but all pixel values are in the desirable range. In the second case, the overall SNR is higher but a few overexposed spots are in the image.

Usually, a few overexposed spots would not be a problem for the detection as the according pixels should have a constant signal over time. But the problem is that around overexposed spots the risk of false positive detections is high because slight vibrations of the sensor can shift the position of the bright areas into the surrounding darker areas. This can cause an increase in intensity that is almost indistinguishable from an increase in intensity from an attaching virus. Therefore, vibration of the sensor needs to be prevented or detections near overexposed spots should be filtered out.

To detect overexposed spots a segmentation of all pixels with a too high brightness level is performed. The same segmentation as in Section 5.5 is used. The output of this segmentation are polygons around the bright spots in the image. This detection step has to be done only once as the position of the bright spots does not change. The detected spots can then be used to sort out false positive detections as described in Section 5.5.

5.3 Signal Restoration

The signal restoration consists of the application of the PAMONO signal model and noise reduction. The output of this step is an approximated virus signal, where viruses should be clearly visible. The application of the PAMONO signal model is crucial for all image-based features because otherwise the virus signal would be dominated by the background signal. A good noise reduction is especially important for a detection of smaller viruses below 200 nm. With lower noise the SNR is larger and the low virus signal is more easily to detect.

5.3.1 PAMONO Signal Model

The following empirical signal model is adopted from Siedhoff et al. [SLW+14]. The image signal I_t as defined in Notation 5 at time t can be expressed as

$$I_t(x,y) = B_t(x,y) \cdot V_t(x,y) + N_t(x,y), \tag{5.4}$$

with B_t as the background signal as defined in Definition 16, V_t as the virus signal as defined in Definition 14, and N_t as the additive noise signal as defined in Definition 17. The background signal consists of a constant background signal B_{constant}, as defined in Definition 13, and an artifact signal A_t, as defined in Definition 15. More details on the relation of signal, background, and noise are given in [ZSS+16].

The virus signal that is recorded by the camera $V_t : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{R}$ at time t is composed of an actual virus signal $\hat{V}_t$ from the sensor, convolved with a kernel K_y [SLW+14], which models the influence of lenses and camera. The kernel is dependent of the y-coordinate in the image because the camera within the PAMONO sensor is not orthogonal to the sensor surface. Therefore, the virus signal is given as

$$V_t(x,y) = (\hat{V}_t * K_y)(x,y), \tag{5.5}$$

with $*$ denoting the convolution.

The noise signal N_t at time t is a combination of shot noise from the PAMONO sensor [BZH+07], fixed pattern noise and Gaussian distributed noise, cf. Definition 22, from the camera sensor [SLW+14].

It should be noted that the intensity of the SLD or laser diode in the PAMONO sensor mostly influences the virus signal and the background signal and only to a small degree the noise. Therefore, the light source intensity should be set to a level with that the CCD chip pixels nearly reach—but not exceed—their saturated capacity, e.g., to 90 % of the saturated intensity [BZH+07]. This increases the intensity of the background signal and virus signal and therefore results in a larger SNR.

5.3.2 Temporal Noise Reduction and Application of the Signal Model

With the signal model from Section 5.3.1, the virus signal can be extracted from the PAMONO sensor signal. For a better approximation of the actual virus signal, temporal noise is handled simultaneously. According to the signal model the image I consists of a background signal B, multiplied with a virus signal V and an additive noise term N, see Equation (5.4). If a single particle appears at time t, the virus signal is not present in all previous images and present in the current and all following images. To extract the virus signal from the sensor signal, the background signal before time t and the current signal is calculated. Each is calculated over one or more frames to reduce the noise in the signal.

The input of this method are two sliding windows of sensor images. One sliding window of size a for the background approximation and one sliding window of size b for the approximation of the current image. The output of this method is an approximated virus signal at time t. Because a virus might not appear instantaneous, due to the binding processes to the antibodies, and because the approximated virus signal should contain a virus for a few frames to calculate the features, additionally a gap of size g can be specified. In consequence, the g frames in $\{t - \lfloor \frac{g}{2} \rfloor, \dots, t + \lceil \frac{g}{2} \rceil - 1\}$ are left out in the calculation.

The background signal B_t at time t is approximated by calculating the median of some values in the past. As the virus has not yet attached in the frames before time t, the inspected image values in the time series contain only the background signal and possibly some old virus signal.

Thus, the background approximation $\widetilde{B}_t : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{R}$ is calculated as

$$\widetilde{B}_t(x,y) = \operatorname{median}\left(\left\{ I_{t'}(x,y) \;\middle|\; t' \in \left\{ t - \left\lfloor \frac{g}{2} \right\rfloor - a, \dots, t - \left\lfloor \frac{g}{2} \right\rfloor - 1 \right\} \right\}\right), \tag{5.6}$$

with $a \in N$ as size of the sliding background window, $g \in \mathbb{N}_0$ as size of the gap and $\{\dots\}$ denoting a multiset.

An important aspect of the median filter is that the step characteristic of attaching viruses is preserved. Additionally, the median filter reduces artifact signal A that is present in less than $\frac{a}{2}$ frames. The complexity of this approach is $\mathcal{O}(n)$, with n as the image size. The number of inspected images is constant and therefore the median calculation for each output pixel can be done in constant time.

As an alternative to the median, averaging over the values can be used to reduce noise and artifact signal. A major advantage of using the average instead of the median is that it can be implemented such that it can be calculated very efficient on the GPU: A sum over all values is stored. Then, the new average value can be calculated by subtracting values from the oldest image and adding the values from the newest image and dividing the updated sum by the number of images. For each pixel only three values have to be read from the global memory.

However, averaging does not preserve the adhesion characteristic of the viruses over time. In addition, an artifact signal in less than $\frac{a}{2}$ frames is only dampened and not removed. In Section 5.4.2 a method is presented to reflect the behavior of averaging and adapt the features to this. As a result, even if the step characteristic is lost, the steps can be detected.

The current image is approximated similar to the background signal. The current image approximation $\widetilde{C}_t : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{R}$ is given as

$$\widetilde{C}_t(x,y) = \text{median}\left(\left\{ I_{t'}(x,y) \;\middle|\; t' \in \left\{ t + \left\lceil \frac{g}{2} \right\rceil, \dots, t + \left\lceil \frac{g}{2} \right\rceil + b - 1 \right\} \right\}\right), \tag{5.7}$$

with $b \in \mathbb{N}$ as size of the sliding window, $g \in \mathbb{N}_0$ as size of the gap and $\{\dots\}$ denoting a multiset.

The signal $\widetilde{B}$ contains a background signal with a reduced additive noise term. The signal $\widetilde{C}_t$ contains a combined virus signal and background signal with a reduced additive noise term.

Consequently, the virus signal approximation $\widetilde{V} : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{R}$ is calculated as

$$\widetilde{V}_t(x,y) = \frac{\widetilde{C}_t(x,y) + \varepsilon}{\widetilde{B}_t(x,y) + \varepsilon}, \tag{5.8}$$

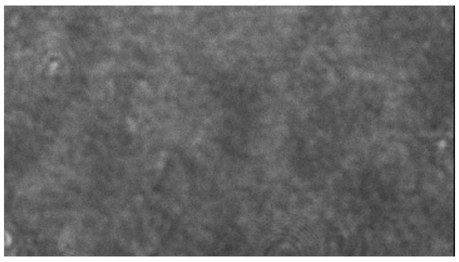
with $\widetilde{B}_t(x,y) \neq 0$ and $\varepsilon \in \mathbb{R}_{>0}$. The ε value is added to the approximated current image values and background image values to prevent a division by zero. For pixel depths of 16 bit, ε is set to $\frac{1}{2^{16}-1}$.

Figure 5.3 shows the application of the signal model on a raw sensor image, cf. Figure 5.3a, resulting in the virus signal approximation, cf. Figure 5.3b. A single virus is visible on the left side in Figure 5.3b. For one pixel position of this virus, the corresponding time series are plotted in Figure 5.3c and Figure 5.3d.

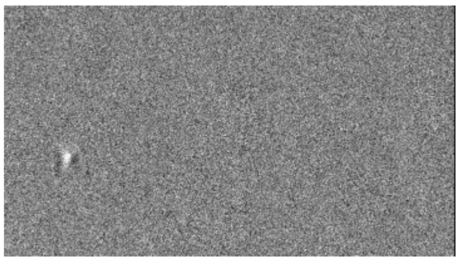
It should be noted that the division in Equation (5.8) might increase the additive noise, left in the approximated background signal. This can be compensating by further spatial noise reduction.

Under ideal conditions, a virus signal shows up in the approximated virus signal $\widetilde{V}$ for $\frac{b}{2} - 1 + g + \frac{a}{2} + 1$ frames. As soon as the virus signal is present in the median of the background signal, the application of the signal model removes it from the approximated virus signal. As a result, only the recent viruses show up in $\widetilde{V}$.

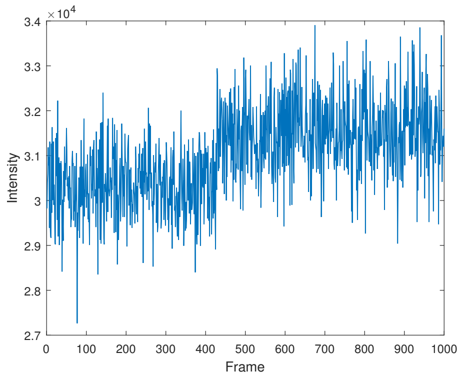
Some implementation details are given in the following because this method uses many input images. If the needed memory accesses are not performed efficiently, this step in the pipeline can strongly affect the overall performance. When calculations on time series of pixel values are done, the data should be loaded cooperatively in every work group one frame after the other. How data should be arranged in the GPU memory to achieve coalescing depends on the access patterns to single data elements. Because the memory is accessed one sub-image after the other and not one time series after the other, it is better to store the data in a ring buffer like structure one image after the other and not as single time series, one after the other. As a result, the memory accesses can easily be coalesced.



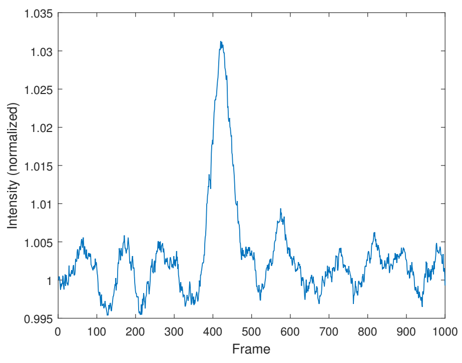
(a) Raw sensor image.



(b) Signal model applied to the raw image from (a). An attached virus is visible on the left side.



(c) Raw time series of the virus pixel position. The virus attaches at frame 428.



(d) Time series with applied signal model and spatiotemporal noise reduction of the same virus pixel position as (c).

Figure 5.3: Raw sensor image and processed sensor image with corresponding time series of a virus pixel. Contrast of the processed image has been enhanced for visualization.

For every new image the array of the ring buffer is updated by replacing the oldest image with the newest. A queue implementation would be very inefficient, as the images need to be shifted, resulting in unnecessary memory accesses. It should be noted that in OpenCL it is not possible to use pointers to pointers. Consequently, the usual solution to simply store pointers to the images in a list and update only the pointers if a new image is added is not possible with OpenCL.

For the GPGPU calculation of the median, the images to filter are loaded into a private memory array of constant size. Then, a Bubble-Select algorithm partially sorts the values over time to determine the median value. For every pixel position in the image, a thread on the GPU is launched, which shows to perform very well. Because the work group sizes are selected as a multiple of a half warp, as explained in Section 4.1.2, array values can be loaded coalesced.

For the inspected small, constant array sizes the complexity of the algorithms is of minor importance. Of more importance is a low branch divergence of the threads. Here, the Bubble-Select algorithm performed better than a Quicksort-based selection algorithm. This is due to the fact that there is no branch divergence with the used Bubble-Select algorithm, while the Quicksort and Quickselect algorithms diverge among threads in the same branch, with each different pivot element. On typical GPUs a branch divergence of threads should be avoided, on CPUs in contrast, the influence of branch divergences can be neglected.

5.3.3 Spatial Noise Reduction

With spatial noise reduction the remaining noise that is left after the application of the signal model and the temporal noise reduction should be reduced. Spatial noise reduction must be conducted on the approximated virus signal $\widetilde{V}_t$ and not on the input images, as the virus signal is very small in comparison to the background signal and can otherwise easily be corrupted.

Three well known methods for noise reduction are considered: applying an average, median, or Gauss filter to the image. These methods are no new contribution.

With the definition of a 3D neighborhood multiset N of width w_n, height h_n, and length l in Definition 18, the averaged output image I'_t is calculated from the input images I_t as

$$I'_t(x,y) := \hat{\mu}\Big(\big\{I_{t'}(x',y') \mid (x',y',t') \in N^{x,y,t}_{w_\text{n}\times h_\text{n}\times 1}\big\}\Big), \tag{5.9}$$

with $\hat{\mu}$ as defined in Definition 19 and $\{\dots\}$ denoting a multiset.

Let N denote the 3D neighborhood as before. The median filtered output image I'_t is then calculated from the input images I_t as

$$I'_t(x,y) := \text{median}\Big(\big\{I_{t'}(x',y') \mid (x',y',t') \in N^{x,y,t}_{w_\text{n}\times h_\text{n}\times 1}\big\}\Big), \tag{5.10}$$

with median as defined in Definition 26 and $\{\dots\}$ denoting a multiset.

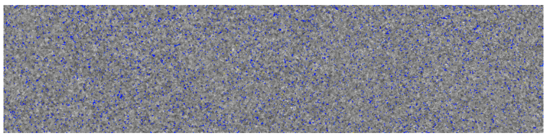
The Gauss filtered output I of the input image J with $\sigma \in \mathbb{R}+$ is calculated with Definition 23 as

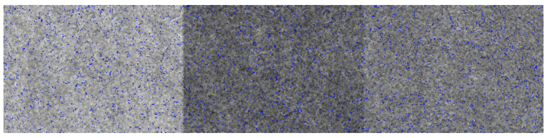
$$I_t(x,y) := (g_\sigma * J_t)(x,y). \tag{5.11}$$

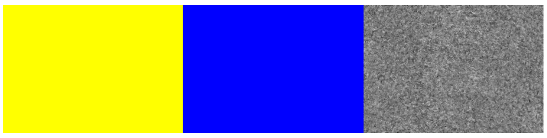
Because the size of the needed neighborhood around every pixel should be small, the size of the Gaussian kernel is limited to 2σ in each direction. The energy is kept constant by normalizing the output value in accordance to the sum of the Gaussian values.

Also, a Fast Fourier Transform (FFT)-based noise reduction has been examined and, in previous work [Lib11], a Haar wavelet-based noise reduction. Both approaches did perform well on easy data sets but not well enough on more challenging data sets. Bublitz has done a multi scale FFT-based pyramid filtering [Bub12] on the PAMONO sensor data. The approach from Bublitz, although it produced good results on all data sets, was too slow to be used within the soft real-time limits.

In Figure 5.4 an example for the spatial and temporal noise reduction is shown. To produce these results a data set of artificially created test images with perfect steps has been added to empty sensor data without any particles. The images are partitioned in three areas. The first area contains steps with increased intensity, the second with decreased intensity and the third area is the control image with no step. Figure 5.4a shows an image before the steps occurred and Figure 5.4b shows an image after the steps occurred. In Figure 5.4c the reference detection is given and in Figure 5.4d the detection from VirusDetectionCL with enabled spatial and temporal noise reduction. The spatial and temporal noise reduction was done with the median filter from Equation (5.7) and Equation (5.10).


(a) Before the up and down step occurred


(b) After the up and down step occurred


(c) Reference detection

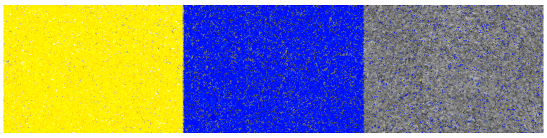
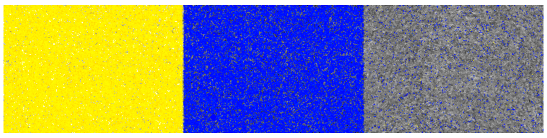
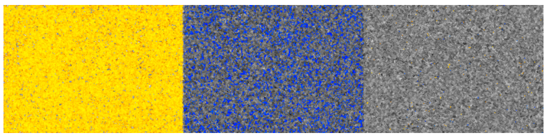
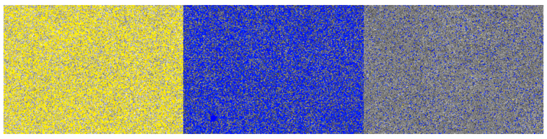

(d) With spatial and temporal noise reduction

Figure 5.4: Artificially created test images, with perfect steps added to PAMONO images without particles. In the left area up-steps, in the middle area down-steps and in the right area no steps as control. In detail, the image before the steps appear (a) and after the steps appeared (b) and a reference detection (c) in comparison with a detection (d) by the VirusDetectionCL software, where spatial and temporal noise reduction was enabled.


(a) With spatial and temporal noise reduction


(b) With spatial noise reduction


(c) With temporal noise reduction

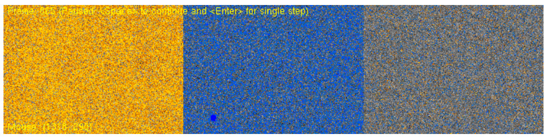

(d) Without noise reduction

Figure 5.5: Different noise reduction methods in comparison, with impact on the step detection. Detection with spatial and temporal noise reduction (a), detection with only spatial noise reduction (b), detection with only temporal noise reduction (c) and detection with no noise reduction (d). A reference detection is shown in Figure 5.4c.

Figure 5.5 shows different noise reduction methods in comparison. The input is the same as for Figure 5.4 and in Figure 5.5a the same result of the combined spatial and temporal noise reduction as in Figure 5.4d is given as base line. In Figure 5.5b only the spatial median filter from Equation (5.10) was used, in Figure 5.5c only the temporal median filter from Equation (5.7) was used, and in Figure 5.5d the same without any noise reduction for comparison. As expected, the combined spatial and temporal noise reduction provides the best overall results with 95.94 % correct classified pixels .

5.3.4 Fuzzy Spatiotemporal Noise Reduction

The most sophisticated noise reduction method in this work is a fuzzy spatial noise reduction, which uses fuzzy filters to remove noise. An overview of using fuzzy filters for noise reduction is given by Nachtegael et al. [NWV+01].

Mélange, Nachtegael, and Kerre have presented a fuzzy noise removal method, in several variations [MNK10b; MNK10a; MNK11; MNS+11; NMK11]. In this thesis the author has combined and adapted these methods, mainly from [MNK10a; MNS+11], to the specific nature of the PAMONO sensor images. The fuzzy rules have been adapted to the characteristics of the PAMONO sensor signal such that the noise is removed and the virus signal is preserved as good as possible. As another contribution of the author, the methods have been adapted for the GPU and implemented in OpenCL. Several optimizations have been done to meet the soft real-time constraint.

The input is the current virus signal approximation $\widetilde{V}_t$ and previously calculated filter results from different points in time. The output is a new virus signal approximation which should contain less noise without affecting the actual virus signal.

So called fuzzy sets are used to model the degree of membership to a certain class. Here, the classes *noise* and *noise-free* are used. The range for the membership functions for each set is $[0,1]$. The degree of membership is similar to a probability but membership values can sum up to values larger or smaller than 1.

To build the fuzzy sets, the functions $\mu^{\pi_1,\pi_2}_{\text{largePositive}}$ and $\mu^{\pi_1,\pi_2}_{\text{largeNegative}}$ with the soft thresholds $0 \le \pi_1 < \pi_2$ are used in the following. The function $\mu^{\pi_1,\pi_2}_{\text{largePositive}} : \mathbb{R} \to [0,1]$ is defined as [MNK10a]

$$\mu^{\pi_1,\pi_2}_{\text{largePositive}}(v) := \begin{cases} 1 & \text{for } v \ge \pi_2 \\ \frac{v-\pi_1}{\pi_2-\pi_1} & \text{for } \pi_1 < v < \pi_2 \\ 0 & \text{else} \end{cases}. \tag{5.12}$$

This function maps the input values to the interval $[0,1]$ with a smooth transition from zero to one for values between the soft thresholds. In the same way, the function $\mu^{\pi_1,\pi_2}_{\text{largeNegative}} : \mathbb{R} \to [0,1]$ is defined with the soft thresholds $0 \le \pi_1 < \pi_2$ [MNK10a]:

$$\mu^{\pi_1,\pi_2}_{\text{largeNegative}}(v) := \begin{cases} 1 & \text{for } v \le -\pi_2 \\ \frac{|v|-\pi_1}{\pi_2-\pi_1} & \text{for } -\pi_2 < v < -\pi_1 \\ 0 & \text{else} \end{cases}. \tag{5.13}$$

The filtering of the noise is done in four steps. The first step removes the obvious noise and the following steps refine the filtering. Each filter step uses different results from other filter steps. Within the filter steps a 3D neighborhood set $N_{w_n \times h_n \times l}(x,y,t)$ is used as defined in Definition 18. The first step $f_0^{x,y,t} : \mathbb{R} \to \mathbb{R}$ uses the PAMONO virus signal as input:

$$f_0^{x,y,t} := \widetilde{V}_t(x,y). \tag{5.14}$$

The filter steps f_1 to f_4 use various neighborhood values from previous filter steps and from different times as input. With $\{\dots\}$ denoting a multiset, their domain is given as follows, adapted from [MNS+11]:

$$f_1^{x,y,t} : A_1 \times B_1 \times C_1 \to \mathbb{R} \tag{5.15}$$

with

$$A_1 = \{ f_0^{x',y',t'} \mid (x',y',t') \in N_{6\times 6\times 1}^{x,y,t} \},$$
$$B_1 = \{ f_2^{x',y',t'} \mid (x',y',t') \in N_{5\times 5\times 1}^{x,y,t-1} \}, \text{ and}$$
$$C_1 = \{ f_4^{x',y',t'} \mid (x',y',t') \in N_{5\times 5\times 1}^{x,y,t-2} \}.$$

$$f_2^{x,y,t} : A_2 \to \mathbb{R} \tag{5.16}$$

with

$$A_2 = \{ f_1^{x',y',t'} \mid (x',y',t') \in N_{3\times 3\times 1}^{x,y,t} \}.$$

$$f_3^{x,y,t-1} : A_3 \times B_3 \times C_3 \to \mathbb{R} \tag{5.17}$$

with

$$A_3 = \{ f_2^{x',y',t'} \mid (x',y',t') \in N_{9\times 9\times 1}^{x,y,t} \},$$
$$B_3 = \{ f_2^{x',y',t'} \mid (x',y',t') \in N_{9\times 9\times 1}^{x,y,t-1} \}, \text{ and}$$
$$C_3 = \{ f_4^{x',y',t'} \mid (x',y',t') \in N_{9\times 9\times 1}^{x,y,t-2} \}.$$

$$f_4^{x,y,t-1} : A_4 \to \mathbb{R} \tag{5.18}$$

with

$$A_4 = \{ f_3^{x',y',t'} \mid (x',y',t') \in N_{5\times 5\times 1}^{x,y,t-1} \}. \tag{5.19}$$

To define the fuzzy rules, the rank function r from Definition 25 is used, which calculates the k-th largest element. Also, the fuzzy operators AND, OR, and NOT are used, defined as [MNS+11]

$$\text{AND }(x_1, x_2, \dots) := \min(x_1, x_2, \dots) \tag{5.20}$$
$$\text{OR }(x_1, x_2, \dots) := \max(x_1, x_2, \dots) \tag{5.21}$$
$$\text{NOT }(x) := 1 - x, . \tag{5.22}$$

Two membership functions are used to classify a pixel as noise free or noisy. The membership of a pixel to the fuzzy set $\mu_{\text{noise-free}}$ explains how noise free the pixel is considered, analogously for the fuzzy set μ_{noise}. To build these sets the characteristic of the PAMONO signal is translated into fuzzy rules. Exemplary, the first fuzzy rule $\mu_{\text{noise-free-1}}$ from the fuzzy set $\mu_{\text{noise-free}}$ is explained.

It is assumed that a pixel is noise free, if there are not at least four pixels in the neighborhood that have a large difference to the current pixel. Or, if there are not at least two pixels in the neighborhood with a large difference and not at least one pixel of two previous filter steps in the past that at the same position and that have a large difference:

$$\mu_{\text{noise-free-1}}^{\pi_1,\pi_2}(s) := r\left(4, \left\{ \text{ NOT } \mu_{\text{largePositive}}^{\pi_1,\pi_2}\left(|f_0^{x,y,t} - a|\right)\right\}\right) \text{ OR} \tag{5.23}$$
$$\left(r\left(2, \left\{ \text{ NOT } \mu_{\text{largePositive}}^{\pi_1,\pi_2}\left(|f_0^{x,y,t} - b|\right)\right\}\right) \text{ AND}\right.$$

$$\Bigg(\text{NOT } \mu^{\pi_1,\pi_2}_{\text{largePositive}}\Big(|f_0^{x,y,t} - f_2^{x,y,t-1}|\Big) \text{ AND}$$

$$\text{NOT } \mu^{\pi_1,\pi_2}_{\text{largePositive}}\Big(|f_0^{x,y,t} - f_4^{x,y,t-2}|\Big)\Big)\Bigg)$$

with

$$a \in \left\{f_0^{x',y',t'} \mid (x',y',t') \in N^{x,y,t}_{6\times6\times1}\right\} \text{ and}$$
$$b \in \left\{f_0^{x',y',t'} \mid (x',y',t') \in N^{x,y,t}_{5\times5\times1}\right\},$$

with $\{\dots\}$ denoting a multiset.

There are several rules to detect noise free or noisy pixels. All these sets are combined to the fuzzy sets μ_{noise} and $\mu_{\text{noise-free}}$ using the AND operator. After these sets are evaluated, the class of the pixel is assigned to the fuzzy set to which it has the highest membership. Finally, the pixel is filtered or not:

$$f_1^{x,y,t} := \begin{cases} \frac{f_4^{x,y,t-2} + f_2^{x,y,t-1}}{2} & \text{for } \mu_{\text{noise}} > \mu_{\text{noise-free}} \\ f_0^{x,y,t} & \text{else} \end{cases}. \tag{5.24}$$

If noise is present, an average over two previous filter results is calculated, which results in a signal that should be almost noise free. If no noise is present, the unmodified input is used as output.

This procedure is then repeated with other fuzzy rules for the next three steps $f_2^{x,y,t}$, $f_3^{x,y,t}$, and $f_4^{x,y,t}$. For every step less noise needs to be filtered and the remaining noise is possibly more easy to detect in the following steps. Overall, the filtering in several steps should be easier than filtering all the noise in one step [MNS+11]. As information about the PAMONO signal is used to build the fuzzy rules, this is an adaptive noise filtering method. In the best case the virus signal is completely preserved while the noise is filtered.

Besides the adaptation to the PAMONO signal, the method has also been extended to run on the GPU. Because a lot of neighborhood information is used by the fuzzy rules, an extensive use of local memory is mandatory for an efficient GPU program. To achieve this local memory with additional padding is used. The padding is chosen accordingly to the neighborhood sizes. For each input array the needed neighborhood is loaded concurrently into local memory, cf. Section 4.1.4. Early termination conditions are added that check if the current pixel achieves a membership of 100 % to the noise free class in which case the rules for the noise class do not have to be evaluated.

This fuzzy based noise reduction also has been extended to a method for spatiotemporal fuzzy-based features as is presented in Section 5.4.3.

5.4 Feature Extraction

In this section different features for the virus detection are introduced. Features are characteristics that can be used to describe the signal. Here, features are used to distinguish between virus and non-virus signals on a per-pixel level. The per-pixel features consists of spatial, temporal and spatial-temporal features and are presented in Section 5.4.1, Section 5.4.2, and Section 5.4.3.

It should be noted that additional per-polygon features are presented in Section 5.5. The per-polygon features can only be calculated after the segmentation in Section 5.5 and therefore they are not presented in this section but in Section 5.5.2. Additionally, the per-pixel features from this section are mapped to polygons resulting in per-polygon features. This mapping is also not presented in this section but in Section 5.5.3.

5.4.1 Spatial Feature Extraction

Virus adhesions have a distinct appearance in the PAMONO sensor signal. A virus adhesion shows as a blob-like structure with more or less wave-like patterns around the center, cf. Figure 2.7. With spatial features the spatial appearance of structures and patterns in the image can be analyzed. The spatial features in VirusDetectionCL include a Gauss-Hesse feature, a Gauss-Hesse blob feature, a Hesse feature, and a template matching feature. The spatial features are especially useful if the SNR in the PAMONO signal is below four.

Gauss-Hesse Feature and Gauss-Hesse Blob Feature

Moon et al. have proposed a blob detection algorithm for tumor detection [MSB+13]. These blobs are similar to the blob like virus attachments in the PAMONO sensor data.

As the blob detection algorithm presented by Moon et al. was not designed to detect viruses, not to be used as a streaming approach, and not to run on a GPU, the novel contribution for this thesis is the transposition of the method to the requirements of the PAMONO task.

The input of the adapted method is a single virus signal approximation $\widetilde{V}_t$. The output consists of one or more per-pixel features.

With the two-dimensional Gaussian distribution $g_\sigma(x, y)$, as defined in Definition 23, the following partial derivatives $g_{1,\sigma}$, $g_{2,\sigma}$, $g_{3,\sigma}$, and $g_{4,\sigma} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ [MSB+13] can be calculated.

Derivative $g_{1,\sigma}$, as depicted in Figure 5.6a, is calculated by deriving g two times in x-direction:

$$g_{1,\sigma}(x, y) := \frac{\partial^2 g_\sigma}{\partial x \partial x}(x, y) = \frac{x^2 - \sigma^2}{\sigma^4} \cdot g_\sigma(x, y). \tag{5.25}$$

Derivative $g_{2,\sigma}$, as depicted in Figure 5.6b, is calculated by deriving g two times in y-direction:

$$g_{2,\sigma}(x, y) := \frac{\partial^2 g_\sigma}{\partial y \partial y}(x, y) = \frac{y^2 - \sigma^2}{\sigma^4} \cdot g_\sigma(x, y). \tag{5.26}$$

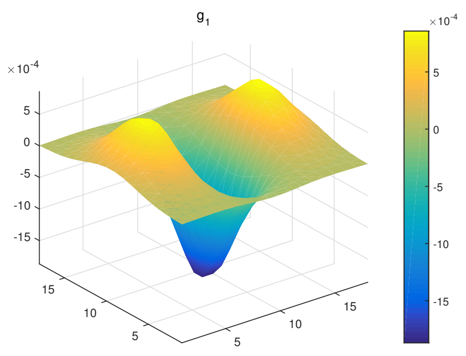
Derivative $g_{3,\sigma}$, as depicted in Figure 5.6c, is calculated by deriving g first in x- and then in y-direction:

$$g_{3,\sigma}(x, y) := \frac{\partial^2 g_\sigma}{\partial x \partial y}(x, y) = \frac{xy}{\sigma^4} \cdot g_\sigma(x, y). \tag{5.27}$$

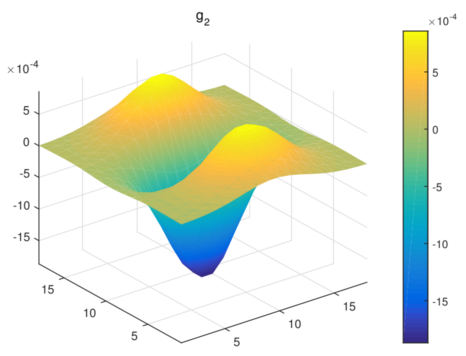
Derivative $g_{4,\sigma}$, as depicted in Figure 5.6d, is calculated by deriving g first in y- and then in x-direction:

$$g_{4,\sigma}(x, y) := \frac{\partial^2 g_\sigma}{\partial y \partial x}(x, y) = \frac{yx}{\sigma^4} \cdot g_\sigma(x, y). \tag{5.28}$$

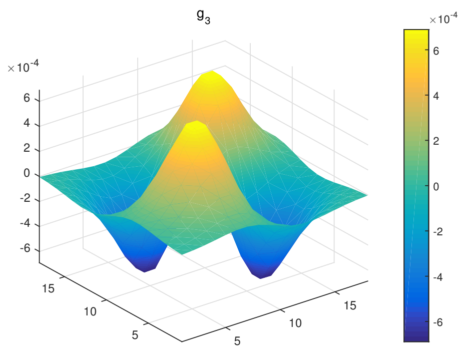
Note that $g_{4,\sigma}$ is equal to $g_{3,\sigma}$ because of the symmetry of the multiplication.



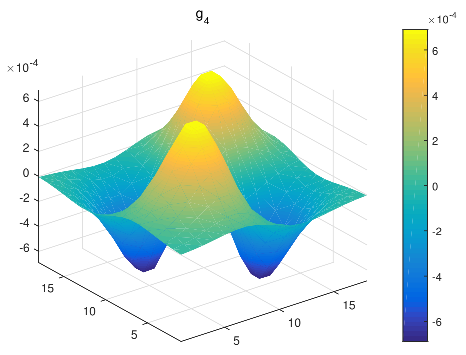
(a) $g_{1,3}(x,y) := \frac{\partial^2 g_3}{\partial x \partial x}(x,y)$



(b) $g_{2,3}(x,y) := \frac{\partial^2 g_3}{\partial y \partial y}(x,y)$



(c) $g_{3,3}(x,y) := \frac{\partial^2 g_3}{\partial x \partial y}(x,y)$



(d) $g_{4,3}(x,y) := \frac{\partial^2 g_3}{\partial y \partial x}(x,y)$

Figure 5.6: Different partial derivatives $g_{1,3}$ (a), $g_{2,3}$ (b), $g_{3,3}$ (c), and $g_{4,3}$ (d) of a 2-dimensional Gaussian distribution $g_\sigma(x,y)$ with $\sigma = 3$. The partial derivatives $g_{3,3}$ and $g_{4,3}$ are equal.

Let $\hat{g}_{i,\sigma}(\widetilde{V}_t, x, y) : \widetilde{V}_t \times \mathbb{N}_0 \times \mathbb{N}_0 \rightarrow \mathbb{R}$ denote the convolution of $g_{i,\sigma}(x', y')$ with the virus signal $\widetilde{V}_t(x,y)$ and $i \in \{1,2,3,4\}$:

$$\hat{g}_{i,\sigma}(\widetilde{V}_t, x, y) := (g_{i,\sigma} * \widetilde{V}_t)(x,y). \tag{5.29}$$

For reasons of clarity, the short notation $\hat{g}_1$ to $\hat{g}_4$ is used instead of $\hat{g}_{1,\sigma}(\widetilde{V}_t, x, y)$ to $\hat{g}_{4,\sigma}(\widetilde{V}_t, x, y)$ in the following equations.

The Hessian matrix H is then given as [MSB+13]

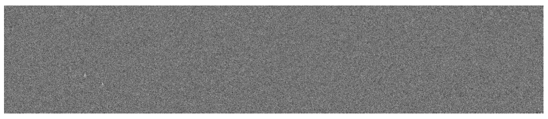
$$H = \begin{pmatrix} \hat{g}_1 & \hat{g}_3 \\ \hat{g}_4 & \hat{g}_2 \end{pmatrix}. \tag{5.30}$$

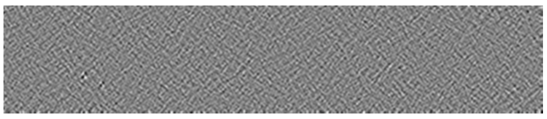
Contrary to Moon et al., here, also the determinant $\det(H)$ and the trace $\operatorname{tr}(H)$ of the Hessian matrix are used as features, similar to features in the Speeded Up Robust Features (SURF) [BTG06] approach:

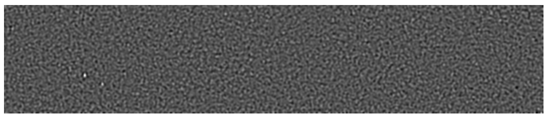
$$\det(H) = \hat{g}_1 \cdot \hat{g}_2 - \hat{g}_3 \cdot \hat{g}_4 \tag{5.31}$$

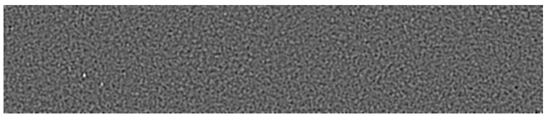
$$\operatorname{tr}(H) = \hat{g}_1 + \hat{g}_2. \tag{5.32}$$

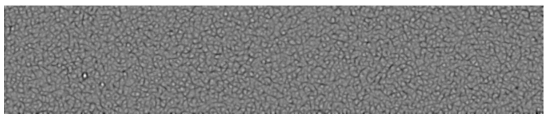
The trace of the Hessian matrix is also known as Laplacian Of Gaussian (LoG) in literature [Gun99]. These two features can easily be calculated from intermediate results of the Gauss-Hesse features.


(a) Pre-processed input image. Two virus attachments are slightly visible at lower left.


(b) Intermediate result $\hat{g}_{3,3}(\widetilde{V}_t, x, y)$. Convolution of a partial derivative with the virus signal, cf. Equation (5.29).


(c) Determinant of the Hesse matrix feature, cf. Equation (5.31).


(d) Trace of the Hesse matrix feature, cf. Equation (5.32).


(e) The Gauss-Hesse blob likeness feature b_1, cf. Equation (5.34).

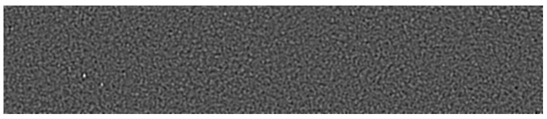

(f) The Gauss-Hesse blob likeness feature b_2, cf. Equation (5.35).

Figure 5.7: A noisy sensor image as input (a). Two virus attachments are slightly visible at lower left. One intermediate results (b) and features of the determinant of the Hessian matrix (c), the trace of the Hessian matrix (d), the Gauss-Hesse blob feature (e), and the Gauss-Hesse feature (f), applied to the sensor image. High values are visualized as white and low values are visualized as black.

Next, the eigenvalues $\lambda_{1,2}$ of the Hessian matrix are calculated with the pq-formula as

$$\lambda_{1,2} = \frac{\operatorname{tr}(H)}{2} \pm \sqrt{\left(\frac{\operatorname{tr}(H)}{2}\right)^2 - \det(H)}. \tag{5.33}$$

The Gauss-Hesse blob likeness feature b_1 is then calculated as [MSB+13]

$$b_1 = \begin{cases} \frac{|\lambda_1|}{|\lambda_2|} & \text{if } \lambda_1, \lambda_2 < 0 \text{ and } |\lambda_1| \le |\lambda_2| \\ \frac{|\lambda_2|}{|\lambda_1|} & \text{if } \lambda_1, \lambda_2 < 0 \text{ and } |\lambda_1| > |\lambda_2| \\ 0 & \text{else} \end{cases}. \tag{5.34}$$

In contrast to Moon et al., in the PAMONO sensor use case, bright spots instead of dark spots need to be detected. Therefore, this equation has been modified to detect bright spots by using $\lambda_1, \lambda_2 < 0$ in the condition and not $\lambda_1, \lambda_2 > 0$.

The Gauss-Hesse blob feature b_2 is calculated as [MSB+13]

$$b_2 = \begin{cases} \left(1 - e^{\frac{-b_1}{2\alpha^2}}\right) \cdot \left(1 - e^{\frac{-\sqrt{\lambda_1^2+\lambda_2^2}}{2\beta^2}}\right) & \text{if } \lambda_1 < 0 \text{ and } \lambda_2 < 0 \\ 0 & \text{else} \end{cases}. \tag{5.35}$$

Just the same as in Equation (5.34), this equation has also been modified to be used for the detection of bright spots instead of dark spots. With $\alpha, \beta \in [0,1]$ a weighting can be conducted. In [MSB+13] α and β are set to 0.5.

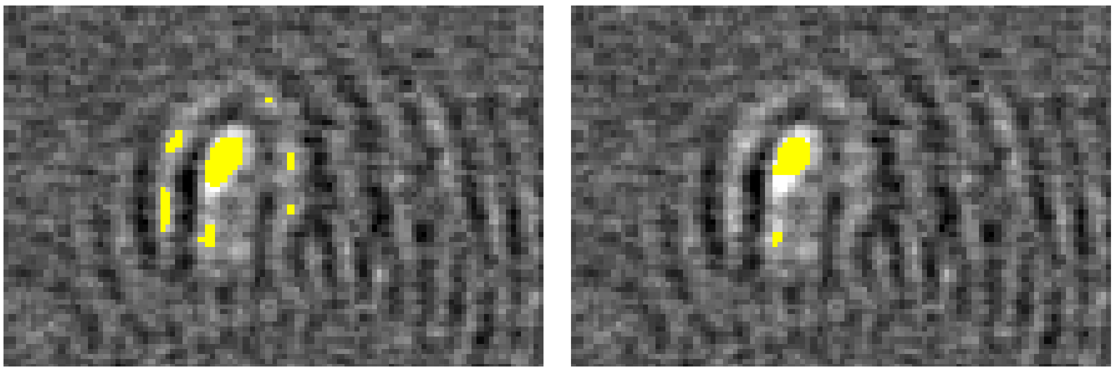

(a) Without Gauss-Hesse Blob Feature **(b)** With Gauss-Hesse Blob Feature

Figure 5.8: Gauss-Hesse blob feature used for pixel classification. Comparison of a segmentation with a time series-based feature (a) and the same time series-based feature combined with the Gauss-Hesse blob feature (b).

Figure 5.7 shows how the four Gauss-Hesse features look like if they are visualized with high feature values mapped to white and low feature values to black. The pre-processed input image is shown in Figure 5.7a. Two virus attachments are slightly visible at the lower left of the image. Intermediate results of the convolution of a partial derivative with the virus signal are shown in Figure 5.7b. The determinant of the Hessian matrix is shown in Figure 5.7c and the trace of the Hessian matrix is shown in Figure 5.7d. Finally, a visualization of the Gauss-Hesse blob likeness feature is shown in Figure 5.7e and for the Gauss-Hesse blob feature in Figure 5.7f. The two attaching viruses that are only slightly visible on the pre-processed input image are clearly visible as bright spots in the visualized feature images.

Additionally, the Gauss-Hesse features have been modified so that they can be used both for a per-pixel classification of time series and as per-polygon features for classification of polygons. To use the Gauss-Hesse features as per-polygon features, the Gauss-Hesse feature is calculated for every pixel in every image. After the polygons are generated for every polygon, the best feature value of the Gauss-Hesse features is searched in the axis-oriented bounding box around the polygon, as described in Section 5.5.3. This is an approximation for reasons of speedup. To search only within the polygon, the pixels within the polygon need to be determined, which is more expensive. However, searching in the axis-oriented bounding box is a valid approximation. Because the Gauss-Hesse feature values that correspond to pixels within the bounding box but not to the polygon usually have lower Gauss-Hesse values, this approximation will in most cases have no effect on the classification.

Figure 5.8 shows the use case of a per-pixel classification in combination with a time series-based temporal feature, cf. Section 5.4.2. With the Gauss-Hesse features, the detection of viruses is focused more on the blob-like center of the virus and less on the wave-like structures around the virus.

In addition to the single-scale approach, also a multi-scale feature approach has been developed, very similar to the approach of Moon et al. [MSB+13]. The multi-scale feature approach works as follows. Starting with a small sigma value for the first scale, e.g.,

$\sigma = 0.75$, the sigma value is doubled for each of the higher scales. The features for each scale are then calculated and stored in a three-dimensional array, which is number of scales times the size as the array for the single-scale approach. The main differences to [MSB+13] are that here the sigma values are doubled and that multi-scale features are used independently and are not reduced to a single feature. With this the Random Forest classifier, cf. Section 5.6.2, or other classifiers, can make use of all scales and not only of the best scale. Moon et al. are simply using the maximum value of all calculated scales as a feature [MSB+13]. They also calculate the maximum value over all sigma values within a range, which is time-consuming.

To conclude, the enhanced Gauss-Hesse features show a good performance. Especially if small viruses need to be detected they enhance the only slightly visible viruses very well. If only a single virus size should be detected, the single-scale approach is well suited. The sigma can be optimized for a known virus size. If more than one virus size should be detected, the multi-scale approach is well suited.

A detection of different virus sizes in one sample is part of future work on the PAMONO sensor. More than one antibody will be used on the same gold layer of the sensor. The automatic processing software can then count the number of viruses for each virus size by including the multi-scale features and applying a multi-class classification.

Hesse Feature

The Hesse feature [TRS+02] is another feature for detecting blob like structures. This feature is similar to the Gauss-Hesse feature in Section 5.4.1. Same as for the Gauss-Hesse feature partial derivatives are used. However, instead of g_σ, the virus signal $\widetilde{V}_t(x,y)$ is used as input. The output is a feature value for every pixel position.

Derivative d_1 is calculated by deriving $\widetilde{V}_t$ two times in x-direction:

$$d_1(x,y) := \frac{\partial^2 \widetilde{V}_t}{\partial x \partial x}(x,y). \tag{5.36}$$

Derivative d_2 is calculated by deriving $\widetilde{V}_t$ two times in y-direction:

$$d_2(x,y) := \frac{\partial^2 \widetilde{V}_t}{\partial y \partial y}(x,y). \tag{5.37}$$

Derivative d_3 is calculated by deriving $\widetilde{V}_t$ first in x- and then in y-direction:

$$d_3(x,y) := \frac{\partial^2 \widetilde{V}_t}{\partial x \partial y}(x,y). \tag{5.38}$$

Derivative d_4 is calculated by deriving $\widetilde{V}_t$ first in y- and then in x-direction:

$$d_3(x,y) := \frac{\partial^2 \widetilde{V}_t}{\partial y \partial x}(x,y). \tag{5.39}$$

The Hessian matrix H is then given as [TRS+02]

$$H = \begin{pmatrix} d_1 & d_3 \\ d_4 & d_2 \end{pmatrix}. \tag{5.40}$$

Analogous to the Gauss-Hesse feature the determinant $\det(H)$ and trace $\operatorname{tr}(H)$ of the Hessian H are calculated according to Equation (5.31) and Equation (5.32). Contrary to [TRS+02], the determinant and the trace are also used as feature.

Finally, the Hesse feature $h_{t,w_\mathrm{h},h_\mathrm{h}} : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{R}$ at time t with filter size $w_\mathrm{h} \times h_\mathrm{h}$ is given as [TRS+02]

$$h_{t,w_\mathrm{h},h_\mathrm{h}}(x,y) = \det(H) \cdot \hat{\mu}\Big(\big\{V_{t'}(x',y') \mid (x',y',t') \in N^{x,y,t}_{w_\mathrm{h} \times h_\mathrm{h} \times 1}\big\}\Big), \tag{5.41}$$

with $\{\dots\}$ denoting a multiset, $\hat{\mu}$ as the average approximation as defined in Definition 19, and N as the multiset of neighborhood coordinates as defined in Equation (4.37).

In contrast to the Gauss-Hesse feature, the Hesse feature shows to be less robust to noise. Additionally, it uses a rectangular average filter kernel, which is, in the PAMONO context, inferior to a Gauss filter. An advantage of the Hesse feature is that it is slightly faster than the Gauss-Hesse feature.

Template Matching Feature

The template matching feature uses previously identified viruses to detect viruses. Virus adhesions have a specific appearance in the signal if the background is removed: a bright, blob like center and wave like structures around this center. Under ideal conditions, the center of the signal and the wave like structure around it would be perfectly circular. Whereas in practice, the center is oval and the wave like structures are more pronounced to one direction because of the angle between camera and gold plate and the orientation of the camera. In Figure 5.8 a typical signal of a virus adhesion can be seen.

The characteristic of the PAMONO signal is the same for every particle of the same size. On the sensor images however, this is only true for similar vertical positions because on the sensor only horizontal lines are of the same sharpness. This is due to the angle between the sensor surface and the camera. Consequently, templates of different sharpness have to be used depending on the vertical position. Overall, very few templates are enough to match almost every virus template in the images

The input to calculate the template matching feature is the current approximated virus signal and one or more previously identified viruses given as templates. The output is a per-pixel matching score of the best match from the templates.

A template $T(x',y')$, with $T : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{R}$, $x' \in \{0,\dots,w_\mathrm{tile}-1\}$, and $y' \in \{0,\dots,w_\mathrm{tile}-1\}$, is usually given as a cutout of a virus signal of the same particle size as the one that should be detected. It can be described as

$$T(x',y') \coloneqq T_{V'_t(a,b)}(x',y'), \tag{5.42}$$

with $T_{V'_t(a,b)}$ as defined in Definition 27 and (a,b) as the upper left position of a virus in the virus signal V'.

To calculate the matching score $S : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{R}$ with template T, the slightly modified normalized cross correlation method CV_TM_CCORR_NORMED from Open Source Computer Vision (OpenCV) [Bra00] is used, given as

$$S_t(x,y) = \frac{\sum_{x'=0}^{w_{\text{tile}}-1} \sum_{y'=0}^{w_{\text{tile}}-1} T(x',y') \cdot \widetilde{V}_t^{\text{clamp}}(x - \lfloor \frac{w_{\text{tile}}}{2} \rfloor + x', y - \lfloor \frac{w_{\text{tile}}}{2} \rfloor + y')}{\sqrt{\sum_{x'=0}^{w_{\text{tile}}-1} \sum_{y'=0}^{w_{\text{tile}}-1} T(x',y')^2 \cdot \sum_{x'=0}^{w_{\text{tile}}-1} \sum_{y'=0}^{w_{\text{tile}}-1} \widetilde{V}_t^{\text{clamp}}(x - \lfloor \frac{w_{\text{tile}}}{2} \rfloor + x', y - \lfloor \frac{w_{\text{tile}}}{2} \rfloor + y')^2}}, \tag{5.43}$$

with $\widetilde{V}_t^{\text{clamp}}$ as the clamped to edge virus signal value according to Definition 9.

For the actual OpenCL implementation, the local memory is heavily used. A local memory of size $w_{\text{pl}} \times h_{\text{pl}}$ with

$$w_{\text{pl}} = w_{\text{l}} + w_{\text{tile}} - 1 \text{ and} \tag{5.44}$$

$$h_{\text{pl}} = h_{\text{l}} + w_{\text{tile}} - 1 \tag{5.45}$$

is used and loaded according to the access pattern presented in Section 4.1.4. It stores a cutout of the virus signal $\widetilde{V}_t^{\text{clamp}}$. Once the threads in one work group have cooperatively loaded the local memory and are synchronized, all calculations in the work group are done completely in private and local memory.

In contrast to the virus signal $\widetilde{V}$, the template T is not stored in global or local memory but in the private memory. As the template is constant, this is the fastest way to access the template values. However, the size of the private memory is limited and shared among all threads in one work group. If the templates for all threads in the work group do not fit in the private memory, the work group size needs to be reduced or the global and local memory has to be used instead.

5.4.2 Temporal Feature Extraction

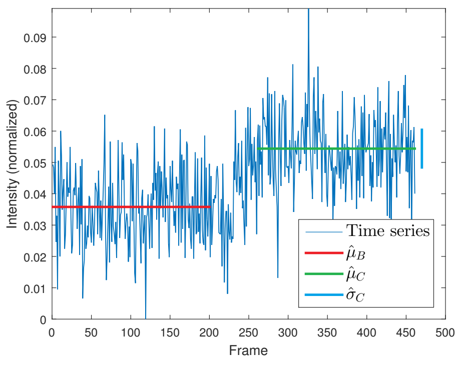
In addition to the spatial features, several temporal features are calculated on the time series.

In Figure 5.9 two time series are shown: in Figure 5.9a a virus pixel position is shown and in Figure 5.9b a non-virus pixel position for comparison. In the first time series a virus attaches approximately at frame 230, which causes an increase in intensity. The noise is quite high, which makes it complicated to calculate reliable features. The measured SNR, as defined in Definition 29, is 1.55. In the second time series no virus attaches, only noise is shown. If an SNR on the noise is calculated by assuming $\hat{\mu}_B$ as the value for background and $\hat{\mu}_C$ as the value for signal plus background the conceived SNR is 0.2.

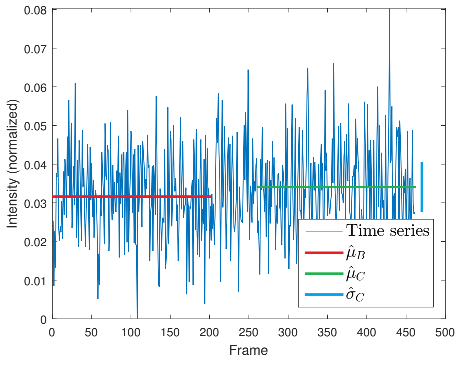
In Figure 5.10 a hard classification task is shown. A time series of a 100 nm particle (Figure 5.10a) is almost indistinguishable from a time series without any particle (Figure 5.10b).

Simple Threshold-based Counting

One of the simplest features on time series is to calculate the fraction of points in one part of the time series that are in the expected range of a virus adhesion. This feature is no



(a) Time series of a virus pixel position with an SNR of 1.55. The virus attaches approximately at frame 230.



(b) Time series of a non-virus pixel position over 460 frames.

Figure 5.9: Example time series of a virus pixel (a) and a non-virus pixel (b). The constant background signal was removed. SNR according to Equation (4.48).

new contribution. It has been developed by Zybin and is used for the manual evaluation in [STM+15]. It is only listed here for the sake of completeness.

The input to the feature calculation are time series of the virus signal $\widetilde{V}$, the output is a matching score for every pixel. To calculate the matching score, the intensity change in comparison to the average approximation $\hat{\mu}$ on a previous part of the time series is considered. The change in intensity must exceed a threshold α but not a threshold β. If this condition holds, the fraction is calculated by dividing by the number of elements n in the first part of the time series. Finally, the matching score $S : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{R}$ is given as

$$S_{t,\alpha,\beta,\hat{\mu}}(x, y) = \frac{1}{n} \sum_{i=0}^{n-1} \chi_{t-i,\alpha,\beta,\hat{\mu}}(x, y), \tag{5.46}$$

with $\chi : \mathbb{N}_0 \times \mathbb{N}_0 \to \{0, 1\}$ as an indicator function:

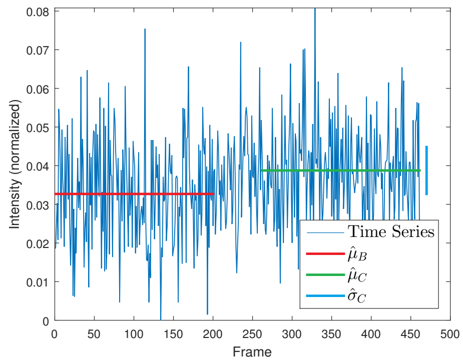
$$\chi_{t,\alpha,\beta,\hat{\mu}}(x, y) = \begin{cases} 1 & \text{if } \alpha \leq \widetilde{V}_t(x, y) - \hat{\mu} \leq \beta \\ 0 & \text{else} \end{cases}. \tag{5.47}$$

Surprisingly, this simple method works well and is also quite robust to noise in the data.

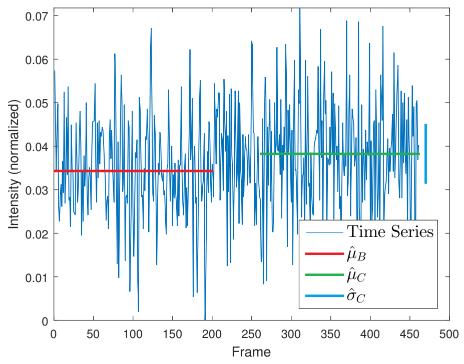
Temporal Template Matching

As the characteristic of an attaching virus in the time series is always very similar, this characteristic can be used as a template to identify the time series corresponding to a virus attachment. The time series template matching approach that is presented in this section is an extended version of previous work [Lib11].

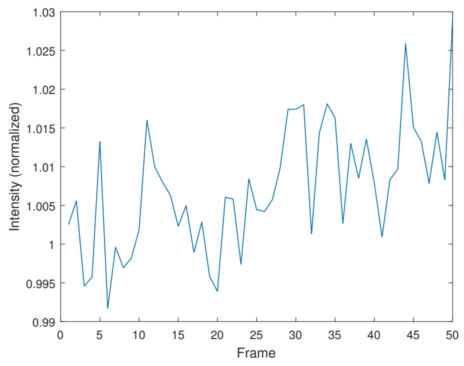
The input is a template vector and a series of virus signals. The output is a matching score for every pixel position, specifying how strongly each time series resembles to the template vector. In [Lib11] a simple Euclidean distance has been used to calculate the



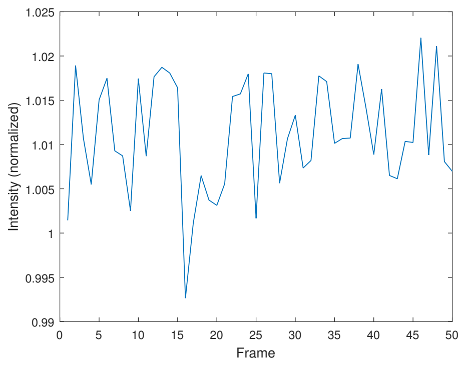
(a) Time series of a 100nm virus pixel position over 460 frames with an SNR of 0.5. The virus attaches approximately at frame 230.



(b) Time series of a non-virus pixel position over 460 frames.



(c) Time series of a 100nm virus over 1000 frames. Each 20 images are combined to one image, resulting in 50 combined frames out of the 1000 input frames. The virus attaches approximately at frame 27.



(d) Time series of a non-virus over 1000 frames. Each 20 images are combined to one image.

Figure 5.10: Example time series of a virus pixel position with a bad SNR (a) and a non-virus pixel position (b) and the same positions with combined images in (c) and (d). The constant background signal has been removed.

matching score. Here, the more sophisticated cosine similarity [DSJ06] is used as matching score. With a given template vector

$$\vec{p} := (p_1, p_2, \ldots, p_n)^\top, \forall i \in [1, n] : p_i \in \mathbb{R} \text{ and } n > 1, \tag{5.48}$$

the matching score $S : \mathbb{N}_0 \times \mathbb{N}_0 \to [-1, 1]$ for matching of a time series to a template is given as

$$S_{t,\vec{p}}(x, y) = \frac{\sum_{i=0}^{n-1} \widetilde{V}_{t-i}(x, y) \cdot p_{n-i}}{\sqrt{\sum_{i=0}^{n-1} \widetilde{V}_{t-i}(x, y)^2 \cdot \sum_{i=0}^{n-1} p_{n-i}^2}}. \tag{5.49}$$

For the implementation it has to be assured that the square root is not zero, which can happen on actual data. A matching score of zero is used as output in this case.

Three different implementations of this algorithm have been examined, with the templates stored in the private memory, constant memory, and global memory of the GPU. For the implementation that uses private memory, the template is stored directly into the code. For the implementation that uses constant memory, the template is stored in a constant, two-dimensional floating point number array, which is globally defined and accessed from within the kernel. Finally, for the implementation that uses the global memory, a read-only global memory is allocated and the template values are uploaded to this array. The use of global memory has the advantage, that templates can be updated by a program argument, which is especially useful if the template is part of an automatic optimization.

As expected, the implementation that stores the template in the private memory is the fastest. The implementation that uses the global memory is the slowest but the most flexible as templates can be changed by a parameter. On a desktop GPU this implementation is also fast, but as can be seen in Section 8.5, the execution time on the Odroid ES is not sufficient to meet the soft real-time conditions of 25 fps.

Automatic Adaption of Time Series Templates

As explained in Section 5.3.2, the temporal noise reduction can alter the step characteristic of signals from attaching viruses. Especially, averaging on intervals of the time series or a gap smaller than the template size results in altered signals. The templates need to reflect this characteristic.

Therefore, a method has been developed, which automatically updates the template regarding the used noise reduction methods and parameters. This is done by taking the step templates and use it as input for the noise reduction pipeline. As the temporal noise reduction algorithms need some frames for initialization, the first template value is used to initialize the queues. Each template value is uploaded one after the other to the input image and this image is treated as regular input to the pipeline. With this trick all noise reduction filters are applied to the template values.

As a result, the step signal in the templates is processed and altered exactly the same way as the step signal of the PAMONO sensor data. The altered templates are then used for the template matching. This method can be used for all kind of templates.

Dynamic Time Warping

If the exact time information of an appearing viruses is not needed, Dynamic Time Warping (DTW) [BC94] can be used to extract features from time series. DTW is widely used for speech recognition and is also well suited for matching step templates. With DTW, signals can be matched despite of being stretched or compressed in the time domain. DTW uses a warping path, which aligns elements of a template vector to the signal. [BC94]

As template vector the following simple pattern is used, that corresponds to a step from −1 to 1:

$$\vec{p} := (-1, -1, 1, 1)^{\top} \tag{5.50}$$

The time series of virus signals $\widetilde{V}_{[t_1,t_2]}(x, y)$ from time t_1 to time t_2 at position (x, y) is defined according to Definition 12. To match the intensities in the time series with the intensities in the template vector, each time series is normalized to zero mean and standard deviation one.

The used distance measure $\delta : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{R}$ between a single element of the template vector and a single element of the time series $\widetilde{V}_{[t_1,t_2]}(x,y)$ is given as magnitude of the difference [BC94]

$$\delta_{\widetilde{V}_{[t_1,t_2]}(x,y),\vec{p}}(i,j) := \left|\widetilde{V}_{t_1+i}(x,y) - p_j\right|. \tag{5.51}$$

A warping path $\vec{w} = (w_1, \dots, w_k)^\top$ can be defined with help of the following two equations. First, the points in the path are defined with $n = t_2 - t_1 + 1$ as the size of the time series and m as the size of the template vector as [BC94]

$$w_l := \begin{cases} (0,0) & \text{for } l = 1 \\ (a,b) & \text{for } 2 \le l \le k-1 \\ (n-1, m-1) & \text{for } l = k \end{cases}, \tag{5.52}$$

with $a, b \in \mathbb{N}_0$, $0 \le a < n$, $0 \le b < m$. Second, in order to form a warping path out of the points, the condition [BC94]

$$w_l \in \left\{(a'+1, b'), (a', b'+1), (a'+1, b'+1) \;\middle|\; (a', b') = w_{l-1}\right\} \tag{5.53}$$

has to hold for all points $w_l \in \{w_2, \dots, w_k\}$.

Finally, the dynamic time warping distance DTW is calculated by minimizing the distance between the time series and the template over all valid warping paths $\vec{w}$ as [BC94]

$$\mathrm{DTW}(\widetilde{V}_{[t_1,t_2]}, \vec{p}, \vec{w}, x, y) := \min_{\vec{w}} \left(\sum_{l=1}^{k} \delta_{\widetilde{V}_{[t_1,t_2]}(x,y),\vec{p}}(w_l) \right). \tag{5.54}$$

The best warping path $\vec{w}$ can be calculated with help of dynamic programming. Cumulative distances are calculated in a matrix of size $(n+1) \times (m+1)$ and then the best path can be found by tracing it back in the matrix. If, like in this case, only the distance is needed, the back tracing can be left out. [BC94]

To speed up the calculation and to prevent that non-virus signals can achieve a high matching score, only time series with a reasonable step height are matched. This is done by checking if the step height of the time series, calculated by the difference of the average value of the first half and the second half, is in the range of possible intensities for the expected virus size. This ensures that the intensity is considered for the detection, even though the time series are normalized to zero mean and standard deviation one.

With DTW the temporal location of the step is of only minor importance. The downside of this is that the exact temporal information about the appearance of the virus is lost. The advantage is that the DTW does not need to be calculated on every frame. For example, if a time series of 16 values is matched, then the next check can be done 8 frames later.

In contrast, the temporal template matching as described in Section 5.4.2 can detect the exact temporal location of the step but needs to be evaluated on every single frame. However, the need to evaluate on every frame can be circumvented by ignoring an amount of data in the middle of the time series, which also causes the exact time information to be lost.

Depending on the task, which should be performed with the PAMONO sensor, one or the other temporal feature is more appropriate. If no virus should be missed and an exact temporal information is not important, the DTW method, calculated on every few frames, is well suited. The detection of a single virus is spread to more frames. If the temporal information is important, time series matching and a temporal noise reduction that preserves the step position can be used. To quantify the virus concentration, especially if the concentration is high, the temporal template size needs to be small to separate viruses attaching short one after the other at the same spot. For low and medium concentrations and if the temporal information is not important, all presented temporal features can be used. The same is true for evaluations of high concentrations, where the exact count of particles is not needed.

Additional Temporal Features

To classify time series several features can be calculated. A set of time series features by Janidarmian, Radecka, and Zilic [JRZ14] is used. They have compiled different time series features, which they use for an automated diagnosis of knee pathology from electromyography and goniometer sensor data. The patients walked wearing the sensors and so produced a set of time series, which were classified using the features. As the features are not application specific, they can be also used for the PAMONO sensor data. Eight useful features are selected, explained in the following.

For given virus signals $\{\widetilde{V}_{t-(n-1)}, \dots, \widetilde{V}_t\}$, given $x \in \{0, \dots, w-1\}$, $y \in \{0, \dots, h-1\}$ and given $a, b, t \in \mathbb{N}_0, t \geq k, 0 \leq a < b < n$, the vector $\vec{i}$ holds k image values:

$$\vec{i} = \left(\widetilde{V}_{t-(n-1)}(x,y), \dots, \widetilde{V}_{t-(n-a)}(x,y), \widetilde{V}_{t-(n-b)}(x,y), \dots, \widetilde{V}_t(x,y)\right)^\top. \tag{5.55}$$

With this vector the following features can be calculated [JRZ14]:

$$\hat{\mu} := \frac{1}{k}\sum_{j=1}^{k} i_j \tag{5.56}$$

$$\hat{\sigma} := \sqrt{\frac{1}{k}\sum_{j=1}^{k}(i_j - \hat{\mu})^2} \tag{5.57}$$

$$\text{coefficientOfVariation} := \frac{\hat{\sigma}}{\hat{\mu}} \tag{5.58}$$

$$\text{peakToPeakAmplitude} := \max(\vec{i}) - \min(\vec{i}) \tag{5.59}$$

$$\text{skewness} := \frac{1}{k \cdot \hat{\sigma}^3}\sum_{j=1}^{k}(i_j - \hat{\mu})^3 \tag{5.60}$$

$$\text{kurtosis} := \frac{1}{k \cdot \hat{\sigma}^4}\sum_{j=1}^{k}(i_j - \hat{\mu})^4 \tag{5.61}$$

$$\text{lagOneAutocorrelation} := \frac{\sum_{j=1}^{k-1}(i_j - \hat{\mu}) \cdot (i_{j+1} - \hat{\mu})}{\sum_{j=1}^{k}(i_j - \hat{\mu})^2} \text{ and} \tag{5.62}$$

$$\text{rootMeanSquare} := \sqrt{\frac{1}{k}\sum_{j=1}^{k} i_j^2}. \tag{5.63}$$

The calculation of these features has been combined with the calculation of the time series matching. This saves execution time and energy as the image values only need to be loaded into the private memory once and do not need to be loaded from global memory again. The eight features are stored in an array of the size eight times the image size. They can be saved for an offline classification of time series but usually they are assigned to polygons as a per virus feature for an online classification with Random Forest. The mapping of per-pixel features to polygons is explained in Section 5.5.3. The classification with Random Forest is explained in Section 5.6.2.

5.4.3 Spatiotemporal Fuzzy-Based Features

The temporal features in Section 5.4.2 do not make use of the spatial information in the PAMONO sensor images. To make these features more robust, a fuzzy-based approach is used. This approach extends ideas for noise removal in image sequences from Mélange, Nachtegael, and Kerre [MNK11] to a segmentation task. It integrates spatial information into the temporal features by taking adjoining temporal features into account. This section is based on the author's publication [LST+13a].

The main idea is that, although the temporal features are calculated per-pixel, the virus adhesions affect more than one pixel. Therefore, neighboring information can be used to decide if a temporal feature corresponds to a virus adhesion, the background, or an artifact. With help of fuzzy rules, expert knowledge can easily be integrated into a detection algorithm.

As introduced in Section 5.3.4, fuzzy rules are used to model the degree of membership to a certain fuzzy set. In this context the fuzzy sets *virus*, *background*, *noise*, and *artifact* are used to model viruses, background, noise, and artifacts in the PAMONO sensor images. The degree of membership to a fuzzy set can be considered as feature. Based on these features a classification can be obtained.

Let $s(x, y, t)$ denote the cosine similarity values, cf. Equation (5.49), as

$$s(x, y, t) = S_{t,\vec{p}}(x, y), \tag{5.64}$$

with x, y as the spatial position and t as the temporal position.

The mapping of the temporal features $s(x, y, t)$ to the fuzzy set $\mu_{\text{step}}^{\pi_1,\pi_2}$ for steps in the signal and $\mu_{\text{downStep}}^{\pi_1,\pi_2}$ for down steps is done by using Equation (5.12), and Equation (5.13), and the soft thresholds $0 \leq \pi_1 < \pi_2$:

$$\mu_{\text{step}}^{\pi_1,\pi_2}\big(s(x, y, t)\big) := \mu_{\text{largePositive}}^{\pi_1,\pi_2}\big(s(x, y, t)\big) \tag{5.65}$$

$$\mu_{\text{downStep}}^{\pi_1,\pi_2}\big(s(x, y, t)\big) := \mu_{\text{largeNegative}}^{\pi_1,\pi_2}\big(s(x, y, t)\big). \tag{5.66}$$

Five fuzzy rules have been defined to model the fuzzy set *virus*. The first fuzzy rule maps the cosine similarity of the temporal template matching to a fuzzy set:

$$\mu_{\text{virus1}}^{\pi_1,\pi_2}(x, y, t) := \mu_{\text{step}}^{\pi_1,\pi_2}\big(s(x, y, t)\big). \tag{5.67}$$

For the following fuzzy rules, let $\{\dots\}$ denote a multiset, let $N_{w_\text{n} \times h_\text{n} \times t_\text{n}}(x, y, t)$ denote the 3D neighborhood multiset as defined in Definition 18, and let r denote the rank function, which calculates the k-th largest element, as defined in Definition 25.

In order to use also Boolean algebra within the fuzzy rules the following function $t : B \to \{0, 1\}$ for a Boolean term B is defined as

$$\tau(b) := \begin{cases} 1 & \text{for } b \text{ is true} \\ 0 & \text{else} \end{cases} . \tag{5.68}$$

The second rule includes the cosine similarity of a neighborhood around the current pixel. If the current pixel has only a low step, then it is a candidate if there are more than 25 % of the neighborhood with a high step. This can be expressed as

$$\begin{aligned} \mu_\text{virus2}^{\pi_1,\pi_2,\pi_3,\pi_4}(x, y, t) := \ & \tau(\pi_1 < s(x, y, t) < \pi_2) \text{ AND} \\ & \mu_\text{step}^{\pi_3,\pi_4}\Big(r\Big(12, \big\{s(x', y', t') \mid (x', y', t') \in N_{7\times7\times3}^{x,y,t}\big\}\Big)\Big), \end{aligned} \tag{5.69}$$

with the fuzzy operator AND as defined in Equation (5.20).

The third rule deals with low steps in a neighborhood. If the current pixel has only a low step, then it is a candidate if there are many adjacent pixels with a low step and at least one pixel with a high step:

$$\begin{aligned} \mu_\text{virus3}^{\pi_1,\pi_2,\pi_5,\pi_6}(x, y, t) := \ & \tau(\pi_1 < s(x, y, t) < \pi_2) \text{ AND} \\ & \Big(r\Big(25, \big\{s(x', y', t') \mid (x', y', t') \in N_{7\times7\times1}^{x,y,t}\big\}\Big) > \pi_5 \text{ AND} \\ & \mu_\text{step}^{\pi_5,\pi_6}\Big(r\Big(1, \big\{s(x', y', t') \mid (x', y', t') \in N_{7\times7\times1}^{x,y,t}\big\}\Big)\Big)\Big). \end{aligned} \tag{5.70}$$

The fourth rule synchronizes detections from different frames, therefore the previous and succeeding values at the same position are considered:

$$\begin{aligned} \mu_\text{virus4}^{\pi_1,\pi_2,\pi_7,\pi_8}(x, y, t) := \ & \tau(\pi_1 < s(x, y, t) < \pi_2) \text{ AND} \\ & \Big(\mu_\text{step}^{\pi_7,\pi_8}\big(s(x, y, t-1)\big) \text{ OR} \\ & \mu_\text{step}^{\pi_7,\pi_8}\big(s(x, y, t+1)\big)\Big), \end{aligned} \tag{5.71}$$

with the fuzzy operator OR as defined in Equation (5.21).

The fifth rule considers also negative cosine values. The PAMONO signal of an attaching virus shows wave like structures around the center, as already shown in Figure 5.8. This information can be used to increase the detection of the center:

$$\begin{aligned} \mu_\text{virus5}^{\pi_1,\pi_2,\pi_9,\pi_{10}}(x, y, t) := \ & \tau(\pi_1 < s(x, y, t) < \pi_2) \text{ AND} \\ & \Big(\mu_\text{downStep}^{\pi_9,\pi_{10}}\Big(r\Big(5, \big\{s(x', y', t') \mid (x', y', t') \in N_{7\times7\times1}^{x,y,t}\big\}\Big)\Big) \text{ AND} \\ & \mu_\text{step}^{\pi_9,\pi_{10}}\Big(r\Big(5, \big\{s(x', y', t') \mid (x', y', t') \in N_{7\times7\times1}^{x,y,t}\big\}\Big)\Big)\Big). \end{aligned} \tag{5.72}$$

To combine these five rules to model the set *virus*, the fuzzy operator OR is used over all rules. As a result, the fuzzy value of the best matching fuzzy rule is used as fuzzy value for the entire set.

More rules are used to model the fuzzy sets *artifact*, *noise* and *background*. These are left out for brevity as the concept should be clear with the five before mentioned rules.

In a last step, the set with the highest fuzzy value is used as result for the current pixel position.

Finally, the fuzzy set with the highest membership value is chosen as output for each pixel. If the set *virus* is assigned, a threshold-based per-pixel classification step can be used to sort out pixels with low membership values. In addition to a per-pixel classification, the fuzzy sets also be used for a per-polygon classification: the degree of membership to the different fuzzy sets can be calculated for each polygon and used as features for classification of polygons.

To meet the soft real-time constraint and to keep the energy consumption low, the OpenCL implementation of this approach has been heavily optimized. The temporal features are stored in a ring buffer, cf. Section 5.3.2. This enables a fast update and coalesced memory access to neighboring entries, needed by the fuzzy rules. The local memory is filled according to Section 4.1.4.

The rank is calculated for every pixel position and for every pixel position it is used by multiple fuzzy rules and with different values of k. To not re-evaluate r for different rank calculations of the same neighborhood, the first k' values of J are sorted with k' as the maximal k for each neighborhood. With a complexity of $\mathcal{O}(|J| \cdot \log |J|)$ for the used algorithm, the rank function $r(k, J)$ is computationally expensive. But as $J \leq 7 \cdot 7 \cdot 3 = 147$ is bounded by a constant the overall complexity is still $\mathcal{O}(n)$.

For the actual OpenCL implementation, a three-dimensional local memory of size $(w_\text{l} + w_\text{n} - 1) \times (h_\text{l} + h_\text{n} - 1) \times 3$ is used for performance reasons. Each of the three dimensions is loaded according to the access pattern presented in Section 4.1.4, one after the other. They store a cutout of $S_{t-1,\vec{p}}$, $S_{t,\vec{p}}$ and $S_{t+1,\vec{p}}$. Once the local memory is loaded and synchronized between the threads, all neighborhood sets for the fuzzy rules only need to access the local memory. For local work groups of size $w_\text{l} \times h_\text{l}$, the number of global memory accesses per analyzed image is reduced to $(\lceil w/w_\text{l} \rceil \cdot \lceil h/h_\text{l} \rceil) \cdot (w_\text{l} + w_\text{n} - 1) \cdot (h_\text{l} + h_\text{n} - 1) \cdot 3$.

As an example, for a typical PAMONO sensor image of size 1024×256 and typical work group sizes, the use of local memory reduces the global memory accesses almost by a factor of 26: with a neighborhood size of $7 \times 7 \times 3$ and a work group size of 16×16, the number of global memory accesses is reduced from $38,535,168$ to $1,486,848$.

To conclude, the fuzzy-based spatiotemporal features are particularly useful to integrate expert knowledge into the calculation of the features, that would otherwise remain unused. With the explained optimizations these features can be calculated and evaluated under soft real-time conditions.

5.4.4 Fuzzy-Based Combination of Features

To obtain a segmentation of the detected viruses, a per-pixel classification is needed with the classes *virus pixel* and *non-virus pixel*. Several features can be important for a good per-pixel classification. To combine several features a fuzzy-based combination of the per-pixel features is used similar as for the fuzzy-based features in Section 5.4.3. A fuzzy set for the likeness of the pixel to be a virus pixel is calculated. This likeness can then be used for classification.

For larger particles, e.g., of size 200 nm, a single temporal feature, as shown in Section 5.4.2, is usually enough for a good pixel classification. With the pixel classification a segmentation can be achieved as shown in Section 5.5. Then, per-polygon features can be calculated, as shown in Section 5.5.3. Finally, a polygon classification can be performed, as shown in Section 5.6.1 and Section 5.6.2.

However, for smaller particles, e.g., below 150 nm, usually several features need to be combined for a robust per-pixel classification. To achieve this all spatial features are converted to fuzzy features, by mapping each of the n feature values from the feature vector $\vec{v} = \{v_1, v_2, \ldots, v_n\}$ to a fuzzy value vector $\vec{\mu} = (\mu_1, \mu_2, \ldots, \mu_n)$. The mapping is done using the parameter vectors $\vec{p} = (p_1, p_2, \ldots, p_n)$ and $\vec{q} = (q_1, q_2, \ldots, q_n)$.

The mapping of each feature v_i to μ_i is done using the parameters p_i and q_i, similar to Equation (5.65):

$$\mu_i := \begin{cases} 1 & \text{for } v_i \geq q_i \\ \frac{v_i - p_i}{q_i - p_i} & \text{for } p_i < v_i < q_i \\ 0 & \text{else} \end{cases} . \tag{5.73}$$

For most of the features, μ_i is equal to v_i as the feature values often can easily be interpreted as a membership to a fuzzy set.

Finally, the fuzzy values of the vector $\vec{\mu} = (\mu_1, \mu_2, \ldots, \mu_n)$ are combined to the fuzzy set *virus* by

$$\mu_{\text{virus}}(\vec{\mu}) := \text{AND}(\vec{\mu}), \tag{5.74}$$

with the fuzzy operation AND given as the minimum value in the vector. Instead of the fuzzy operation AND, other fuzzy operations can be feasible. Which operation is most suitable, depends on how the features are configured. Alternatively, the fuzzy values can be combined using fuzzy rules, as shown in Section 5.4.3.

It should be stressed that the fuzzy-based combination of features is only used to generate a robust per-pixel classification for the segmentation. Once the segmentation is created, cf. Section 5.5, the single features and not the combined features are used for classification. In Section 5.5.3 the mapping of single features to polygons is shown and in Section 5.6.1 and Section 5.6.2 the classification is explained.

5.5 Segmentation

With the segmentation step the spatial information of the virus signal is combined to polygons. It uses the marching squares algorithm, the two-dimensional case of marching cubes [LC87]. It is based on the author's previous work [Lib11]. Therefore, the segmentation method is no new contribution. However, the method has been extended with the calculation of additional polygon features and the detection of overexposed spots.

To briefly sum up the existing segmentation method, the per-pixel classifications are segmented to polygons by tracking the borders of coherent areas with the marching squares algorithm, where all pixels correspond to the fuzzy class *virus pixel* by a certain degree. Then, the polygons are matched over time and space to sort out double detections. In the best case, at this point only one polygon is left for every virus adhesion but as also some artifacts may be detected the polygons are considered as virus candidates. [Lib11]

The matching step in the existing method has been extended to additionally handle detections near overexposed spots. These detections are possibly false positives and need to be sorted out, cf. Section 5.2.4. This is done by matching new polygons not only to other polygons in the area but also to the polygons of the overexposed spots. The detection of overexposed spots is described in Section 5.2.4. New polygons that are near an overexposed spot are sorted out.

5.5.1 Memory Management for Polygons

For the management of the polygons in the GPU memory, a single array of fixed size is used. For every pixel position of the used input image, only a single polygon can be saved and the maximum polygon size is fixed, e.g., to 32 or 64 polygon points. If a polygon exceeds the polygon size, it is simply closed as it reaches the limit of polygon points. Both limitations are due to performance reasons. They limit the number of detectable viruses and also the size of the viruses. In practice these limitations do not show to have any negative effect on the detection quality. A maximum polygon size of 64 is large enough for all virus sizes and typical image sizes. Only large artifacts can exceed the maximum polygon size but these polygons are sorted out anyway. Also, usually viruses do not appear exactly at the same spot and the array size is large enough for long measurements with a high concentration of viruses in the sample.

The simple memory layout has the advantage that the parallel access of the work items does not need to be synchronized as every work item can be assigned to a unique array position. Also, the memory access can easily be coalesced.

However, if the virus concentration is the main objective and at the same time very high concentrations of viruses should be measured in long experiments, this limitation has to be removed. For example, this can be done by moving older detections to another buffer or download the buffer from the GPU to the host memory from time to time.

5.5.2 Polygon Features

With polygon features, high level information of the virus signal can be gathered. In VirusDetectionCL 16 polygon features are calculated. These are described in the following.

Ten shape-based polygon features from Landini [Lan06] are reused in VirusDetectionCL, described in the author's previous publication [Lib11] and only mentioned briefly. For example, these include area, circumference, circularity, compactness, and roundness of a polygon. As a virus adhesion is showing up as a round spot in the images and artifacts are often more elongated, these features are suitable to separate the polygons caused by viruses from these not caused by viruses.

To smooth out the shape of the polygons, the well known morphological operations opening and closing could be performed. Opening is performed by running first an erosion and then a dilatation operation on the polygons, closing on the other hand is performed by running first a dilatation and then an erosion operation. With opening and closing, noise on the contour can be removed.

However, simple shape-based features do not include the virus signal information into the calculation. Therefore, two central moment features [JT81] and one elongation feature [JT81] from Jarvis and Tyson are used. They also include intensity information and they use the intensity weighted centroid of a polygon as a reference. This makes them translation invariant. Additionally, they are size and rotation invariant [JT81].

Using an intensity weighted centroid instead of the standard centroid has some advantage in the PAMONO context. A centroid describes the center of gravity of a polygon. If the asymmetrical waves around a virus adhesion are also covered by the polygon, the centroid could be outside of the bright part of the virus adhesion, which is not desirable. An intensity weighted centroid, on the other hand, is pulled to the brighter parts of the segmentation and is therefore better aligned with the bright part of the adhesion.

For a given polygon P, the intensity weighted centroid $(\bar{x}, \bar{y})$ is calculated as [JT81]

$$(\bar{x},\bar{y}) := \left(\frac{\sum\limits_{\forall (x',y') \in A(P)} x' \cdot \widetilde{V}_t(x',y')}{\sum\limits_{\forall (x',y') \in A(P)} \widetilde{V}_t(x',y')}, \frac{\sum\limits_{\forall (x',y') \in A(P)} y' \cdot \widetilde{V}_t(x',y')}{\sum\limits_{\forall (x',y') \in A(P)} \widetilde{V}_t(x',y')} \right) \tag{5.75}$$

with $A(P)$ as the set of coordinates that are covered by the polygon P. In other words, all pixel coordinates (x', y') covered by the polygon are examined. The sum of x' and y' weighted by the intensity $\widetilde{V}_t(x', y')$ is calculated. Both results are normalized by the summed up intensities.

With the intensity weighted centroid $(\bar{x}, \bar{y})$, the central moments $\mu_{i,j}$ can be calculated. They are defined as [JT81]

$$\mu_{i,j} := \sum_{\forall (x',y') \in A(P)} (x' - \bar{x})^i \cdot (y' - \bar{y})^j \cdot \widetilde{V}_t(x',y'), \tag{5.76}$$

for $i, j \in \mathbb{N}_0$ and $A(P)$ as defined before.

Based on the central moments, the two central moment features and the elongation feature can be calculated. First, the c_2 feature is calculated as [JT81]

$$c_2 := \frac{\mu_{2,0} + \mu_{0,2}}{\mu_{0,0}}. \tag{5.77}$$

Second, the c_4 feature is calculated as [JT81]

$$c_4 := \frac{\mu_{4,0} + 2 \cdot \mu_{2,2} + \mu_{0,4}}{\mu_{0,0}}. \tag{5.78}$$

Third, the elongation feature e is calculated as [JT81]

$$e := \frac{1}{\mu_{2,0} + \mu_{0,2}} \cdot \sqrt{(\mu_{2,0} - \mu_{0,2})^2 + (4 \cdot \mu_{1,1}^2)}. \tag{5.79}$$

These three features have shown to be well suited for the polygon classification task and outperform the aforementioned shape-based features.

Finally, three more intensity-based polygon features are used: Average, maximum, and standard deviation of the virus signal $\widetilde{V}_t$ over all pixel positions covered by the polygon. These can be calculated cheaply in conjunction with c_2, c_4, and e as in both cases the same virus signal values are considered.

For the OpenCL implementation the parallelization is done per polygon. A single thread samples the values, covered by the polygon. The sampling is done row wise by using an existing polygon fill algorithm that uses an ordered edge list [AW81]: A list of the polygon lines is created and sorted according to the x-position. An active list is created and pixels between node pairs need to be filled. Instead of filling the polygon, the corresponding virus signal $\widetilde{V}_t$ is used for the calculation of the mentioned features.

To calculate the features each polygon position is sampled twice: First, to calculate central moment $\mu_{0,0}$, the maximum, the average and the intensity weighted centroid $(\bar{x}, \bar{y})$. Second, to calculate the standard deviation and the features c_2, c_4, and e. Two steps are needed, as the second step uses $\mu_{0,0}$ and the average of the first step.

Overall, 16 features are calculated for every polygon. In the next section a mapping of the per-pixel features to the polygons is presented.

5.5.3 Mapping Per-Pixel Features to Polygons

As already mentioned, the polygons form the set of virus candidates. The virus candidates are classified according to the calculated features. However, several features are calculated per pixel and not per polygon and need to be mapped to the virus candidates to use them for classification. How this mapping can be done efficiently is explained in the following.

Usually, up to 35 per-pixel features are calculated: eight per-pixel features from Section 5.4.2, four Gauss-Hesse features from Section 5.4.1 for each of the, e.g., five scales, three Hesse features, up to three time series features from Section 5.4.2, and one template matching feature from Section 5.4.1.

The per-pixel features are mapped to the polygons by assigning the best feature value within the axis-aligned bounding box of each new polygon to the polygon. A more accurate way to do this would be to get the corresponding best feature value for every pixel that is covered by the polygon. Here however, for speedup reasons the best feature within the axis aligned bounding box around the polygon is used. Identifying pixels within a bounding box is faster than to identify pixels within a polygon, which would require running a scan line algorithm for example. The chance of making an error is low, as pixels slightly outside of the polygons usually have low feature values, e.g., for the Gauss-Hesse feature. However, for the features maximum, average, and standard deviation of the polygon only pixels within the polygon are considered as the bounding box is too inaccurate for these.

Interesting implementation details are that the parallelization is performed per polygon and not per pixel, warp divergence is not avoided, voluntarily, the benefit of coalesced memory accesses is waived, and no local memory is used. The optimization techniques that are useful and recommended under normal conditions [NVI09b] are not useful here, they are even harmful to the performance. As only very few viruses show up at the same time if virus concentration and frame rate are on a reasonable level, a coalesced memory access would load data from global memory, from which most is not needed.

Although usually warp divergence should be avoided [NVI09b], here warp divergence is intentionally not avoided. Warp divergence appears, if threads within one warp access different code. This can happen, e.g., within an if-statement that is only true for part of the threads within the warp. As only one code line can be executed for all threads, all the threads that are not in the active branch perform busy-waiting by executing NO OPERATIONS (NOPs). Here, for most of the warps, there is no polygon to process. Only a negligible amount of threads is performing NOPs because only for a few warps one or more polygon needs to be processed. On the other hand, only submitting the threads to the GPU that have a polygon to process would require some kind of sorting, which is slower than simply submitting all threads. To sum up, avoiding warp divergence would require some overhead, which does not pay off. As only a few threads are affected by the busy waiting, the calculation is faster with warp divergence.

The same is true for the use of local memory and for coalesced memory accesses. Both requires overhead, which slows down most of the threads and speed up only a few threads. These optimizations have been implemented but have shown to have a much worse execution time. Therefore, neglecting these optimizations is the best optimization for mapping features to polygons.

Another problem is how to synchronize the polygons with the features or the features with each other. This is solved by extending the period how long the features are present in the processed images. This is mainly controlled by parameters of the temporal noise reduction and the application of the signal model. If, for example, the gap between the current and background image is set to several frames, a virus adhesion is visible longer in the processed images and consequently the image-based features are available on several frames. However, if the gap is chosen too large, the accuracy of the appearance time is lost and particles are possibly detected twice. Respectively, more effort has to be put into the matching of similar polygons over time to prevent double detections.

5.6 Classification

After the segmentation and mapping of the features, virus candidates with assigned features are available that need to be classified to sort out false positives. The classification of the virus candidates can be done online or offline. For the online classification two methods have been implemented: a simple threshold-based classification and a Random Forest-based classification. The classes virus and non-virus are used, the classification is therefore a two-class classification.

In the future the classification can be extended to also classify different virus sizes. Preliminary work to extend the classes has been done. It has been presented in a collaborative publication [STM+15].

5.6.1 Threshold-Based Classification

The first method for online classification is threshold-based. This threshold-based classification has been done in previous work [Lib11] and is no new contribution. Briefly summarized, a simple and fast classification can be achieved with hard thresholds. For each feature a desired range is defined. The virus candidate is classified as non-virus if at least one feature is not in the desired range.

As an example, most false positives are either too small or too large and can be sorted out by size. Another example is that false positives that are much brighter than actual viruses can be easily be sorted out by the relative intensity. However, this simple method reaches its limit if the false positives are similar to the true positives. This can happen if viruses smaller than, e.g., 150 nm should be detected. Therefore, a Random Forest-based classification has been implemented as explained in the following. The threshold-based classification is also useful for a fast preliminary classification to sort out obvious false positives.

5.6.2 Random Forest-Based Classification

The second method for online classification uses Random Forest [Bre01]. Random Forest combines several weak tree-based classifiers to build a strong classifier.

The use of Random Forest for the online classification is well justified: Siedhoff has evaluated several methods for the classification of virus candidates generated with VirusDetectionCL [Sie16]. Random Forest has been found to give the best results for an offline classification. Liaw and Wiener state that the performance for classification and regression of Random Forest is well, even if compared to support vector machines and neural networks [LW02]. In addition, Random Forest do not tend to overfit if more trees are used [Bre01]. For an online classification the Random Forest is also well suited as it is fast and can be executed in parallel on the GPU.

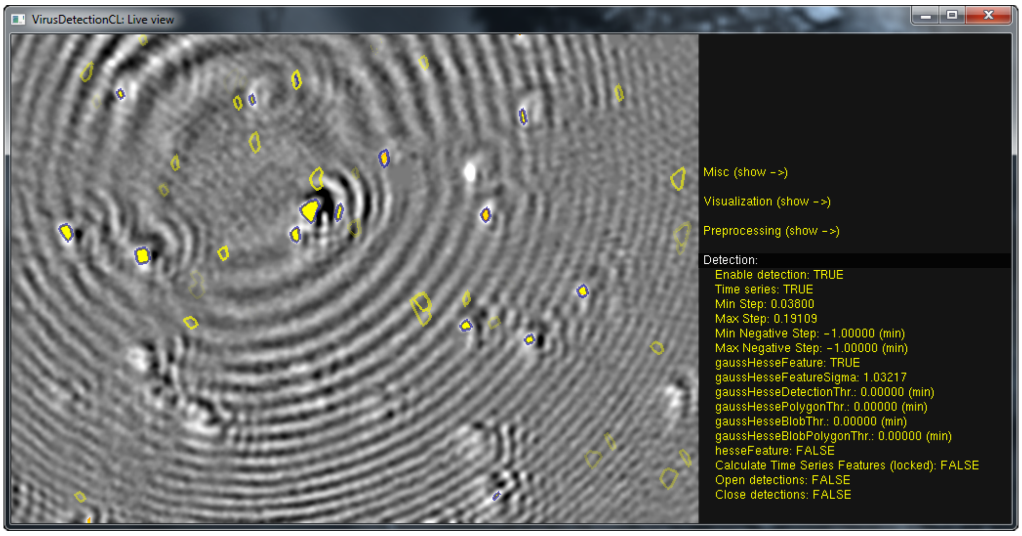
Breiman defines the Random Forest as a collection of tree-structured classifiers. Each classifier is created with an independent identically distributed random vector. For each given input the Random Forest then votes for the most popular class. [Bre01]

The input of a Random Forest classifier is a Random Forest model and a feature vector that should be classified. The output is the assigned class for the feature vector. In VirusDetectionCL all previously calculated features can be used as input. The input consists of the polygons that need to be classified and the associated feature vectors. The Random Forest model is given as a set of decision trees. As output the class virus or non-virus is assigned to each polygon. Polygons that are classified as non-virus are removed. The classification is done a few frames after the polygon is generated as viruses and artifacts like air bubbles or those introduced by vibration take some time to appear fully. Smaller polygons are automatically removed and only the largest are kept. These polygons are then classified. Using this approach reduces the number of small polygons that need to be classified. Especially artifact signals of the same size as viruses are hard to distinguish from virus signals.

The Random Forest model for VirusDetectionCL has been trained and generated with Waikato Environment for Knowledge Analysis (WEKA) [HFH+09]. The author's method RFC-Gen, which is introduced in Chapter 6, has been used to generate OpenCL device code and additional host code. As a result, all needed features are generated, the Random Forest model is evaluated, and viruses are classified on the GPU.

It should be especially emphasized that the size of the data structure with virus candidates is not reduced in order to keep the input of the Random Forest classification small. Even if only one or two virus candidates are found in the current frame, the whole global index space is submitted to the OpenCL device. The effect is, most work items load their associated memory and then notice that there is no virus candidate to classify. This introduces additional overhead. However, a previously performed compactification would require an additional kernel call where also all entries need to be accessed at least once. This would introduce more overhead. As a result, submitting a sparse matrix of virus candidates and then find out which entries need to be processed is faster than to previously find out which entries should be processed and to submit only these entries.

The evaluation for Random Forest in VirusDetectionCL is presented in Section 5.9. More details on the Random Forest-based classification are given in Chapter 6 where the RFC-Gen method for the automatic generation of OpenCL code for the application of Random Forest models are presented.



```
Misc (show ->)

Visualization (show ->)

Preprocessing (show ->)

Detection:
   Enable detection: TRUE
   Time series: TRUE
   Min Step: 0.03800
   Max Step: 0.19109
   Min Negative Step: -1.00000 (min)
   Max Negative Step: -1.00000 (min)
   gaussHesseFeature: TRUE
   gaussHesseFeatureSigma: 1.03217
   gaussHesseDetectionThr.: 0.00000 (min)
   gaussHessePolygonThr.: 0.00000 (min)
   gaussHesseBlobThr.: 0.00000 (min)
   gaussHesseBlobPolygonThr.: 0.00000 (min)
   hesseFeature: FALSE
   Calculate Time Series Features (locked): FALSE
   Open detections: FALSE
   Close detections: FALSE
```

Figure 5.11: The VirusDetectionCL GUI.

5.7 Post-Processing

In the post-processing step, the desired output of the VirusDetectionCL application is prepared. Usually, the output simply consists of the number of viruses but also several other outputs can be generated: the classified viruses with the polygon segmentation, the same as before with additional information about the viruses like intensity and other polygon-based features, the same as before with a full feature vector, and also all unclassified virus candidates with full feature vectors for offline classification.

Furthermore, several image buffers or feature buffers can be provided as output. In most cases, the unmodified input images are used as output, especially if the image stream is provided by a camera. Sometimes the approximated virus signal, feature arrays, or the input image with enhanced SNR is needed. All this can be part of the output. The output conversion is done vice versa to the input conversion explained in Section 5.2.1.

The processing of VirusDetectionCL can also be visualized live while the processing is performed. For visualization an Open Graphics Library (OpenGL) Graphical User Interface (GUI) was implemented based on previous work [Lib11]. VirusDetectionCL can visualize the attaching viruses live, while a data set is processed.

As the virus signal in the PAMONO sensor images is small compared to the background and usually invisible to the naked eye, a contrast enhancement has to be done if the images should be visually inspected. Otherwise, the output would be almost black.

For the contrast enhancement a method from Siedhoff is used. The contrast enhanced image I is calculated from the virus signal approximation $\widetilde{V}$ as

$$I_t(x,y) := m \cdot \operatorname{clamp}\Big(g \cdot \big(\widetilde{V}_t(x,y) - \hat{\mu}(\widetilde{V}_t)\big) + 0.5, 0.0, 1.0\Big), \tag{5.80}$$

with $\hat{\mu}$ as the average approximation as defined in Definition 19, $g \in \mathbb{R}$ as a gain factor, m as the desired maximum output value, and clamp as the clamp function as defined in Definition 8. The virus signal that has extracted with help of the signal model is distributed close to one. By subtracting the average the resulting signal is distributed around zero. With the gain a contrast enhanced signal is obtained that is still distributed around zero. The gain factor is usually chosen in a range of 1.0 to 15. Afterward, an offset of 0.5 is added to obtain a signal distributed around 0.5. The signal is then clamped to clip the signal. Finally, the clamped signal is multiplied by the desired maximum output for visualization, which is usually 255 for 8 bit gray-scale images.

The classification of the per-pixel features and of the polygons is visualized as an overlay of the contrast enhanced image of the viruses. In addition, VirusDetectionCL can visualize most of the features, enabling deeper insights of how good the features perform. If parameters are edited on-the-fly, changes can be seen immediately. The visualization is especially useful for the use in a laboratory, as the qualified personnel receive a live feedback during the measurement. Problems with the PAMONO sensor setup can be fixed and also a detection feedback of changed physical parameters, like the laser intensity or the gain of the signal, can be given. Also, if physical parameters of the PAMONO sensor are changed for further research, instantaneous feedback of the changes can be seen and adjustments to the sensor setup can be made interactively with the processing software.

Figure 5.11 shows the GUI of VirusDetectionCL. Several aspects can be seen. The PAMONO sensor image with applied noise reduction and applied signal model is shown in 8 bit gray-scale image with enhanced brightness and gain. Detected viruses are visualized with yellow polygons. Older detection are gradually faded out to not distract from the new detections. Virus candidates are visualized with blue polygons. The virus candidate pixel classification is visualized as orange to yellow pixels inside of the blue polygons. Parameters can be displayed on the right. They can be edited even during the processing.

Additionally, some general characteristics of the PAMONO sensor signal can also be seen in Figure 5.11. The displayed PAMONO image is dominated by an artifact covering the whole image with strong concentric circles wave pattern. These artifacts often appear in the sensor images, they can be caused by dust or air bubbles in the system. Some effort has been spent to keep the detection uninfluenced by these distortions. Another aspect that can be clearly seen in the image is that the virus signal appears to have different shape depending on the y-position in the image. This is caused by the different focus of the viruses. In the first horizontally quarter of the shown image the particles are out of focus and show weak intensities, in the next quarter they are perfectly in focus and show high intensities and in the last two quarter they are out of focus again. This is due to the fact that the camera views the sensor in an angle as depicted in Figure 2.1.

5.8 Parameters

An excerpt of the most important parameters for VirusDetectionCL is shown in Table 5.1. For each parameter the name, the range, and a short description is given. The table is structured in five blocks that correspond to the pipeline design of pre-processing, signal restoration, feature extraction, segmentation, and classification, cf. Figure 5.1. A full version of this table is provided in Table A.1.

Exemplary, the parameters *currentImageRefs* and *segmentationThreshold* should be explained. The parameter *currentImageRefs* is used in the application of the PAMONO signal model, as shown in Section 5.3.1, to approximate the PAMONO virus signal. To reduce the noise in the PAMONO signal, usually several images are combined to obtain a better approximation of the signal. With the parameter the number of images can be specified that are used, cf. Equation (5.7). The resulting image is then used to apply

Table 5.1: Excerpt of VirusDetectionCL parameters. For a full list see Table A.1.

Name	Range	Description
brightnessCorrection	$\{0, 1\}$	Flag to enable the brightness correction.
detectOverexposedSpots	$\{0, 1\}$	Flag to enable the detection overexposed spots.
temporalNoiseReductionOption	$\mathbb{N}$	Option to select the temporal noise reduction method.
currentImageRefs	$\mathbb{N}$	Number of images to use for the signal approximation.
backgroundImageRefs	$\mathbb{N}$	Number of images to use for the background approximation.
gaussImageSigma	$\mathbb{R}^+$	The σ for the Gauss filter.
timeSeriesOption	$\mathbb{N}$	Option to select the method for the time series features.
timeSeriesDistancePatternSize	$\mathbb{N}$	The size of the temporal pattern.
timeSeriesAutoUpdateVariablePatterns	$\mathbb{N}_0$	Adapt ideal patterns with the signal model.
timeSeriesDistanceMin/MaxStep	$[0, 1]$	Range for the step sizes to detect different virus sizes.
timeSeriesDistanceMin/MaxNegativeStep	$[-1, 0]$	Range for the negative step sizes.
fuzzyDetectionEnhancementX1	$[0, 1]$	Parameter for the fuzzy detection enhancement.
templateMatchingFeature	$\{0, 1\}$	Calculate the template matching features.
gaussHesseFeature	$\{0, 1\}$	Flag to calculate Gauss-Hesse features.
gaussHesseFeatureSigma	$\mathbb{R}^+$	The σ for the Gauss-Hesse feature.
gaussHesseFeatureNumScales	$\mathbb{N}$	Number of scales for the Gauss-Hesse feature.
segmentationThreshold	$[0, 1]$	Influences which pixels are used for segmentation.
mergingMaxDistance	$\mathbb{R}^+$	The maximum distance to combine two virus detections.
mergingMaxFrameDistanceForPolygon	$\mathbb{N}_0$	The maximum number of frames between two detections to combine them to one.
classifyWithThresholds	$\{0, 1\}$	Flag to use threshold-based classification.
classifyWithRandomForest	$\{0, 1\}$	Flag to use Random Forest classification.

the PAMONO signal model. As a result, the virus signal approximation is obtained, cf. Equation (5.8).

The parameter *segmentationThreshold* ranges from zero to one. With this parameter the pixel classification is segmented into pixels that pertain to a virus adhesion or not. For each pixel a fuzzy set for the virus pixel likeness is calculated, cf. Section 5.4.4. The segmentation threshold is then a defuzzyfication threshold for the virus pixel fuzzy set. Pixels with a membership to the virus pixel fuzzy set equal or greater than the segmentation threshold are classified as virus pixels. Afterward, these pixels are enclosed by a polygon.

5.9 Evaluation

In this section evaluations for execution time and detection quality are performed. Additional evaluations of VirusDetectionCL are provided in Chapter 7 and Chapter 8 as part of the evaluation of the PSE and DSE methods.

5.9.1 Evaluation of the Execution Time

VirusDetectionCL has been tested on multiple operating systems and on multiple hardware platforms. This includes several GPU platforms and also some CPU platforms. With VirusDetectionCL the evaluation a PAMONO data set with 4000 images can be performed in two minutes on an NVIDIA®GTX-480 GPU. For an expert a manual evaluation of the same data set takes two days [Sie16].

On average, VirusDetectionCL can analyze images faster than the recording speed of the camera of about 25 fps, cf. Section 7.4. Thus, the soft real-time constraint of processing faster than 40 ms per frame could be achieved with an efficient parallel design and the use of GPUs.

Regarding the execution time it should be especially emphasized that the VirusDetectionCL pipeline has a complexity of $\mathcal{O}(n)$ in Bachmann-Landau notation, with n as the number of pixels in the input images. Measurements on different hardware platforms confirm that the run time scales linearly with the number of pixels in the images, e.g., as shown in Table 7.2. The main reason to use only algorithms from this complexity class is the soft real-time constraint. The second reason is to save energy, in particular on embedded devices. In general, a lot of effort has been spent to achieve a good detection quality, even with the restriction of a fast and energy efficient processing.

5.9.2 Evaluation of the Random Forest Classifier

The Random Forest classifier has been evaluated on one of the most difficult data sets, which contains 100 nm VLPs. A balanced training set with 462 virus class instances and 469 non-virus class instances has been generated by using the SynOpSis approach, see Section 4.5. WEKA [HFH+09] has been used to train the Random Forest. The generated Random Forest has then been applied to unseen testing data set by using the RFC-Gen approach, which is introduced in Chapter 6.

It should be noted that this evaluation is a proof of concept. The training data set has not been optimized to provide a good input to the Random Forest. By optimizing

Table 5.2: Stratified 10-fold cross-validation for the Random Forest classifier on 931 samples.

Quality measure	Virus class	Non-virus class	Weighted avg.
F_1 Score	0.934	0.939	0.937
Precision	0.968	0.910	0.939
Recall	0.903	0.970	0.937
ROC Area	0.969	0.969	0.969

the parameters of the algorithms that generate the features, more useful features can be generated which result in a better performance of the Random Forest classifier. The performance of the Random Forest classification in VirusDetectionCL has also been evaluated in more detail in [Sie16] with the optimization stage of the SynOpSis approach, cf. Section 4.5.

Table 5.2 shows the results of a stratified 10-fold cross-validation [RPC84] on the unseen testing data set. F_1 score, precision, recall, and the area under the Receiver Operating Characteristic (ROC) curve have been evaluated. For the virus class an F_1 score, as defined in Definition 37, of 0.934 could be achieved and for the non-virus class 0.939. Precision and recall, as defined in Definition 35 and Definition 36, show similar performance. For the ROC area [HTF13] for both classes 0.969 could be achieved. This is a very good performance on this difficult data set.

Overall, the performed evaluation on one of the most difficult data sets confirms the good performance of the Random Forest-based classification. The Random Forest-based classification outperforms the simple threshold-based classification. For ESs the threshold-based classification is still useful, because the evaluation of a Random Forest model on an ES can easily consume too much memory and exceed the soft real-time limit.

5.9.3 Evaluation of Detection Quality with the SynOpSis Approach

The detection quality of VirusDetectionCL has been evaluated with the SynOpSis approach, cf. Section 4.5, in the co-authored publication [SLW+14]. This evaluation has mainly been performed by Siedhoff. Real sensor data from data set $\text{Ds4}_{\text{10Apr13-1}}$, cf. Table 2.1, was used to synthesize four new data sets with different virus template and background characteristics and it was also used to validate the results. Four experiments are conducted: $\text{Exp}_{\text{real/real}}$, $\text{Exp}_{\text{real/synth}}$, $\text{Exp}_{\text{synth/real}}$, and $\text{Exp}_{\text{synth/synth}}$.

In $\text{Exp}_{\text{real/real}}$ real background and real virus templates are used, in $\text{Exp}_{\text{real/synth}}$ real background and synthetic virus templates, in $\text{Exp}_{\text{synth/real}}$ synthetic background and real virus templates, and in $\text{Exp}_{\text{synth/synth}}$ synthetic background and synthetic virus templates. For the real background, frames without viruses are used from data set $\text{Ds4}_{\text{10Apr13-1}}$. For the synthetic background, a single frame from $\text{Ds4}_{\text{10Apr13-1}}$ is used to generate every frame of the new data set with Gaussian noise added to each frame. The real virus templates have been extracted from $\text{Ds4}_{\text{10Apr13-1}}$ by manual segmentation of several viruses. For the synthetic templates, a single virus signal is used for all templates. The templates are positioned in the data set with a uniform distribution. As objective recall,

Table 5.3: Evaluation of VirusDetectionCL with the SynOpSis approach on synthetically generated sensor data. The data sets were generated by using real and synthetic virus templates on real and synthetic background signal. Modified version of [SLW+14].

		Real background signal		Synthetic background signal	
Data	Measure	Real virus templates	Synthetic virus templates	Real virus templates	Synthetic virus templates
Training	Precision	1.000	1.000	1.000	0.998
	Recall	0.948	0.953	0.980	0.898
Test	Precision	**1.000**	1.000	1.000	0.998
	Recall	**0.943**	0.927	0.987	0.855
Real	Precision	**0.967**	0.948	0.953	0.949
	Recall	**0.929**	0.954	0.929	0.834

cf. Definition 36, is used and as additional constraint a certain precision, cf. Definition 35, needs to be achieved. In other words, the number of true positives should be maximized, the number of false negatives should be minimized and not too much false positives should be detected. A certain degree of false positives is acceptable because it might be possible to sort them out afterward.

In Table 5.3 results from the single-objective optimizations with SynOpSis are shown. The most interesting results are highlighted in bold font. For experiment $\textsc{Exp}_{\textsc{real/real}}$ a recall of 0.943 with a precision of 1.000 could be achieved on the testing data set and a recall of 0.929 with a precision of 0.967 on manually the segmented data set $\textsc{Ds4}_{\textsc{10Apr13-1}}$ that was not generated with SynOpSis.

Additional evaluations of the detection quality of VirusDetectionCL have been performed by Siedhoff [Sie16] which should be listed here only for the sake of completeness. Thus, the following evaluations are no contribution of the author.

The detection quality has been evaluated with SynOpSis on data sets with 100 nm and 200 nm particles. Table 5.4 shows the averaged result of a global optimization with three folds per data set. VirusDetectionCL was optimized for recall, cf. Definition 36 and a low double detection rate of particles. In the best case, all viruses are found and only detected once. However, the number of false positives was not part of the optimization. In theory, VirusDetectionCL could simply detect a virus candidate at every position at every time and the recall would be 1.0. However, it has shown that this unwanted behavior does not occur in practice. This is due to the process of how the polygons are created and matched with other polygons as described in Section 5.5. If too many artifacts are detected this would lead to less detected viruses as artifacts and viruses are combined to one polygon. Thus, optimizing for recall shows good results. The classifier in SynOpSis, on the other hand, was optimized with precision and recall as objectives. Hence, the classifier is optimized to sort out false positives while keeping the true positives. [Sie16]

All evaluations are performed on real sensor data, which is more difficult than synthetically generated data. The data sets are of different quality with regard to noise level and

Table 5.4: Evaluation of VirusDetectionCL with the SynOpSis approach on real sensor data. Averaged values over three folds. Adapted from [Sie16].

Data set	Detector recall	Classifier precision	Classifier recall	Deviation rate
$\mathrm{Ds10}_{\mathrm{200NM\text{-}HQ}}$	0.961	0.961	1.000	−0.067
$\mathrm{Ds11}_{\mathrm{200NM\text{-}MQ}}$	0.884	0.925	0.998	−0.018
$\mathrm{Ds12}_{\mathrm{200NM\text{-}LQ}}$	0.814	0.909	0.968	−0.090
$\mathrm{Ds13}_{\mathrm{100NM\text{-}HQ}}$	0.715	0.870	0.901	−0.241
$\mathrm{Ds14}_{\mathrm{100NM\text{-}LQ}}$	0.887	0.769	0.901	0.134

presence of artifacts. The quality of the data sets is indicated by *HQ* for high quality, *MQ* for medium quality, and *LQ* for low quality. [Sie16]

The detector recall varies from 0.715 to 0.961. This shows the overall performance of VirusDetectionCL for being able to detect all viruses in the sensor data. Classifier precision and recall show how good SynOpSis can classify the found virus candidates with the generated features of VirusDetectionCL. The classifier precision is in a range of 0.769 to 0.901 and the classifier recall is in a range of 0.901 to 1.0. Finally, with the deviation rate, that varies from −0.241 to 0.134, the overall analysis quality of VirusDetectionCL combined with SynOpSis. The deviation rate provides the deviation to the ground truth. It quantifies the relative deviation of the detected viruses from the actual number of viruses. This indicates how much the actual number of viruses is over- or underestimated. [Sie16]

Finally, another important result is that VirusDetectionCL can handle an SNR below four. As has been shown by Siedhoff, viruses with a median SNR below 2 can be detected with VirusDetectionCL. [Sie16]

5.10 Summary and Conclusion

In this chapter the VirusDetectionCL approach has been presented, which is a computer vision-based method for the detection of biological viruses in PAMONO sensor images. VirusDetectionCL can automatically detect and count individual viruses in PAMONO sensor signal.

To sum up, VirusDetectionCL is a streaming approach with an image processing pipeline that makes use of only a few previous sensor images. The pipeline consists of the steps pre-processing, signal restoration, feature extraction, segmentation, classification, and post-processing. The pipeline is reconfigurable with algorithmic options that replace or extend steps.

The first step deals with changing brightness levels, vibration, and overexposed spots. In the second step the virus signal is extracted from the PAMONO signal. This includes application of the signal model and noise reduction. In the third step features are extracted from the virus signal. This includes per-pixel features that can be used in the next steps. In the fourth step a segmentation of virus candidates is performed. In this step also more high level features are calculated for each polygon. The fifths step performs a classification of the virus candidates. This is done either by a threshold-based approach or with a Random Forest. The last step consists of optional visual processing of the data and creating the output.

As introduced in Section 5.1, several challenges exist to handle the PAMONO signal. Small signal, noise, irregularities in the signal, overlapping signal, artifacts, irregularities in the background, drift of intensity, vibrations, out of focus areas, and fast processing per frame. All these challenges have been addressed.

Drift of intensity, vibrations, and irregularities in the background are handled by the pre-processing step, cf. Section 5.2. To handle the small signal and the noise, several methods for signal restoration and noise reduction have been introduced. Several spatial, temporal, and spatiotemporal features are calculated that help to distinguish between actual virus signal and artifacts.

With the segmentation methods, cf. Section 5.5, the spatial information of the virus signal is combined to polygons, per-pixel features are combined to per-polygon features, and overexposed spots and double detections are dealt with.

With the Random Forest and threshold-based classification, cf. Section 5.6, signals from viruses and artifacts can be distinguished. Additionally, irregularities in the signal and signals at out of focus areas can be addressed.

Regarding the fast processing speed, it could be shown that the automatic detection and counting of viruses is more than 300 times faster than a manual evaluation by an expert. A data set with 4000 images can be evaluated in less than three minutes. A processing time below 40 ms per frame could be achieved with a common desktop GPU. This fast processing also enables real-time processing of the sensor data as fast as it is recoded by the camera. This could only be achieved by a careful design of every algorithm in the detection pipeline. An accelerated diagnosis opens up the possibility to use the PAMONO sensor as an epidemic early warning system or at places with a high passenger volume, e.g., at airports. For a mobile use additionally the energy consumption plays an important role. The foundation of low energy consumption has been made by considering only algorithms with low complexity and by handling the memory accessed efficiently. It is handled in more detail in Chapter 7 and Chapter 8.

Finally, the most important challenge of SNR below four could also be solved. With VirusDetectionCL, viruses with a median SNR below 2 can be detected [Sie16]. This is a very encouraging result because most methods for blob detection and cell tracking, which are both similar to the virus detection task, can not handle an SNR below four [CSG05; SLN+09]. All this enables the detection of small viruses with a size down to 100 nm and slightly below.

Chapter 6

Automatic GPU Code Generation with RFC-Gen for the Application of Random Forest Models

Contents

6.1 The RFC-Gen Method 111
6.2 Limitations of the RFC-Gen Method 113
6.3 Summary and Conclusion 113

This chapter introduces a method for automatic OpenCL code generation for the application of Random Forest [Bre01] models. The method is called Random Forest Code GENeration (RFC-Gen).

As shown in Section 5.6.2, Random Forest is well suited to classify virus candidates in the VirusDetectionCL pipeline. Once a Random Forest model is trained, it can be used to classify several data sets. With the RFC-Gen method a trained Random Forest model can be automatically translated to optimized OpenCL code for the application of the model.

The chapter is structured as follows. First, the RFC-Gen method is introduced in Section 6.1. Second, limitations of the method are listed in Section 6.2. Last, the chapter closes with summary and conclusion in Section 6.3.

6.1 The RFC-Gen Method

Random Forest Code GENeration (RFC-Gen) is an automatic OpenCL code generation for the application of Random Forest models. It uses the Random Forest implementation in WEKA [HFH+09] to train and evaluate Random Forests and then converts the forests into optimized OpenCL code.

The input is a WEKA file with training data, optional a WEKA file with testing data, and a description of some details of how the code should be generated. The output is an OpenCL kernel with the code to apply the learned Random Forest, the needed host code for this kernel, code for an automatic configuration of needed parameter settings, and a performance evaluation of the Random Forest model on the given testing data.

To generate the OpenCL kernel, RFC-Gen performs the following steps. First, RFC-Gen determines which features are needed to apply the Random Forest model. Second, RFC-Gen generates an OpenCL kernel interface with the needed features provided as parameters by using the additional description. With the description the features in the Random Forest model can be mapped to the correct OpenCL global memory arrays. Third, needed index calculations are performed and counting variables are added for each output class of the model, which is usually the virus and non-virus class. Fourth, a fast check is added, if for the current work item a polygon needs to be processed. Every polygon is only classified once. As a virus adhesion takes some frames, the classification is done after a specified number of frames or after the last input image is processed. Fifth, the needed features are loaded into local variables of the correct type. Which type is needed and how the global memory needs to be accessed to load the feature is also included in the description. The local variables are named equal to the feature names in the tree, which makes the OpenCL code easily to generate and to read. Sixth, all trees are added. This is done by first parsing the Random Forest model into a set of tree data structures. Each tree is then translated into OpenCL code by traversing each tree with breadth-first search and adding appropriate `if-else` conditions for the split criterion. The feature names are equal to the local OpenCL variables. At the leaf level the corresponding class variable is counted up. Lastly, code for the classification of the polygons is added. For the two class classification, polygons that are classified as non-virus are removed from the set of polygons.

To speed up the evaluation of the trees, RFC-Gen automatically generates termination conditions to stop the evaluation of the trees early without scarifying accuracy. The first termination condition is inserted after 50 % plus one trees, the second one at 60 % of the trees for example, and so on. The termination conditions checks if more than 50 % of the total trees vote for the virus or non-virus class. If this is true, the correct class has already been found and can be returned. All the following trees do not have to be evaluated. The spacing between checks can be modified up to one check after every tree. Depending on the size of the trees, the overhead of additional checks can pay off.

If approximate computing should be used, the termination conditions of RFC-Gen can be based on less than 50 % votes to one class. For example, if 20 % of the votes are for the non-virus class and 1 % for the virus class, it might be feasible to vote for the non-virus class and skip the evaluation of 79 % of the trees. Good termination conditions can be automatically identified by MOGEA-DSE, cf. Chapter 8. As the trees are generated with a good portion of randomness, the error probability can be kept low.

What is also possible with RFC-Gen is an optimized access to global memory in a hardware/software codesign manner. If, for example, one global memory access is only needed in one of the last trees, it is probably not needed to be loaded most of the times as enough votes might be counted before the tree needs to be evaluated. If the global assess is delayed until it is needed, global memory accesses can be saved. This can save energy and execution time, as global memory accesses are expensive and should be avoided [NVI09b].

Another strategy is to distribute the global memory accesses more evenly within the code. There is a latency of 400-600 cycles for one memory read to complete [NVI09b] on NVIDIA®GPUs. If there is enough code, which can be processed in the kernel before the memory entry is needed, this can provide a huge speedup [NVI09b]. So the RFC-Gen can assure that there is some independent calculation after the memory reads and before the memory writes. This is part of future work on RFC-Gen, currently all memory reads are done right in the beginning, which is usually a good strategy but might not be optimal.

The reordering of the trees and memory accesses is also possible and part of the future work. If no approximate computing is used, reordering of trees does not affect the detection quality. If however approximate computing should be used and decisions are made on less than 50 % of the votes for one class, it has to be asserted that the randomness of the order of trees is still given. Otherwise, it could happen, that if weak trees are evaluated first the early termination condition is only based on a lot of weak trees.

To generate the host code, RFC-Gen generates a *C* function which can call the generated OpenCL kernel. First, the needed parameters for the OpenCL kernel are set. This includes the device memory with the polygons that should be classified and device memories for all needed features. Second, error handling is added. Third, code to enqueue the OpenCL kernel is added. Finally, code to submit the kernel is added. The generated host code assures that only the needed features are loaded.

To generate the automatic configuration function for VirusDetectionCL, RFC-Gen uses a description for each feature of how the feature calculation can be enabled and configured if it is needed as input. RFC-Gen generates a *C* code configuration function, where for each needed parameter the needed settings are chosen. For example, if the multi-scale Gauss-Hesse feature, cf. Section 5.4.1, is used in the Random Forest model, the configuration function enables this feature and determines the number of needed scales that need to be calculated. If only a single scale is used, the single-scale Gauss-Hesse feature is used. The resulting configuration function can be called within VirusDetectionCL.

As a result, the calculation of features does not need to be enabled or disabled manually. If a new Random Forest model is trained and loaded that uses other features, the dependencies are automatically updated. The configuration function assures that the features, accessed by the Random Forest, are generated before the model is applied.

6.2 Limitations of the RFC-Gen Method

The RFC-Gen approach has limitations in some use-cases. First, all trees are stored in program code and therefore two size limitations are given. The code needs to fit into the program code memory of the GPU and it must fit into the program code memory of the SMPs. Second, if the Random Forest changes, the GPU code has to be compiled again. This can happen on-the-fly by the OpenCL compiler of the GPU driver but takes some time. Third, it might be more efficient for some use cases to parallelize the evaluation of the trees and not to parallelize the evaluation of the forest. RFC-Gen evaluates the whole forest in parallel for several inputs. If the forest consists of many trees and the number of parallel evaluations is small, a parallel evaluation of the trees is presumable faster. Also, the warp divergence can be high for trees where threads in one half warp chose different paths. However, for VirusDetectionCL warp divergence is not a problem as usually only a few virus candidates are detected in each frame.

6.3 Summary and Conclusion

In this chapter the method RFC-Gen has been presented, which can be used to generate GPU code for the application of Random Forest models.

A highlight of RFC-Gen is that a Random Forest model can be automatically coded in OpenCL code that can be loaded by the GPU and shared among all threads, which is a very fast solution. With RFC-Gen the model and the code can be kept separated and combined as required. Therefore, the RFC-Gen method combines maintainability with performance, which can rarely be found in one method.

Another advantage of RFC-Gen is that it enables further opportunities for code optimization. The order in which the trees are evaluated can influence the performance because potentially not all trees need to be evaluated. If trees that are expensive to evaluate are evaluated late, their evaluation can be possibly omitted. The order and the position in the code of load operations also play an important role for the performance which opens new opportunities for optimization.

Furthermore, RFC-Gen determines the needed features for the model and automatically adapts the pipeline configuration such that these features are generated.

In conclusion, RFC-Gen is well suited to evaluate Random Forest models on a GPU. Although the approach has some limitations, the advantages outweigh the disadvantages.

Chapter 7

Single-Objective PSE and DSE with SOG-PSE and SOG-DSE

Contents

7.1 Introduction to SOG-PSE and SOG-DSE 116
7.1.1 Motivation 117
7.1.2 Evaluated Scenarios 118
7.2 General Methods for SOG-PSE and SOG-DSE 119
7.2.1 Architectural Design of SOG-PSE and SOG-DSE 119
7.2.2 Parameter Space and Design Space Definition 120
7.2.3 Single-Objective Optimization 121
7.2.4 Considering Several Data Sets for Optimization 121
7.3 Single-Objective PSE with SOG-PSE 121
7.3.1 Methods 122
7.3.2 Experiments 124
7.3.3 Results 125
7.3.4 Discussion 126
7.3.5 Summary and Conclusion 128
7.4 Manually Performed DSE of Actual CPUs and GPUs 129
7.4.1 Methods 129
7.4.2 Experiments 130
7.4.3 Results and Discussion 131
7.4.4 Summary and Conclusion 133
7.5 Single-Objective Energy-Aware Hardware DSE with SOG-DSE 134
7.5.1 GPGPU-Sim Simulator 135
7.5.2 GPUWattch Simulator 136
7.5.3 Methods 138
7.5.4 Experiments 142
7.5.5 Results and Discussion 144
7.5.6 Summary and Conclusion 145
7.6 Single-Objective Power-Aware DSE of GPUs for HPC Systems with SOG-DSE 146
7.6.1 Motivation 146
7.6.2 Methods 147
7.6.3 Experiments 148
7.6.4 Results and Discussion 148

7.6.5 Summary and Conclusion 150
7.7 Limitations of SOG-PSE and SOG-DSE 151
7.8 Summary and Conclusion of the SOG-PSE and SOG-DSE Methods 151

This chapter presents a method for automatic single-objective PSE and a method for automatic single-objective DSE. The first method is named Single-Objective GPGPU Parameter Space Exploration (SOG-PSE) and the second method is named Single-Objective GPGPU Design Space Exploration (SOG-DSE). Additionally, a manual DSE on hardware is performed, which is the foundation for the automatic DSE on simulated hardware.

The structure of this chapter is as follows. First a general introduction to the presented SOG-PSE and SOG-DSE is given in Section 7.1, followed by general methods for SOG-PSE and SOG-DSE in Section 7.2. Then the SOG-PSE method is presented in Section 7.3. Afterward, the focus switches from PSE to DSE with an introduction by a manually performed DSE in Section 7.4. The SOG-DSE method is presented in the sections Section 7.5 and Section 7.6. Finally, the chapter closes with limitations, summary, and conclusion of the two methods in Section 7.7 and Section 7.8.

This chapter is based on the author's publications [LWT12; LST+13a; LSW14; LMS+14].

7.1 Introduction to SOG-PSE and SOG-DSE

SOG-PSE is an automatic, single-objective PSE method. PSE is the exploration of different parametrizations toward one or more objectives. With SOG-PSE it is possible to optimize software parameters for given programs. Typically, this is used to optimize QoR.

SOG-DSE is an automatic, single-objective DSE method. DSE is the exploration of design alternatives, e.g., different software designs, design of new hardware, suitable existing hardware architectures, or hardware configurations toward one objective. Typical objectives are performance, energy consumption, and power consumption. With SOG-DSE it is possible to optimize hardware parameters of GPU architectures for given GPGPU programs. This is useful to identify suitable hardware for a given GPGPU program.

Several objectives can be explored. For SOG-PSE this is consists of different quality measures to explore the QoR. For SOG-DSE execution time, power consumption, and energy consumption are especially in focus.

The two most important research issues for this chapter are the following:

- How can parameters of a given GPGPU program be automatically explored?
- How can suited GPU hardware for a given GPGPU program be automatically explored?

With SOG-PSE and SOG-DSE these research issues can be answered.

The SOG-PSE and SOG-DSE methods are the foundation for the MOGEA-DSE approach in Chapter 8.

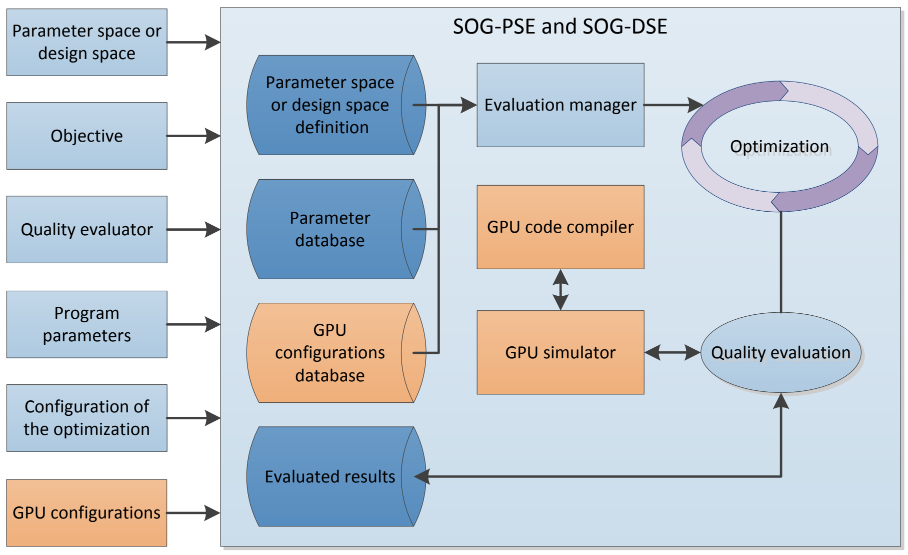


Figure 7.1: Schematic of the general architectural design of SOG-PSE and SOG-DSE. The additional parts of SOG-DSE are highlighted.

7.1.1 Motivation

The methods for PSE and DSE presented in this chapter are motivated by the PAMONO sensor use case and the PAMONO scenarios in Section 1.1. The PAMONO sensor will be used, e.g., in laboratories, hospitals, at airports, or in-the-field. For each scenario a suitable CPS of PAMONO sensor, VirusDetectionCL processing software, and target hardware needs to be found and configured. The requirements that the CPSs need to fulfill in the scenarios differ substantially. For each CPS in the scenarios, the parameters of VirusDetectionCL can be explored with SOG-PSE to ensure a good detection quality, and suited hardware can be explored with SOG-DSE to ensure a fast and energy efficient processing.

An exploration of parameters and hardware is not only important to find and configure the actual systems that are used in practice but also for exploring systems during the design phase in which hardware and software change frequently and the rest of the system needs to be adapted to these changes.

However, the parameter and design space is difficult to explore. For VirusDetectionCL, for example, the number of parameters is quite high. This program has 110 Boolean, integer, and floating point number parameters, which can affect the detection quality. Additionally, several dependencies exist between the pipeline steps, and for each pipeline step a selection of different algorithms can be made. To explore this parameter space manually is time-consuming. For the GPU hardware fewer configurations exist. However, testing out various GPUs from the design space is also time-consuming.

As these tasks are also of interest for many other scenarios, this leads to the research issue of how parameters and suitable hardware can be identified in general for all kind of GPGPU programs.

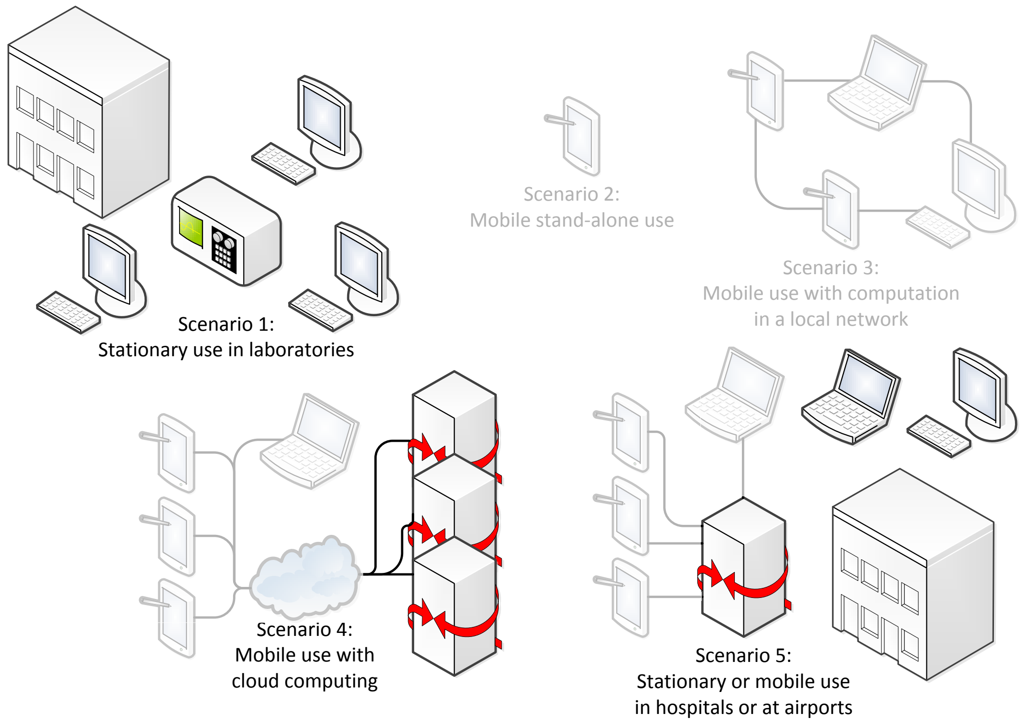

Scenario 1:
Stationary use in laboratories
Scenario 2:
Mobile stand-alone use
Scenario 3:
Mobile use with computation in a local network
Scenario 4:
Mobile use with cloud computing
Scenario 5:
Stationary or mobile use in hospitals or at airports

Figure 7.2: Different scenarios for the PAMONO sensor use case inspected in this chapter.

7.1.2 Evaluated Scenarios

In Section 1.1 several scenarios for the PAMONO sensor use case have been presented. Figure 7.2 shows the three different scenarios that are inspected in this chapter: SCN1_{LAB}, $\text{SCN4}_{\text{CLOUD}}$, and $\text{SCN5}_{\text{HOSPITAL}}$.

Scenario SCN1_{LAB} is a laboratory use case, where different stationary desktop systems are used independently without the need of a central server. Each system processes PAMONO tasks and probably other tasks. Detection quality is the most important objective. Energy consumption and execution time are not crucial.

In scenario $\text{SCN4}_{\text{CLOUD}}$ distributed computing using severs in the cloud is considered. Several mobile devices offload work to the cloud, where it is processed by one or more servers. In this chapter only the server part is considered. For the servers suitable GPU hardware needs to be selected. Several tasks are processed on each server. Beside execution time energy consumption, power consumption, thermal design power, or other objectives can be considered.

Scenario $\text{SCN5}_{\text{HOSPITAL}}$ shows a typical setup for hospitals or airports. Usually, several mobile devices can be used to take measurements where needed. Data can be processed locally or offloaded to a server in the local network. Here, only the desktop, laptop, and server part of this scenario is inspected. Hand-held devices and how offloading can be done is considered in Chapter 8. For each inspected system different hardware and objectives might be of importance.

In addition to the PAMONO sensor use case, 14 industrial, biological, and physical applications are evaluated.

7.2 General Methods for SOG-PSE and SOG-DSE

In the following general methods and design decisions for SOG-PSE and SOG-DSE are introduced: the architectural design, handling of the parameter and design space, single-objective optimization, and how several data sets can be considered for optimization.

7.2.1 Architectural Design of SOG-PSE and SOG-DSE

Figure 7.1 shows a general architectural design for SOG-PSE and SOG-DSE. The parts that differentiate SOG-DSE from SOG-PSE are highlighted in the figure.

For SOG-PSE the architectural design is as follows. The main part consists of the evaluation block with the optimization cycle, an evaluation manager, a quality evaluation, and three databases for the search space, the parameters, and the evaluated results. The inputs for the evaluation are: a definition of the search space, the desired objective for the optimization, a quality evaluator, program parameters, and a configuration of the optimization.

The evaluation manager is responsible for running the evaluation. It configures, starts, and manages the optimization process. In addition, it also manages all inputs and provides them to the optimization and the quality evaluation. The optimization process creates new configurations from the design space and submits them to the quality evaluation.

All parameters are stored in a database as name and value tuple. The value is set to the given start value. This parameter database can also hold parameters that are not optimized.

The genes are built according to the provided definition of the search space and the provided program parameters. For each gene the type and range is assigned. A mapping of parameter name to gene position is stored in conjunction with the parameter space or design space definition.

Quality evaluation evaluates individuals and assigns a quality to it. For each individual that should be evaluated a parameter set is created. This set contains all parameters from the parameter database with values updated according to the current allele of the genes. The mapping of parameter names to gene position is done with the parameter space or design space definition. Afterward, it runs the program that should be evaluated with the current configuration and determines the quality of the result.

For speedup reasons a database with evaluated results is maintained throughout the evaluation. Individuals with the same allele, the same expression of chromosomes, do not need to be evaluated twice. As data structure a hash map is used with the allele as hash key.

Before an individual is evaluated, a query to the database is done to check if its gene configuration has already been evaluated. If this is true, the already evaluated result is used as fitness value for this individual without running the fitness evaluation again.

If a fitness evaluation fails, the corresponding parameter set is stored in another database. An evaluation can fail if the chromosome maps to an invalid parameter set that is not accepted by VirusDetectionCL or if VirusDetectionCL crashes unexpectedly. The stored parameter sets can be inspected later to fix errors.

For SOG-DSE four parts are added to the architectural design of SOG-PSE. As new input one or more GPU configurations can be specified. These configurations are held in a

GPU configuration database. The evaluation manager provides them to the optimization. For the quality optimization a GPU simulator is added. It takes a GPU configuration as input and can simulate the execution of GPGPU programs. Finally, a GPU code compiler is added, which can provide GPU code if the program does not come with pre-compiled GPU code.

Three limitations of this design are that it is not intended for multi-objective optimization, that only a single quality evaluation takes place at the same time, and that a remote evaluation of the quality is possible but not directly provided. A remote evaluation can only be done by providing a quality evaluator that handles this. These limitations are addressed in Chapter 8.

7.2.2 Parameter Space and Design Space Definition

The parameter space or design space that should be explored is the most important input for SOG-PSE or SOG-DSE. This search space is spanned by all considered combinations of software or hardware parameters. The defined parameter or design space should be large enough to not limit the optimization to identify the best parameters or design but small enough to keep the evaluation time low.

For each software or hardware parameter that should be optimized, a data type and a valid range have to be provided. As data types integer numbers, floating point numbers, and vectors are supported.

Listing 7.1 shows an example of how a parameter space is defined. The names of the parameters are given in a simple text file. For reasons of simplicity, this text file can also contain parameters that should not be optimized but for which a fixed value should be provided. For each parameter the first given number is a flag if this parameter should be optimizes or not, the second number gives a start value, e.g., based on previous optimizations or chosen manually. Finally, the third and fourth number provide the valid range of the optimization.

```
backgroundOption=1;1;0;1
backgroundGap=0;16;1;40
averageImage=1;0;0;1
averageImageKernelWidth=1;2;1;7
averageImageKernelHeight=1;3;1;7
medianImage=1;0;0;1
medianImageKernelWidth=1;2;1;7
medianImageKernelHeight=1;3;1;7
gaussImage=1;0;0;1
gaussImageSigma=1;1.5;1.5;3.5
```

Listing 7.1: Parameter space

For example, the *gaussImageSigma* parameter in Listing 7.1 provides the σ for the Gaussian distribution in Equation (5.11). This parameter should be optimized, the first individual starts with $\sigma = 1.5$, and the range for the optimization is $[1.5, 3.5]$. The parameter *backgroundGap* influences the number of frames where on the images with applied sensor model a current detection is visible. It is the distance between the background approximation, cf. Equation (5.6), and the current image approximation, cf. Equation (5.7). This parameter should be chosen small enough to keep only the current viruses visible and large enough to be able to detect them. It is optimized in a range of $[1, 40]$.

For the parameters that should not be optimized, the start value is used in all runs and the given range is ignored. If the parameter should be optimized, the start value is used to initialize only the first individual. All other individuals are initialized with a random value within the valid range of the parameter.

Based on the definition of the parameter space or design space the genes of the EA can be set up.

7.2.3 Single-Objective Optimization

The single-objective optimization in SOG-PSE and SOG-DSE is done with an EA. The input to this EA is a population with several individuals. Each individual represents one point in the parameter or design space that should be evaluated.

The optimization cycle, cf. Definition 47, is as follows: First, the fitness value for all individuals in the population is evaluated toward the single objective. Second, with a tournament selection, cf. Definition 48, a selection of individuals is made based on their fitness value. Third, the selected individuals are recombined to produce new children. Fourth, the gene expression of the children is mutated with a certain probability. Finally, these children form a new population. If the concept of elitism is used, also some parents might be part of the new population. This cycle is continued until a desired quality of the result is achieved, the improvement of the population is below a threshold, or until the maximum number of generations is reached.

For the actual implementation of SOG-PSE and SOG-DSE, the Java Genetic Algorithms Package (JGAP) framework [MR11] from Meffert and Rotstan is used to create and run this EA cycle. JGAP is a framework that integrates GA and genetic programming methods.

7.2.4 Considering Several Data Sets for Optimization

To prevent overfitting to the input data, several data sets should be taken into account for optimization. With SOG-PSE parameters can be optimized to match several data sets. As not every data set has same degree of importance, a weighting factor for every data set can be given. With several data sets and the weighting factors, the same program can be evaluated on different data sets and each evaluated result is weighted according to the importance of the data set.

Thus, for each data set a weighting factor $w_d \in \mathbb{R}$ is defined with $d \in [1, \ldots, D]$ and $D \in \mathbb{N}$ as the number of data sets that should be evaluated. To determine the weighted overall fitness value $f \in \mathbb{R}$, the sum over all weighted fitness values is calculated as

$$f = \sum_{d=1}^{D} w_d \cdot f_d, \tag{7.1}$$

with $f_d \in \mathbb{R}$ as the fitness value for the evaluation of the d-th data set. If no weighting factors are defined, each weighting factor w_d is simply chosen as $\frac{1}{D}$.

7.3 Single-objective PSE with SOG-PSE

In this section the Single-Objective GPGPU Parameter Space Exploration (SOG-PSE) method is developed. SOG-PSE is an optimization method for parameter optimization. With this Parameter Space Exploration (PSE), parameters for CPU and GPU programs can be optimized toward a given objective.

Usually parameters for programs are manually tuned by an expert. However, with an increasing number of parameters, the parameter space can grow rather quickly, making manual tuning more and more difficult and time-consuming. This problem is aggravated if the program or the input to this program change. Previously optimized parameters might no longer be suited for the changed conditions. With SOG-PSE the parameter tuning can be automated in both cases. This is especially suited for a codesign of processing software and sensor hardware. With such an HIL, software and sensor hardware can be developed continuously in small steps and with a fast feedback of the achievable QoR.

For the VirusDetectionCL use case, the detection quality is the most important objective. What makes a parameter tuning toward this objective difficult is that within the VirusDetectionCL pipeline several dependencies exist. For example, the spatial and temporal noise reduction influences how the features can be generated, how the segmentation looks, and how the classification has to handle the detections. The pipeline configuration is also not fixed, elements can be left out or added or algorithms can be switched. All this increases the complexity of the optimization task. As soon as a certain level of complexity is exceeded, the time needed for manual tuning rises strongly because large numbers of configurations have to be evaluated. Additionally, both VirusDetectionCL and the PAMONO sensor setup changed quite often throughout the development. Better methods have been added to VirusDetectionCL and the PAMONO sensor setup also changed with other cameras, gold layers, and optics. If changes on the sensor setup are made, it is often not clear if these changes actually improve the detection quality or even deteriorate it. With SOG-PSE, parameters can be easily tuned or adapted to reflect changed conditions. This provides a fast feedback for changes in software or hardware.

Overall, the SOG-PSE method is a starting point for the other PSE and DSE methods in this work. In the following sections it is developed further toward a DSE and in the next chapter toward a hybrid PSE and DSE.

The following research issues are examined:

- How can a parameter set that delivers a desired quality be automatically identified for a given program?
- Does sequential optimization lead to the same result as a global optimization?

This section is based on the author's publications [LWT12; LST+13a].

7.3.1 Methods

With SOG-PSE first concepts and methods for a PSE are introduced. First, the architectural design of SOG-PSE is explained, followed by the fitness evaluation, and the calculation of the fitness value. Because most steps for a PSE are already explained in Section 7.2, this section is rather short.

Architectural Design of SOG-PSE

The architectural design of the SOG-PSE approach is shown in Figure 7.3. The input of SOG-PSE is a description of the parameter space that should be explored, the objectives that should be optimized, a description for the fitness evaluation, a set of program

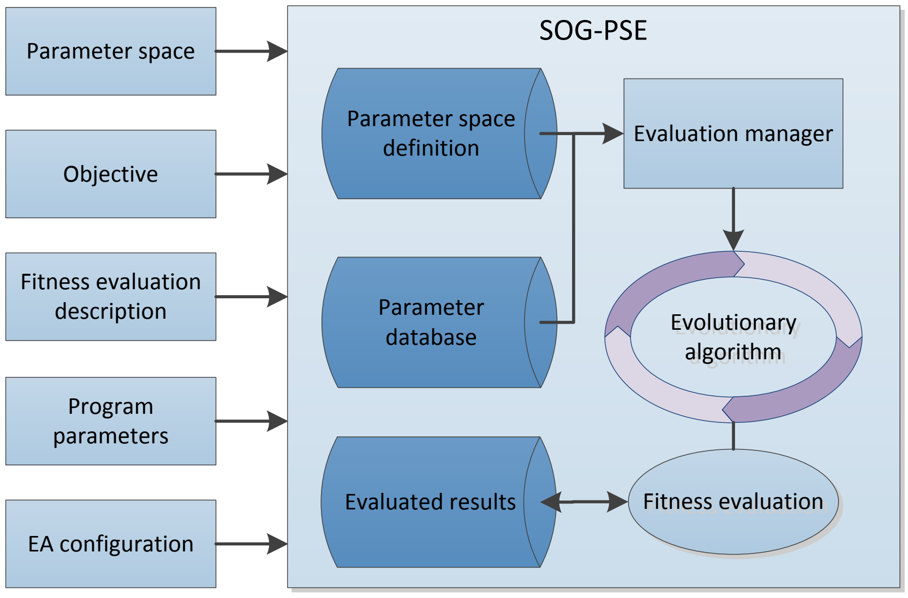


Figure 7.3: Schematic of the architectural design of SOG-PSE.

parameters as default values for the parameter space, and a configuration of the EA. The parameter space definition and the base parameters are stored in databases and provided to the evaluation manager.

The evaluation manager drives the EA. Within the EA cycle new individuals are generated, which are evaluated with the fitness evaluation. All evaluated parameter configurations are stored with the results in a database so that the same gene configurations do not need to be evaluated twice. In SOG-PSE both the EA and the evaluation run locally.

Fitness Evaluation

For each new individual generated by the EA, the fitness needs to be calculated. The fitness evaluation of VirusDetectionCL, other programs can be evaluated similarly, works as follows: First, any previously calculated results are deleted. Second, the current chromosome is converted to a list of arguments. Third, a program call to VirusDetectionCL with the arguments is made. A timeout is used for this call to cancel non-terminating runs after some time to keep the evaluation running. Forth, after the program call is completed the fitness is calculated by matching the found viruses with the viruses in a gold standard file. Fifth, if no elitism is used in the EA, even the best individuals can be removed from the population. Hence, the current chromosome is stored if the current fitness is best. Finally, to prevent that evaluations get lost if the evaluation is canceled or crashes, a checkpoint of the current population is stored every generation. The optimization can be resumed from this checkpoint if more evaluations are needed.

Calculation of the Fitness Value

Several performance measures can be used to calculate the QoR, which is used as fitness value of individuals in the EA. For VirusDetectionCL the available performance measures include accuracy (Definition 34), precision (Definition 35), recall (Definition 36), F_1 score (Definition 37), positive agreement (Definition 38), and Matthews correlation coefficient (Definition 39). Which performance measure is most suited, highly depends on the use case of the PAMONO sensor.

Because each of the performance measures can be calculated from TP, TN, FP, and FN, cf. Definition 30 to Definition 33, the main effort of the fitness calculation method is how to determine these values. As already mentioned, TN is always zero for the PAMONO use case.

To calculate TP, TN, and FP, the received result from VirusDetectionCL is matched with a gold standard. The gold standard is created manually for real sensor data or manually or automatically for synthetically generated sensor data. In the gold standard every virus adhesion is marked as a polygon that is slightly bigger than the round center of the adhesion. The wave like structures are not marked in the gold standard as they can be much bigger than the center region of the adhesion.

A polygon is counted as TP, as successfully detected by the detector, if the spatial and temporal position of the polygon is near to the spatial and temporal position of the gold standard polygon. A spatial nearness is given if the center of gravity of the found polygon is within the gold polygon. Alternatively, spatial nearness can be defined as if the distance of the centroids of the two polygons is below a certain threshold. A temporal nearness is given if only few frames are between the found polygon and the gold polygon. However, it is difficult to specify what a few frames are. Also, depending on the processing methods, some time information is lost if a polygon is detected by the processing software. Details can be found in Section 5.4.2.

Within the processing software some effort has been made to estimate the exact time of appearance of the particle. To not give an unfair advantage or disadvantage to the processing software or to specific methods, the term temporal nearness is dynamically adapted to the used method. In practice this has shown to give reliable results.

A polygon is counted as FP, as wrongly detected by the detector, if no gold polygon is present near the found polygon.

Finally, gold polygons that are missed by the detector are counted as FN. It is determined by the number of gold polygons that did not match to at least one found polygon. If several polygons match to a single gold polygon, these are counted as double detections.

7.3.2 Experiments

Two experiments are conducted for SOG-PSE named $\text{Exp1}_{\text{Fuzzy}}$ and $\text{Exp2}_{\text{SignalStrength}}$. These are described in the following.

For experiment $\text{Exp1}_{\text{Fuzzy}}$ the fuzzy method from Section 5.4.3 are in focus. SOG-PSE is used to optimize sequentially the fuzzy detection parameters only and afterward the detection parameters only using the optimized fuzzy parameters. This is compared to a global optimization of the fuzzy and detection parameters combined.

The questions answered by experiment $\text{Exp1}_{\text{Fuzzy}}$ are:

- Does the fuzzy method improve the detection quality?
- Does sequential optimization lead to similar results as a global optimization?

As baseline measurement, a manually optimized parameter set was used. This was also the starting point for the automatic optimization with SOG-PSE.

Three manually annotated data sets are used, cf. Section 2.2. First, the data set $\textsc{Ds1}_{\text{HIV-VLPs}}$ with images of size 1000 px × 295 px with 100 nm HIV VLPs. Second, the data set $\textsc{Ds8}_{\textsc{Poly200nm-2}}$ with images of size 1000 px × 566 px with 200 nm polystyrene particles. Third, the data set $\textsc{Ds9}_{\textsc{Poly280nm}}$ with images of size 1000 px × 367 px with 280 nm polystyrene particles. For each optimization SOG-PSE has been run for 800 generations with 20 000 evaluations in total.

For experiment $\textsc{Exp2}_{\textsc{SignalStrength}}$ particles with different signal strengths have been simulated with two different noise sources. To create the data sets, the data driven synthesis in SynOpSis, cf. Section 4.5, is used. The particle signal is extracted from $\textsc{Ds7}_{\textsc{Poly200nm-1}}$ in Table 2.1. Four attenuation or gain factors are used: 2.0, 1.0, 0.75, and 0.5. This simulates particle sizes of approximately 400 nm, 200 nm, 150 nm, and 100 nm. The size of particles is in a linear relation to the signal strength [ZKG+09].

The questions answered by experiment $\textsc{Exp2}_{\textsc{SignalStrength}}$ are:

- How does the signal strength affects the detection quality?
- Is the fuzzy method more advantageous for lower signal strengths?
- How does the sensor noise affects the detection quality compared to Poisson noise?

For the background signal two noise sources are used: synthetically generated Poisson noise and real sensor noise. The Poisson noise is generated with the Poisson distribution as defined in Definition 24 and with the parameter λ set according to Equation (4.43). For every frame different Poisson noise is added to a single background image. The real sensor noise is recorded directly from the PAMONO sensor without any viruses in the system resulting in empty background images. The real sensor noise does not only contain noise from the camera and the sensor but also other kind of distortions, e.g., caused by vibration or air bubbles in the system and is therefore more challenging.

Same as in $\textsc{Exp1}_{\textsc{Fuzzy}}$, the fuzzy method from Section 5.4.3 are part of the evaluation. The detection with the fuzzy method is compared to a detection that does not use the fuzzy method. However, here, the focus is more on the signal strength.

To measure the quality of the detection, the F_1 score, cf. Definition 37, is used. The F_1 score has the advantage that it penalizes both FP and FN. This is especially useful if a virus concentration should be measured.

7.3.3 Results

In the following the results for the two experiments $\textsc{Exp1}_{\textsc{Fuzzy}}$ and $\textsc{Exp2}_{\textsc{SignalStrength}}$ are presented.

Figure 7.4 shows the results for $\textsc{Exp1}_{\textsc{Fuzzy}}$. Results for the different optimization strategies compared to the manual baseline measurement are shown for the three evaluated data sets. With *manual* the baseline measurement is given, with *fuzzy* the optimization of the parameters for the fuzzy method, with *detection* the optimization of the detection parameters, and with *global* the global optimization.

For the first data set, $\textsc{Ds1}_{\text{HIV-VLPs}}$, with 100 nm HIV VLPs the accuracy could be improved from 45 % for the manually tuned parameters to 48 % for optimization of the

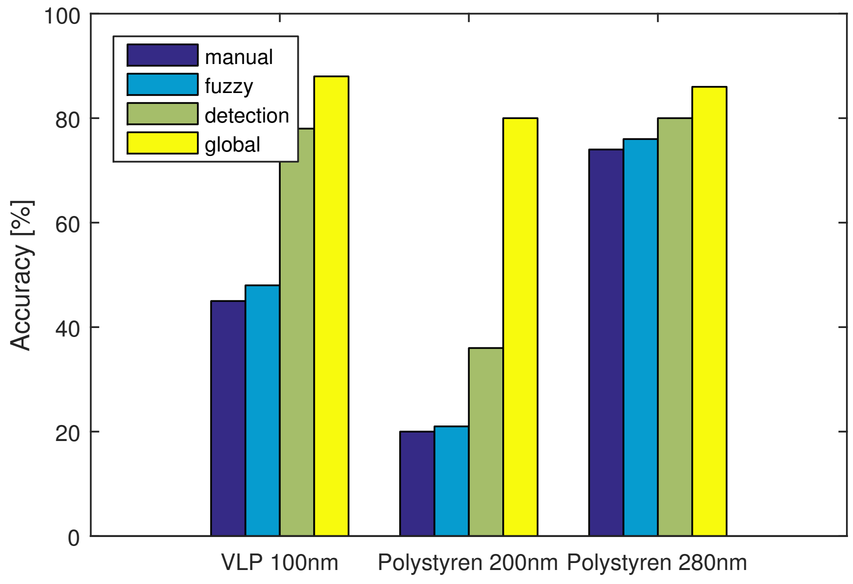


Figure 7.4: Detection quality for different single-objective optimizations. Three different particles are inspected. The optimization was done manually (manual), for the fuzzy filter (fuzzy), for the detection filters (detection), and finally globally (global). Each step uses the results from the previous one. Adapted from [LWT12].

fuzzy parameters, to 78 % for optimization of the detection parameters, and finally to 88 % for the global optimization of fuzzy and detections parameters.

For the second data set, $\text{Ds8}_{\text{Poly200nm-2}}$, with 200 nm polystyrene particles the accuracy could be improved from 20 % for the manually tuned parameters to 21 % for optimization of the fuzzy parameters, to 36 % for optimization of the detection parameters, and finally to 80 % for the global optimization.

For the third data set, $\text{Ds9}_{\text{Poly280nm}}$, with 280 nm polystyrene particles the accuracy could be improved from 74 % to 86 % for the global optimization.

Table 7.1 shows results for $\text{Exp2}_{\text{SignalStrength}}$ measured with F_1 score. For all cases the use of the fuzzy method results in an improved detection quality. Also, the signal strength influences the detection quality.

For the data sets with Poisson noise, the F_1 score could be improved from 0.995 to 1.000 for signal strength of 1.00, from 0.975 to 0.979 for signal strength of 0.75, and from 0.761 to 0.828 for signal strength of 0.50.

For the data sets with real sensor noise, the F_1 score could be improved from 0.988 to 1.000 for signal strength of 2.00, from 0.917 to 0.970 for signal strength of 1.00, from 0.917 to 0.970 for signal strength of 0.75, and from 0.518 to 0.593 for signal strength of 0.50.

7.3.4 Discussion

In the first experiment, $\text{Exp1}_{\text{Fuzzy}}$, it could be shown that in all cases the automatic optimization with SOG-PSE could obtain a better detection quality. The accuracy could be improved by 12 to 60 percentage points. Additionally, it could also be shown that the fuzzy method led to an accuracy of at least 80 %, which is a good result. It should be noted

Table 7.1: Single-objective optimization for different sources of noise and attenuation or gain factors of the virus signal. F_1 score is shown for fuzzy and non-fuzzy detection. Extended version of [LST+13a].

Noise source	Signal strength	F_1 Score for non-fuzzy	F_1 Score for fuzzy
Poisson	1.00	0.995	1.000
Poisson	0.75	0.975	0.979
Poisson	0.50	0.761	0.828
Sensor	2.00	0.988	1.000
Sensor	1.00	0.917	0.970
Sensor	0.75	0.801	0.873
Sensor	0.50	0.518	0.593

that this evaluation has been performed on a relatively early version of VirusDetectionCL with a simple pipeline configuration. Due to several improvements, e.g., by the calculation of additional features like the Gauss-Hesse feature, the full VirusDetectionCL method exceed these results, cf. Section 5.9.

As expected, the sequential optimization led to worse results than the global optimization of fuzzy and detection parameters combined. However, a surprising result is how much better the global optimization works compared to the sequential optimization. In all three data sets of $\text{Exp1}_{\text{Fuzzy}}$, the global optimization could improve the sequential optimization by 6 to 44 percentage points. This clearly shows that a global optimization of fuzzy and detection parameters is needed.

This can be especially seen for the bad detection results for the sequential optimization for the data set $\text{Ds8}_{\text{Poly200nm-2}}$ of only 36 % compared to the global optimization with 80 % accuracy. Here, the fuzzy parameters have been optimized according to the given manual parameters that only achieved an accuracy of 20 % and therefore was a bad starting point. The optimization of the fuzzy parameters led only to an accuracy of 21 %. Afterward, the optimization of the detection parameters could not compensate the optimization of the fuzzy parameters to achieve a good detection quality.

The need of a global optimization is somehow disappointing as the parameter space for a global optimization can be much larger compared to the combined size of the parameter space of sequential optimizations because for a global optimization the parameter spaces are combined multiplicative instead of additive.

For the second experiment, $\text{Exp2}_{\text{SignalStrength}}$, the influence of the noise type and the signal strength to the detection quality could be shown. The use of the fuzzy method is beneficial for all data sets. The improvement of detection quality is smaller for the data sets with Poisson noise than for the data sets with real sensor noise. The average improvement with the fuzzy method amounts to 2.5 percentage points. This is because the SNR is better in the data sets with Poisson noise so that the fuzzy method produces only a slight improvement.

For the data sets with sensor noise, the average improvement with the fuzzy method amounts to 5 percentage points, which is a very good result. For the low SNR in the data set with a signal strength of 0.5, an improvement of 7.5 percentage points could be achieved with the fuzzy method. This answers the research question if the fuzzy method is more advantageous for lower signal strengths.

Regarding the research question of how the signal strength affects the detection quality a quick falloff of detection quality could be observed if the signal strengths decreases. This is in accordance with the signal model in Section 5.3.1. For the examined version of VirusDetectionCL, a particle size of 150 nm shows a good detection quality while a particle size of 100 nm still has a reasonable detection quality. Based on these observations VirusDetectionCL has been updated in many details to successfully improve the detection quality for smaller particle sizes.

Regarding the research issue of how the real sensor noise compares to the Poisson noise, it could be shown that for Poisson noise the signal strength can be lower to achieve the same detection quality. For example, an F_1 score of 0.970 could be achieved for the real sensor noise for a signal strength of 1.0 while an F_1 score of 0.979 could be achieved for the Poisson sensor noise for a signal strength of 0.75. This corresponds to an approximate particle size of 150 nm for the Poisson noise and 200 nm for the real sensor noise. Similar results could be observed for the other signal strengths. Consequently, if the real sensor noise can be reduced to be more similar to Poisson noise, smaller particles can be detected with the same quality. This can either be achieved with an improved physical setup of the sensor or with a better noise reduction in VirusDetectionCL.

It could also be observed that the quality of the segmentation seems to be better if the fuzzy method is used. The noise corrupted boundaries of a virus signal could be covered more precisely. At the same time the number of FP does not increase. However, as no gold standard is available for the segmentation, this is only a subjective impression. Such an improved segmentation can be beneficial if the size of the viruses should be determined not only by the signal strength but also by the shape.

Finally, it could be observed that with the fuzzy method, lower detection thresholds can be used. This is a good indication that the detection is more easy with the fuzzy method enabled. The same could be observed for a better noise reduction.

7.3.5 Summary and Conclusion

In this section the SOG-PSE approach has been presented. With SOG-PSE an automatic parameter optimization can be performed regarding a single objective. For a given CPU or GPU program, the best parameter configuration can be obtained automatically. For this PSE the focus has been on a concrete application of the general methods that have been introduced in Section 7.2. This included the architectural design, the fitness evaluation, and the calculation of the fitness values.

VirusDetectionCL has been used for the evaluation. For this use case several parameters need to be optimized to obtain a good detection of viruses in PAMONO sensor data. A manual tuning of parameters is difficult as several dependencies exist between the parameters. It could be shown that with SOG-PSE a better detection quality could be achieved for VirusDetectionCL as with a manual optimization.

All research issues for SOG-PSE could be answered. Results from the first experiment strongly indicate that a global optimization is needed for VirusDetectionCL. The global optimization always resulted in a better detection quality than an optimization of the pipeline steps one after the other. Results from the second experiment showed how the signal strength affects the detection quality and parameters could be optimized toward different types of noise. In addition, the influence of the noise to the detection quality could be demonstrated. It could be shown that the detection quality on data sets with synthetically generated Poisson noise is better than on data sets with real sensor noise.

Overall, SOG-PSE is an important starting point for the DSE methods in the following sections and chapters. Several valuable insights have been gained from SOG-PSE. In the following SOG-PSE is extended toward a DSE for GPU hardware in Section 7.5. Section 7.4 with a manually performed DSE is bridging toward this. In Chapter 8 ideas from SOG-PSE are used to build a hybrid approach for a combined PSE and DSE.

7.4 Manually Performed DSE of Actual CPUs and GPUs

In this section a manually performed DSE on mobile and desktop CPUs and GPUs is presented. This is a starting point for the automatic DSE method in Section 7.5. The motivation behind this manual DSE is to find out if an energy-aware DSE on actual hardware produces useful results. As use case the performance of a laptop equipped with a mobile CPU and GPU is compared to the performance of a desktop system equipped with a desktop CPU and GPU.

The research issues that should be answered are:

- How can a DSE on actual hardware be performed for a given application?
- How can energy consumption be integrated into a DSE?
- How does the performance of a laptop system compares to that of a desktop system?
- What is the speedup of using GPUs instead of CPUs?
- How does energy consumption of CPU processing compares to that of GPU processing?
- How does the VirusDetectionCL approach perform in practice?
- Is the use of mobile GPUs reasonable with respect to the soft real-time constraint?
- How does the performance scale for different image sizes?

Manually performed DSE in this section provides the necessary basis for SOG-DSE in Section 7.5 and finally leads to the hybrid PSE and DSE approach MOGEA-DSE in Chapter 8. In addition, this chapter can be seen as an example of how application designers usually chose a system without help of automatic DSE methods.

This section is based on the author's publication [LST+13a].

7.4.1 Methods

For a manual DSE the most important challenge is the setup and conduction of the measurements. To investigate the performance and scalability of the VirusDetectionCL approach on CPU and GPU systems, different image sizes need to be considered. However, to achieve a good comparability of different image sizes some effort has to be spent. Simply taking a PAMONO sensor data set and using crops of different sizes out of this data set to measure the execution time or energy consumption is not a good solution. If a set of bigger images is simply subdivided into smaller images, not all parts of the image have the same amount of viruses appearing, and not all viruses within the sensor image have the same amount of defocus. As a result, smaller images would either have more or fewer

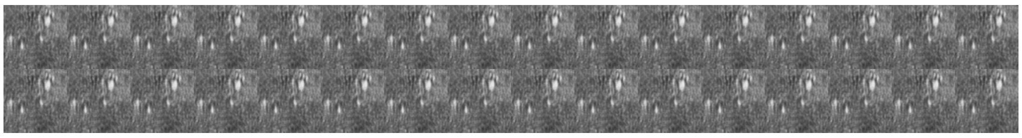


Figure 7.5: Synthetically designed PAMONO sensor image of size 1024 px × 128 px with 16 × 2 identical sub-images of size 64 px × 64 px.

viruses per area in perfect focus, which would influence the number of detected viruses per area and which ultimately results in differing execution time or energy consumption.

To solve this problem the number of viruses per area and the focus characteristic has to be identical in each of the data sets. Therefore, a small crop of, e.g., 64 px × 64 px with a well defined number of viruses appearing is used to construct bigger images by simply repeating the small image along the x- and y-axis for each frame. To keep the characteristic of the used data set, the appearing viruses in the sub-image should be fully covered, be in good focus, and should represent a typical virus concentration. Using a small image to construct bigger images has the advantage that the amount of viruses and the focus within each region is equal. The number of detected viruses is proportional to the image size which gives accurate results and makes the measurements comparable. The only variable that is changed is the image size. Figure 7.5 shows an example image of size 1024 px × 128 px that has been constructed with 16 × 2 identical sub-images of size 64 px × 64 px.

One target system for the mobile use of the PAMONO sensor and VirusDetectionCL is a laptop. To measure the energy consumption of the laptop, the battery capacity information is logged with the SYSFS file system before and after each run of VirusDetectionCL. The difference of capacity in Ampere hours can be used to calculate the energy consumption E_{Laptop} in Joule by multiplying the Voltage of the system with the difference of the capacity as

$$E_{\text{Laptop}} = V \cdot (C_{\text{before}} - C_{\text{after}}), \tag{7.2}$$

with C_{before} as the capacity right before the measurement, C_{after} as the capacity right after the measurement, and V as the voltage of the system.

7.4.2 Experiments

To answer the research issues raised in this section, the experiment Exp3$_{\text{Manual}}$ has been conducted. For this experiment a mobile system is compared to two desktop systems. On the mobile system, energy and execution time are measured. On the desktop system only the execution time is measured because energy consumption is not important on these systems. Different image sizes are considered. As already shown in Chapter 5, the VirusDetectionCL approach has a complexity of $\mathcal{O}(n)$ in Bachmann-Landau notation, with n as the image size. This experiment is conducted to show the performance and scalability of VirusDetectionCL in practice.

As mobile system, a Lenovo™T510 laptop is used. The laptop is equipped with an Intel®Core™i7-620M CPU, 4 GB RAM, an NVIDIA®Quadro NVS 3100M with 256 MB RAM, and a battery with a capacity of 8.4 A h at 11.1 V. The NVS 3100M GPU is a low cost graphics card with only 16 SPs, a core clock of 600 MHz, and a thermal design power

Table 7.2: Frame rate measurements for different CPUs and GPUs and for different image sizes. A soft real-time limit of 30 fps should be met. The second column indicates if the visualization was enabled. Extended version of [LST+13a].

Device	Vis.	1024 × 128	1024 × 256	1024 × 512	1024 × 1024
INTEL®i7-620M CPU	yes	16.0 fps	8.5 fps	4.6 fps	2.4 fps
INTEL®i7-2600 CPU	no	23.0 fps	20.1 fps	16.1 fps	11.5 fps
NVIDIA®3100M GPU	yes	53.7 fps[1]	32.6 fps	16.0 fps	7.8 fps
NVIDIA®GTX 480 GPU	yes	53.7 fps[1]	53.7 fps[1]	53.7 fps[1]	40.3 fps
NVIDIA®GTX 560 Ti GPU	yes	53.7 fps[1]	53.7 fps[1]	53.7 fps[1]	26.8 fps
NVIDIA®GTX 480 GPU	no	161.0 fps[2]	80.5 fps	80.5 fps	80.5 fps
NVIDIA®GTX 560 Ti GPU	no	161.0 fps[2]	161.0 fps[2]	161.0 fps[2]	80.5 fps

[1] Bounded by the synchronization to the monitor refresh rate of 60 fps.
[2] Bounded by the speed of loading and decoding the images.

of 14 W. The i7-620M CPU has two cores with a core clock of 2.66 GHz, a turbo clock rate of 3.333 GHz, and a thermal design power of 35 W.

For comparison purposes, two desktop systems are used. The first one with an INTEL®Q9550 CPU, 8 GB RAM, and a GTX 480 GPU. The second one with an INTEL®i7-2600 CPU, 16 GB RAM, and a more powerful GTX 560 Ti GPU. The GTX 480 has 480 SPs and is clocked at 700 MHz. The GTX 560 Ti has 384 SPs and is clocked at 822 MHz.

As data sets 160 images of 1024 px in width and 128 px, 256 px, 512 px, and 1024 px in height have been generated as explained in Section 7.4.1. Each data set is generated from 160 64 px × 64 px sub-images from DS7$_{\text{POLY200NM-1}}$, cf. Section 2.2. With the fixed width and variable height, the image size is only doubled not squared within each step. Reducing or increasing the height is also the best way to adapt the image size of the PAMONO images in practice as always the full width should be used to keep as many viruses in focus as possible. In contrast, if the image size is reduced in width, viruses perfectly in focus are left out and more out of focus viruses are kept.

7.4.3 Results and Discussion

The results of the experiment EXP3$_{\text{MANUAL}}$ are shown in Table 7.2. Achieved frame rates are given for the different devices on the different image sizes. The first two columns show results for the mobile CPU and the fastest desktop CPU. The third row shows the mobile GPU and the last four rows show the desktop GPUs. As the visualization can influence the measurements, the second column indicates if the visualization was enabled or not. As the camera in the PAMONO setup is usually operated at 30 fps, this is considered as the soft real-time limit the VirusDetectionCL software should meet.

A somehow surprising result is that both CPUs, mobile and desktop, are not able to achieve the necessary frame rate of 30 fps. Even for the smallest image size of 1024 px × 128 px only frame rates of 16 fps and 23 fps could be obtained. This image size is only half of the size needed in practice.

To further inspect why the CPUs are not able to meet the soft real-time limit, the kernel run times of the INTEL®CORE™i7-620M CPU has been measured individually. Exemplary, the 1024 px × 256 px data set has been inspected. The total execution time per frame is 118 ms, resulting in a frame rate of 8.5 fps. It has been found that the marching squares algorithm, explained in Section 5.5, is one of the limiting factors. The marching squares execution time varied from 9 ms to 239 ms with an average execution time of 30 ms. However, not one single factor could be identified, the CPU is slower than the GPUs for every single kernel. Simply put, a CPU is not the best architecture for fast image processing.

By using the mobile NVS 3100M GPU, the soft real-time limit could be achieved for image sizes up to 1024 px × 256 px. This is also surprising, as this GPU has only 16 SPs and is therefore not very powerful. For the larger images a slightly more powerful GPU would be needed as for these the 3100M only achieved 16 fps and 7.8 fps. For example, a NVS 4200M with 48 SPs or a NVS 5200M with 96 SPs should be sufficient for the larger images.

The results for the mobile GPU also demonstrate that the VirusDetectionCL algorithm scale almost linear. Images with half of the size can be processed with a doubled frame rate. Only for the smallest image size the frame rate of 53.7 fps is not doubled in relation to the 32.6 fps of the 1024 px × 256 px images. However, this is due to the visualization in VirusDetectionCL with enabled synchronization to the monitor. This bounds the frame rate by the monitor refresh rate of 60 fps. The synchronization to the monitor has been disabled in a later version of VirusDetectionCL but still present in these experiments.

For the desktop GPUs GTX 480 and GTX 560 Ti, the soft real-time limit could be achieved in all but one setup. Only the GTX 560 Ti is not able to process the biggest image size in more than 30 fps. This might be because the 560 Ti has less memory bandwidth than the GTX 480. For all other combinations of desktop GPU and image size, the 30 fps could be exceeded easily. The entries with 53.7 fps are also bound by the vertical synchronization in the visualization. Therefore, two more experiments without visualization are performed. The last two rows in Table 7.2 show results for the desktop GPUs without visualization. However, the visualization with synchronization to the monitor refresh rate is not the only limiting factor. The desktop GPUs also exceeds the performance of the CPU image decoding pipeline, which is used to load and decode the images for the analysis. As a result, the maximum frame rate that could be measured in the experiments is 161 fps. In the table the values are labeled accordingly, indicating if the frame rate is limited. If a camera is used, the loading from the hard drive and the decoding are not needed. The frame rates of 80.5 fps are also limited by some factor, this has not been inspected any further as such fast processing is currently not needed.

The results for the 1024 px × 1024 px images with desktop GPUs and without visualization show that both desktop GPUs are more powerful than needed for the VirusDetectionCL pipeline processing PAMONO sensor images within the soft real-time limit: both GPUs reach more than 80 fps. Usually, the frame rate is limited by the camera. If however recorded images are processed, a higher frame rate is useful, as experiments can be performed faster.

For the LENOVO™T510 laptop, a comparison of the energy consumption of GPU processing to CPU processing is shown in Figure 7.6. The energy consumption was measured for the processing of 160 frames of 1024 px × 128 px, 1024 px × 256 px, and

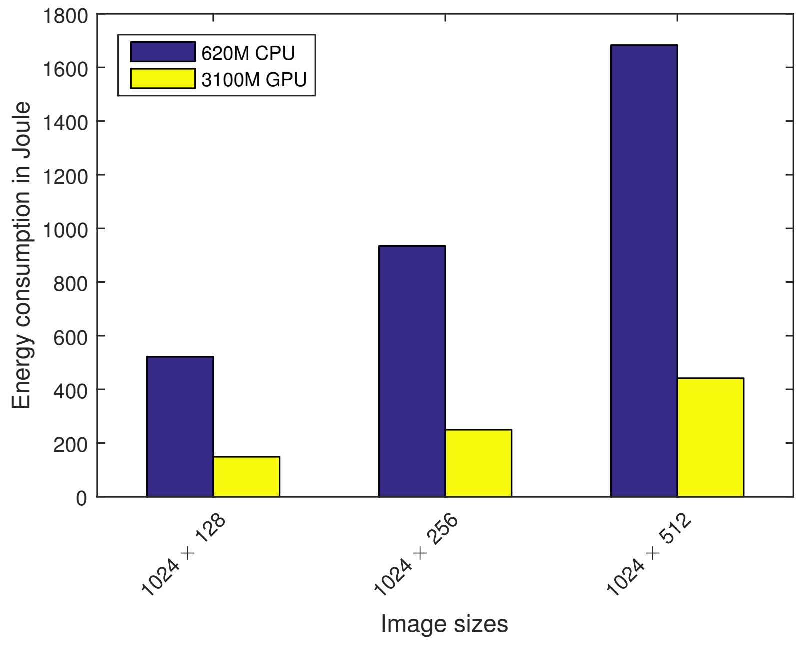


Figure 7.6: Energy consumption of a laptop system for the processing of 160 PAMONO images of different sizes. Processing on a mobile INTEL®i7-620M CPU compared to processing on a mobile NVIDIA®3100M GPU. Based on data from [LST+13a].

1024 px × 512 px sized PAMONO images on the 620M CPU and on the 3100M GPU. With help of the GPU, the energy consumption could be reduced by a factor of 3.74: with processing on the CPU the whole system consumes 934.4 J and with processing on the GPU it consumes 249.6 J. Similar results were obtained for 1024 px × 128 px images and 1024 px × 512 px images.

The energy consumption, while processing the PAMONO sensor images on the laptop, is 50 W h in average. With the battery capacity of 8.4 A h and the operating voltage of 11.1 V, this leads to two hours continuous processing depending on internal and external conditions. As a result, 24 to 60 samples can be processed without recharging for a typical length of two to five minutes for one data set.

7.4.4 Summary and Conclusion

In this section a manually performed DSE has been performed as preliminary experiments toward an automatic DSE as in Section 7.5. It could be shown that with a fixed parameterization and a limited number of target platforms a manual DSE can be sufficient to identify suitable hardware for a given software.

All raised research issues could be answered. The energy consumption could be integrated in the DSE by evaluating the software several times on the target laptop running on battery and logging the battery status with the LINUX®SYSFS interface. These measurements show that processing of the PAMONO images on the GPU is not only faster than a processing on the CPU but also more energy efficient.

A surprising result is that the NVIDIA®3100M GPU with only 16 SPs is sufficient to meet the soft real-time constraints of the VirusDetectionCL software. However, it could also be shown that both inspected CPUs were not sufficient to meet this constraint. At least for this experiment the processing on the GPU is not only faster but also more energy efficient for the processing of PAMONO images.

To conclude, with this manually performed DSE provided several valuable insights. The energy consumption and execution time on a laptop system can be reduced at the same time if the GPU is used for processing. A mobile GPU is suitable to enable a fast processing of PAMONO data with VirusDetectionCL. And CPUs are not useful to process such image processing tasks. These insights motivate the automatic DSE method in the following section.

7.5 Single-Objective Energy-Aware Hardware Design Space Exploration with SOG-DSE

In this section a single-objective energy-aware DSE for GPU architectures is presented. The method is called Single-Objective GPGPU Design Space Exploration (SOG-DSE). The approach is based on SOG-PSE in Section 7.3.

The number of transistors in GPUs is increasing from generation to generation. The NVIDIA®GeForce™GTX Titan [NVI15d] for example has 7.1 billion transistors, twice as many as an NVIDIA®GeForce™GTX 680. Currently still the Moore's law [Moo98] holds. The number of transistors is raising exponentially. Numerous challenges arise from this ongoing trend.

With SOG-DSE it is possible to explore existing and not yet existing GPU hardware under practical conditions in order to identify suitable GPUs for given programs. GPUs or entire GPU architectures can be simulated using a parametric GPU model. The advantage of simulating GPUs is that no actual hardware needs to be bought and measured. Extensive testing of different hardware is expensive and time-consuming and sometimes not even possible, e.g., if the desired GPU architecture is not yet available on the market.

The output of SOG-DSE is a Pareto front, cf. Definition 51, of best suited GPUs for a chosen objective. Available objectives are power consumption, energy consumption, number of core cycles, and execution time. Overall, it is possible to identify GPUs that fulfill the requirements of a system without the need to switch hardware and before the GPUs are bought or even exist.

If best suited GPU hardware should be identified for a given GPGPU program the execution time is often the only objective that is considered. However, taking power or energy consumption into account is becoming increasingly important [RSR+08; MWI+12; LHE+13]. For example, Leng et al. are arguing that to explore new GPU architectures, tools are needed to optimize the energy consumption of GPUs [LHE+13]. With the manual DSE in the previous section it could be shown that the use of GPUs can be more energy efficient than the use of CPUs only. This has also been shown by Rofouei et al. [RSR+08].

The following research issues are considered in this section:

- How can suited GPU hardware for a given GPGPU program be automatically explored?
- How can GPU architectures be explored efficiently?

The structure of this section is as follows. First, an introduction to GPGPU-Sim is given in Section 7.5.1. The extensions made on GPGPU-Sim are presented in Section 7.5.3. The power model GPUWattch, which can be used in conjunction with GPGPU-Sim, is explained in the Section 7.5.2. Afterward, the methods for SOG-DSE are presented in Section 7.5.3, followed by the description of the experiments in Section 7.5.4. Finally, the section concludes with results, discussion, summary, and conclusion in Section 7.5.5 and Section 7.5.6.

This section is based on the author's publications [LSW14; LMS+14].

7.5.1 GPGPU-Sim Simulator

For SOG-DSE, simulation of GPGPU task should be made possible. Therefore, a GPU simulator is integrated into the DSE method. As suitable simulator, the state-of-the-art GPU simulator GPGPU-Sim [BYF+09] has been chosen. GPGPU-Sim is a simulator for NVIDIA®GPUs, developed by Bakhoda et al. at the University of British Columbia. GPGPU-Sim is a cycle accurate simulator, which simulates Parallel Thread eXecution (PTX) and Source and ASSembly (SASS) GPGPU code.

The input of GPGPU-Sim is PTX or SASS code of the GPU program and an actual execution of the host code of this GPU program. Because for both OpenCL and CUDA the high level code is translated to PTX and SASS code by the NVIDIA®driver, GPGPU-Sim can run OpenCL code as well as CUDA code-based on the PTX or SASS code.

The output is the total GPU simulation cycles $\widetilde{C}_{\text{GPU}}$ and the number of operations, memory reads, memory writes, and memory row buffer activations. Additionally, the output contains activation factors α_i with $i \in \{1, \dots, N\}$ for each of the $NN \in \mathbb{N}$ GPU components. With the additional outputs the power and energy consumption of the GPU can be estimated with an additional GPU power model.

In the following a basic introduction to GPGPU-Sim should be given. As the CUDA architecture and the OpenCL architecture model are very similar, only the already introduced terminology and architecture of OpenCL from Section 4.1 is used in this section instead of a mixed OpenCL and CUDA terminology. To briefly link the most important terms, a grid is the same as the global index space, a cooperative thread array is the same as a work group, and a thread is the same as a work item that is currently executed.

To repeat the GPU architecture, generally a GPU consists of SMPs and the global memory, each SMP contains SPs and local memory. Several work items can be executed on the SPs. The work items are grouped in a work group.

Broadly speaking, in GPGPU-Sim an SMP is modeled with an SIMD pipeline that contains the SPs, a warp scheduler that selects the warps of the work group which should be executed on the SIMD pipeline, and a set of miss-status holding registers. The global memory is modeled with several memory modules of DRAM memory. Each memory module is driven by a memory controller. All memory controllers are connected via an interconnection network to the SMPs. [BYF+09]

In more detail, an SMP contains an SIMD pipeline. A work group is scheduled to an SMP. The work group is divided into warps. A warp is then scheduled to the SIMD pipeline. The work items in the warp cooperate and execute the same instruction. The pipeline is a 24 stage pipeline and consists of six logical stages: *fetch*, *decode*, *execute*, *memory1*, *memory2*, and *writeback*. The stages *fetch*, *decode*, and *writeback* are the same for all SPs because of the SIMD paradigm: all work items execute the same instruction but on different data. The width of the pipeline depends on the architecture. For example, the width is 8 for some NVIDIA®GPUs. All threads from one warp are processed with groups of the size of the width of the pipeline one group after the other until the warp is complete. Then the next warp is processed. [BYF+09]

The local memory is modeled as on chip memory that is shared by a work group. This memory is, e.g., 16 kB in size, has a low latency and is structured in memory banks, cf. Section 4.1. The global memory is modeled with several memory modules of DRAM memory with a cache for global texture memory and global constant memory on every SMP. [BYF+09]

To access global memory, constant memory, and per-thread off chip memory that is currently not cached, each SMP contains so called miss-status holding registers. With these registers many outstanding memory requests can be handled. Such a hardware support for cache misses is essential for a massive parallel hardware like GPUs. For GPUs a far larger number of outstanding memory requests need to be handled than for CPUs. [BYF+09]

It should be noted that, for NVIDIA®GPUs, the per-thread off chip memory is equal to global memory that is only used by one thread. This memory is called local memory in the CUDA terminology and is used, e.g., if there are not enough registers to hold the private variables of the threads. This behavior is called register spilling and should be avoided. [NVI09b]

For memory coalescing, the combined access of several threads to the global memory as explained in Section 4.1.3, the memory accesses of up to a half warp of threads can be combined. This is modeled in GPGPU-Sim. An intra-warp memory coalescing is also modeled where memory accesses of up to a warp of threads can be combined. [BYF+09]

The interconnection network is modeled with memory controllers that are distributed over the chip. Each memory controller is connected to two chips of the Graphics Double Data Rate (GDDR) memory. Not every SP is connected directly to a memory controller. For some of them the routing needs to be done through another SP. A 6×6 layout is used, where 28 SPs are combined with 8 memory controllers. [BYF+09]

A loose round robin scheduling is used to select warps for the execution on the SMPs. If enough warps are issued to an SMP, the long latency of the global memory accesses can be hidden. [BYF+09]

To sum up, GPGPU-Sim is a cycle accurate simulator that can simulate PTX and SASS code. It uses a detailed GPU model that has been developed for NVIDIA®GPUs. The power model GPUWattch, which can be used in conjunction with GPGPU-Sim, is explained in the following.

7.5.2 GPUWattch Simulator

GPUWattch [LHE+13] by Leng et al. is a power model for GPUs. The input of GPU-Wattch is a utilization of the hardware elements in the GPU and a description of the GPU architecture. The output is then a detailed information about the power consumption of the GPU.

GPUWattch can be tightly coupled with GPGPU-Sim. In this approach GPGPU-Sim is used to determine utilizations of the different hardware elements of the GPU by simulating an actual workload. Based on these statistics GPUWATTCH can estimate the power consumption of GPUs.

To estimate the power consumption, GPUWATTCH integrates the two tools MULTICORE POWER, AREA, AND TIMING (McPAT) [SAS+09] and CACTI-D [TAM+08]. McPAT is used to model the power consumption of different circuits on the GPU. McPAT was developed by Sheng et al. at IBM®. Because some of the GPU components are considerably different from the components in CPUs, McPAT has been modified by Leng et al. [LHE+13] to be able to model all GPU components. In addition, CACTI-D is used to model the power consumption of the GPU memory. CACTI-D has been developed by Thoziyoor et al. and is based on CACHE ACCESS AND CYCLE TIME MODEL (CACTI).

In the following an overview of how the power consumption of an GPU is calculated in GPUWATTCH is given from coarse to fine. Overall, the estimated total power consumption $\widetilde{P}_{\text{GPU}}$ of the GPU is modeled as a sum of static and dynamic power consumption as [LHE+13]

$$\widetilde{P}_{\text{GPU}} = \widetilde{P}_{\text{static}} + \widetilde{P}_{\text{dynamic}}. \tag{7.3}$$

The estimated static power consumption $\widetilde{P}_{\text{static}}$ consists of [LHE+13]

$$\widetilde{P}_{\text{static}} = \widetilde{P}_{\text{proc\_leakage}} + \widetilde{P}_{\text{mem\_leakage}} + \widetilde{P}_{\text{VRM}} + \widetilde{P}_{\text{peripherals}}, \tag{7.4}$$

with $\widetilde{P}_{\text{proc\_leakage}}$ as the leakage power of the processor, $\widetilde{P}_{\text{mem\_leakage}}$ as the leakage power of the main memory, $\widetilde{P}_{\text{VRM}}$ as the power of the VOLTAGE REGULATOR MODULE (VRM), and $\widetilde{P}_{\text{peripherals}}$ as the circuit power of all peripherals.

The dynamic power consumption $\widetilde{P}_{\text{dynamic}}$ is calculated as [LHE+13]

$$\widetilde{P}_{\text{dynamic}} = \widetilde{P}_{\text{proc\_dynamic}} + \widetilde{P}_{\text{mem\_dynamic}}, \tag{7.5}$$

with $\widetilde{P}_{\text{proc\_dynamic}}$ as the dynamic power consumption of the processing elements and $\widetilde{P}_{\text{mem\_dynamic}}$ as the dynamic power consumption of the main memory.

The dynamic power consumption of the processing elements, $\widetilde{P}_{\text{proc\_dynamic}}$ is modeled with activation factors for each of the components [LHE+13]

$$\widetilde{P}_{\text{proc\_dynamic}} = \sum_{i=1}^{N} \alpha_i \cdot \widetilde{P}_i^{\text{max}}, \tag{7.6}$$

with α_i as an activation factor and $\widetilde{P}_i^{\text{max}}$ as the peak power consumption of the i-th processing component. The processing elements include the SMPs with the SPs, instruction caches, L1 caches, texture caches, caches for constant memory, the memory coalescing logic, the shared memory, shared memory banks, shared memory crossbar network, register files, operand collectors, operand collection crossbar network execution units, floating point units, and special function units. The activation factors $\alpha_i \in [0, 1]$ are calculated by GPGPU-SIM and the peak power consumption for the different components is calculated with McPAT.

The dynamic power consumption of the memory, $\widetilde{P}_{\text{mem\_dynamic}}$, is modeled using the counts $n_\circ$ and an energy approximation $\widetilde{E}_\circ$ for the pre-charges (charge), row buffer activations (rb_act), reads (read), and writes (write) of the GPU memory as [LHE+13]

$$\widetilde{P}_{\text{mem\_dynamic}} = \frac{n_{\text{charge}} \cdot \widetilde{E}_{\text{charge}} + n_{\text{rb\_act}} \cdot \widetilde{E}_{\text{rb\_act}} + n_{\text{read}} \cdot \widetilde{E}_{\text{read}} + n_{\text{write}} \cdot \widetilde{E}_{\text{write}}}{T_{\text{GPU}}}, \tag{7.7}$$

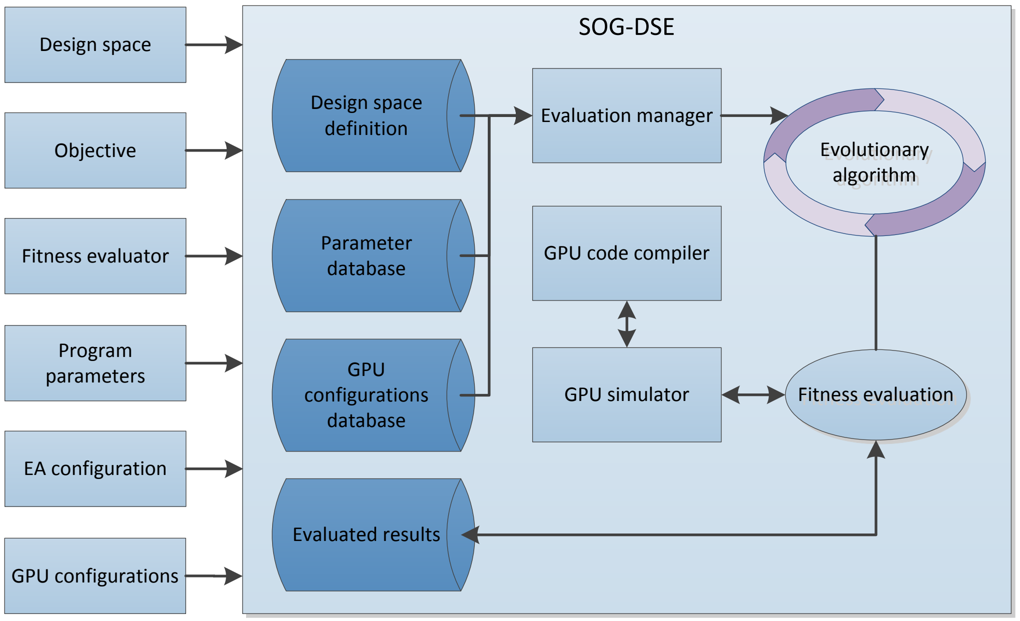


Figure 7.7: Schematic of the architectural design of SOG-DSE.

with T_{GPU} as the execution time.

The count of memory accesses are determined with GPGPU-Sim, same as for the activation factors. For the energy consumption of the GPU memory, the memory controller, the on-chip memory controller, and the interconnection network McPAT is used. Array structures are modeled with CACTI-D. For the remaining memory components an empirical model is used. For the overall remaining uncertainties, GPUWattch has been calibrated using micro benchmarks, which stress certain components of the GPU. Additionally, false assumptions can be compensated to a certain degree with this calibration. [LHE+13]

Finally, GPUWattch has been verified with measurements on actual hardware. The averaged modeling error for the inspected GPUs was 15.0 % and 16.2 % for the micro benchmarks and 9.9 % and 13.4 % for full benchmarks. [LHE+13]

7.5.3 Methods

In the following the methods for SOG-DSE are presented. As the SOG-DSE approach is based on the SOG-PSE approach in Section 7.3, the focus is on the differences between these methods. The main difference to SOG-DSE is that GPGPU-Sim and GPUWattch are integrated to enable a DSE of GPUs. Additionally, the handling of hardware related parameters and the calculation of hardware related fitness values is an important aspect of SOG-DSE.

First, the architectural design of SOG-DSE is shown. Then the integration and extension of GPGPU-Sim, the integration of GPUWattch, and the calculation of the fitness values is described. Followed by the modeling of dependencies in the design space. Finally, an extension of VirusDetectionCL is made to enable the access of hardware parameters.

Architectural Design of SOG-DSE

Figure 7.7 shows a schematic of the SOG-DSE approach. In contrast to the SOG-PSE, cf. Figure 7.3, a GPU simulator and a GPU compiler are added. The GPU configurations that should be optimized are provided as parameters and stored in the GPU configurations database. The evaluation manager provides the GPU configurations to the EA. Within the fitness evaluation the GPU simulator is used and the needed GPU code is compiled by the GPU compiler. The GPU compiler to compile OpenCL and CUDA code can be provided, e.g., by an installed GPU driver software of the host system.

Integration and Extension of GPGPU-Sim

To enable the simulation of GPUs for GPGPU processing, the GPGPU-Sim simulator, cf. Section 7.5.1, is integrated into SOG-DSE. GPGPU-Sim can simulate the execution of OpenCL and CUDA programs on different GPUs.

The integration of GPGPU-Sim into SOG-DSE works as follows. First, GPGPU-Sim is installed on the evaluation machine, which has only to be done once. Second, the GPGPU-Sim parameters are set up. Third, the OpenCL or CUDA code of the GPGPU program that should be evaluated is compiled to PTX code by an NVIDIA®driver. Fourth, the GPGPU-Sim library is linked to the GPGPU host program that should be run. Hence, each GPU related call is redirected to the GPGPU-Sim library instead of the GPU library and the GPU. Fifth, the GPGPU program is executed. Finally, the results from GPGPU-Sim are used to calculate the necessary fitness values.

In order to make this integration work, some extensions and modifications had to be done on GPGPU-Sim, which are listed in the following. First, the OpenCL Application Programming Interface (API) of GPGPU-Sim has been modified and extended. This includes the adding of a few missing OpenCL API calls, missing cases within the calls, and correction of wrong function signatures. Also, one API call was not fully implemented according to the OpenCL specifications. According to the specifications a work group size can be determined automatically by the OpenCL driver if it is not provided. One evaluated program did make use of this missing feature and therefore did not run. The feature has been added by automatically choosing the work group size in each direction as the maximum possible power of two that also divides global work size without remainder.

Second, GPGPU-Sim offers the remote compilation of OpenCL and CUDA code. This is especially useful if a compute server without GPUs is used to do the calculations. However, GPGPU-Sim does not offer to reuse the PTX code that has been compiled by a remote computer. The remote compilation uses a lot of time and SOG-DSE performs several runs of GPGPU-Sim on the same OpenCL program. Therefore, GPGPU-Sim has been extended to save and reuse the PTX code once it has been compiled. This speeds up the evaluation especially if several threads on the compute server use the same remote computer for compilation.

Third, as SOG-DSE can evaluate several hundred GPU configurations and not every configuration is guaranteed to be valid, a mechanism of detecting software freezes of GPGPU-Sim has been added. Every few minutes a file is written with the current status of GPGPU-Sim. This can be checked from outside of GPGPU-Sim and GPGPU-Sim can be canceled if a problem is detected. This keeps the overall evaluation running even if single runs fail. Each run that did not complete is logged with the current GPU configuration for further debugging.

Fourth, the output of GPGPU-Sim has been modified to provide a file output with all needed statistics and not only a system out text output.

Sixth, the output on the system out of GPGPU-Sim has been reduced to a minimum to speed up the remote evaluation.

Finally, during the extension of GPGPU-Sim also a few bugs were found in the code. Bug fixes, missing OpenCL API calls, missing cases within the API calls, correction of wrong API function signatures, and the missing automatic work group calculation have been provided to the authors of GPGPU-Sim.

Integration of GPUWattch

In order to make SOG-DSE energy-aware, GPUWattch, cf. Section 7.5.2, is integrated into SOG-DSE. GPUWattch calculates the estimated energy consumption based on an architecture model and a utilization of hardware elements. As GPGPU-Sim and GPUWattch are build to work as one unit, the integration of GPUWattch is relatively straight forward.

First, GPUWattch is installed on the evaluation machine, which has only to be done once. Second, GPGPU-Sim is configured to use GPUWattch. Third, the GPUWattch parameters are set up. Fourth, the GPGPU program is executed with GPGPU-Sim. The output from GPGPU-Sim is automatically used as input for GPUWattch and afterward the output of GPUWattch is calculated. Finally, the results from GPUWattch are used to calculate the necessary fitness values.

The only difficulty is to keep the parameters for GPGPU-Sim and GPUWattch consistent. Both programs use a different parameter representation and also slightly different names for the parameters. To solve this, the needed GPGPU-Sim parameters and the GPU configuration is automatically extracted from the databases by SOG-DSE, modified accordingly, and provided to GPUWattch.

Calculation of the Fitness Values

For the seamless integration of the results from GPGPU-Sim and GPUWattch, SOG-DSE automatically calculates several fitness values from the output of GPGPU-Sim and GPUWattch. GPGPU-Sim provides the needed cycles $\widetilde{C}_{\text{GPU}}$ for the evaluated GPU program and GPUWattch provides the estimated power consumption $\widetilde{P}_{\text{GPU}}$. From these two values the fitness values execution time, and energy consumption are calculated as described in the following.

The execution time $\widetilde{T}_{\text{GPU}}$ is calculated with the known shader clock rate S and the number of cycles $\widetilde{C}_{\text{GPU}}$:

$$\widetilde{T}_{\text{GPU}} = \frac{\widetilde{C}_{\text{GPU}}}{S}. \tag{7.8}$$

With the execution time $\widetilde{T}_{\text{GPU}}$, the energy consumption $\widetilde{E}_{\text{GPU}}$ can be calculated:

$$\widetilde{E}_{\text{GPU}} = \widetilde{T}_{\text{GPU}} \cdot \widetilde{T}_{\text{GPU}}. \tag{7.9}$$

Table 7.3: Evaluated FERMI™GPU architectures. For each architecture the number of SMPs, number of SPs, clock rates, and DRAM type are listed. Adapted from [LSW14; LMS+14].

Architecture	SMPs	SPs	Core clock	DRAM	DRAM clock
GF-108	1-2	48,96	700-810 MHz	GDDR-3	400-450 MHz
GF-106	3-4	114,192	590-790 MHz	GDDR-5	450-1000 MHz
GF-104	6-7	288,336	650-675 MHz	GDDR-5	850-950 MHz
GF-100	11-15	352,384,416,448,480	610-780 MHz	GDDR-5	800-1000 MHz

Modeling of Dependencies in the Design Space

One difficulty for the evaluation of different GPU architectures is that not all parameter combinations map to valid GPUs. If this is not considered in the evaluation, not existing GPUs can be identified as best suited with respect to the evaluated objectives. Thus, dependencies need to be modeled appropriately.

As an example, the four NVIDIA®FERMI™architectures [NVI09a] from Table 7.3 shall be considered. These are also used for the experiments in Section 7.5.4. Four different classes of FERMI™architectures are listed in the table, namely: GF-108, GF-106, GF-104, and GF-100. The different configurations from these architectures corresponds to nine actual NVIDIA®GPUs that are available in the market, namely: GEFORCE™GT 420, GEFORCE™GT 430, GEFORCE™GT 440, GEFORCE™GTS 450, GEFORCE™GTX 460, GEFORCE™GTX 460 SE, GEFORCE™GTX 465, GEFORCE™GTX 470, and GTX 480. The four architectures and which actual GPU uses which architecture is explained in the following. A special case is the GEFORCE™GT 440 since it is available both with the GF-108 and GF-106 architecture.

The GF-108 architecture has only one or two SMPs with 48 or 96 SPs in total but can be clocked at the highest core clock rate of up to 810 MHz. For the DRAM only GDDR-3 is available, which is also clocked with the lowest DRAM clock of 400 MHz to 450 MHz. The actual NVIDIA®GPUs in the market from this architecture are GEFORCE™GT 420, GEFORCE™GT 430, and GEFORCE™GT 440.

The GF-106 architecture has three or four SMPs with 114 or 192 SPs in total. For the DRAM it uses the faster GDDR-5. The GPUs from this architecture are GEFORCE™GT 440 and GEFORCE™GTS 450.

The GF-104 architecture has six or seven SMPs with 288 or 336 SPs. The GPUs from this architecture are GEFORCE™GTX 460 and GEFORCE™GTX 460 SE.

Finally, the GF-100 architecture, the most powerful of the four, contains 11 to 15 SMPs with up to 480 SPs in total. The core clock rate is in a range of 610 MHz to 780 MHz and the GDDR-5 DRAM is clocked in a range of 800 MHz to 1000 MHz. The GPUs from this architecture are GEFORCE™GTX 465, GEFORCE™GTX 470, and GTX 480.

It is not feasible to optimize for the complete parameter space that is spanned by the ranges 1 to 15 SMPs, 48 to 480 SPs, a core clock of 590 MHz to 810 MHz, GDDR-3 or GDDR-5 memory, and a 400 MHz to 1000 MHz DRAM clock. The useful parameter space is much smaller than this as several dependencies exist. As an example, for GF-108, GF-106, and GF-104 each SMP contains 48 SPs. For GF-100 each SMP contains 32 SPs. As a consequence, a GF-104 GPU with 6 SMPs and 192 SPs does not exist.

The solution for modeling of dependencies in the design space is to define two design spaces: a larger design space without dependencies and the actual design space for which all dependencies are considered. Solutions for the larger design are easy to generate. These solutions are then mapped to the actual design space. This is done by a nearest neighbor matching of new parameter sets to the nearest valid parameter set. In more detail, a k-d tree is build for the actual search space. For new parameter sets invalid configurations are allowed and not prevented. Each new parameter set is then matched to the nearest valid parameter set in the k-d tree. As distance measure, a normalized Euclidean distance is used.

One drawback of this method is that the tree size grows quickly for larger design spaces. Additionally, first generating not valid parameters and then matching to a valid parameter set also introduces overhead that can be prevented if no invalid parameter sets are generated in the first place.

Extension of VirusDetectionCL to Provide Hardware Parameters

VirusDetectionCL has been extended to enable access to hardware parameters of the OpenCL program. This includes the size of some memory buffers, access to hardware optimization options of the OpenCL compiler, and individual access to all work group sizes of every kernel.

The memory sizes and the parameters for the OpenCL compiler are simply made available as parameters whereas the work group sizes are more difficult to handle. These are usually provided as constants in a header file. For the host code these values can be variable but for the OpenCL code they are often required to be constant and cannot be changed at runtime. This is because often some array sizes depend on these values and the array sizes in OpenCL need to be constant and known during compilation. Therefore, it is not possible to change these values after compilation.

The solution to this is to change these values and afterward compile the GPU code again. While the device code can be compiled fast at runtime, a compiling of the host program takes longer and should be avoided. Therefore, the same header file with the work group sizes that is provided as device code is loaded and parsed dynamically at runtime by the host program. As a result, the usually fixed sizes for the work group sizes in the header file are made variable.

7.5.4 Experiments

For the SOG-DSE method the experiment $\text{Exp4}_{\text{Energy}}$ has been conducted. Within this experiment the most energy efficient GPU architecture and GPU configuration for different programs should be identified.

Different NVIDIA®Fermi™architectures are used as target architectures. Namely, the architectures GF-108, GF-106, GF-104, and GF-100 as shown in Table 7.3. Each of these four Fermi™architectures is available in different hardware configurations: The number of SMPs varies from 1 to 15 and the number of SPs in a range of 48 to 480. The core clock rate varies from 590 MHz to 810 MHz. The GF-108 GPUs use GDDR-3 memory with a

Table 7.4: Evaluated programs for the single-objective optimization from benchmark suits CUDA-SDK, GPGPU-Sim, and Rodinia. Adapted from [LMS+14].

Suite	Program	Language	Description
CUDA-SDK	quasiRandomGen	OpenCL	Random number generator
CUDA-SDK	matrixMul	OpenCL	Parallel matrix multiplication
CUDA-SDK	vectorAdd	OpenCL	Parallel vector addition
GPGPU-Sim	AES	CUDA	AES encryption algorithm
GPGPU-Sim	CP	CUDA	Calculation of the Coulomb potential physics simulation
GPGPU-Sim	LPS	CUDA	Laplace discretization on a 3D structured grid
GPGPU-Sim	RAY	CUDA	Raytracing simulation for rendering of light effects
GPGPU-Sim	STO	CUDA	MD5 hash calculation
Rodinia	backProp	OpenCL	Back propagation algorithm in neural networks
Rodinia	hotSpot	OpenCL	Processor temperature physics simulation
Rodinia	kmeans	OpenCL	K-Means clustering algorithm
Rodinia	gaussian	OpenCL	Gaussian elimination for solving systems of linear equations
Rodinia	nn	OpenCL	Nearest neighbor location calculation on hurricane data
Rodinia	nw	OpenCL	Needleman-Wunsch nonlinear global optimization on DNA sequences

clock rate of 400 MHz to 450 MHz and the other architectures use GDDR-5 memory with a clock rate of 450 MHz to 1000 MHz.

Programs from three different benchmark suites are used for the evaluation to show the generality of SOG-DSE: the CUDA Software Development Kit (SDK) [NVI15c] by NVIDIA®, GPGPU-Sim [BYF+09] from Bakhoda et al., and Rodinia [CBM+09] from Che et al. The benchmarks and the evaluated programs from the benchmarks are listed in Table 7.4 and are explained in the following.

The NVIDIA®CUDA SDK contains a set of CUDA and OpenCL benchmarks, for example, price prediction of options, a hidden Markov model, a random number generator, a sum reduction on large arrays, or a parallel matrix multiplication. The benchmark programs are highly optimized for GPUs. The GPGPU-Sim benchmarks provide a wide range of applications from different authors, for example, encryption, graph traversal, physics simulation, neural networks, or weather prediction. All benchmark programs from GPGPU-Sim are only available for CUDA devices. Rodinia consists of several parallel programs from a wide field of application and from different authors: image processing,

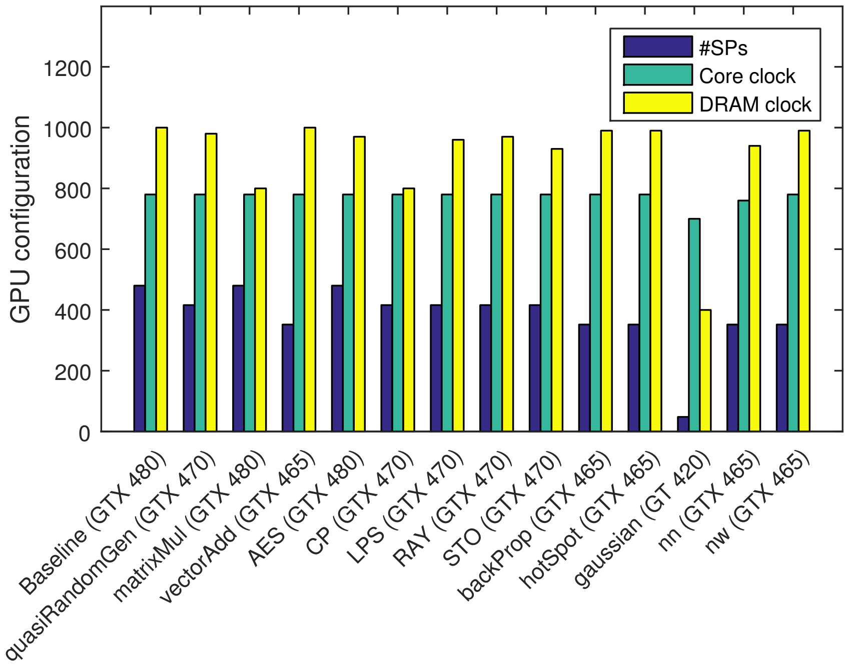


Figure 7.8: The most energy efficient GPU configurations for different programs (cf. Table 7.4). The name of the identified GPU is provided in brackets. As baseline the NVIDIA®GTX 480 with GF-100 architecture is shown. Evaluated for the architectures GF-108, GF-106, GF-104, and GF-100. Adapted from [LMS+14].

medical imaging, bioinformatics, physics, data mining math, pattern recognition, and others. Parallel implementations for CUDA, OpenCL, and Open Multi-Processing (OpenMP) are available.

Short descriptions of the programs are provided in Table 7.4. For example, the program hotSpot is a well known physics simulation of heat distribution on CPUs. The nw program uses a Needleman-Wunsch algorithm to calculate the similarity of two DNA sequences and finding the best alignment of these two sequences. gaussian refers to the Gaussian elimination algorithm, used for solving systems of linear equations. matrixMul is a highly optimized parallel matrix multiplication and CP is a physics simulation for the calculation of the Coulomb potential.

The evaluation was performed on a compute server with four AMD Opteron™6272 CPUs and 256 GB RAM with Debian™Linux®as host operating system. Each of the Opteron™CPUs has 16 cores, resulting in 64 cores in total. Because the GPUs were simulated, the compute server did not need to have GPU hardware installed. As guest operating system Linux®Ubuntu™12.10 for servers was used, which has been executed in a Kernel-based Virtual Machine (KVM) on the host.

7.5.5 Results and Discussion

The results for experiment Exp4$_{\text{Energy}}$, the single-objective evaluation of the benchmark programs, are shown as bar plot with the configurations of the single-objective evaluation in Figure 7.8. The first entry shows the baseline GPU configuration with a GTX 480 GPU with GF-100 architecture. Afterward, the most energy efficient GPU configurations for thirteen programs are shown. The programs quasiRandomGen to nw are listed and described in Table 7.4. Each shown configuration can be mapped unambiguously to the architectures with help of Table 7.3.

Overall, the GF-100 architecture with the GPUs GeForce™GTX 465, GeForce™GTX 470, and GTX 480 shows to be the best for all but one inspected program. The actual GPU and actual configuration of the GF-100 architecture varies for the programs. The number of SMPs for the GF-100 architectures is in a range from eleven to fifteen, the core clock in a range of 760 MHz to 780 MHz, and the DRAM clock in a range of 800 MHz to 1000 MHz.

The only program for which the GF-100 architecture has not been chosen is gaussian. For this program the GeForce™GT 420 GPU with GF-108 architecture was the most energy-efficient. This GPU has only one streaming multiprocessor, a low core clock of 700 MHz, and a low DRAM clock of 40 MHz. This indicates that the gaussian program cannot fully utilize GPUs with more SMPs at least for the considered input size.

A surprising result is, that the most powerful GPU, the GeForce™GTX 480, has only been chosen for matrixMul and AES. And for both the DRAM clock is reduced. As a result, actually no program uses the most powerful GPU configuration that was chosen as baseline. This also indicates that a proper utilization of the GPU plays an important role.

The remaining GPU architectures GF-106 and GF-104 have not been chosen for any of the programs. Except for the gaussian program, the more powerful GF-100 architecture is a better choice regarding energy consumption.

7.5.6 Summary and Conclusion

In this section the SOG-DSE method has been presented. With SOG-DSE a single-objective DSE of GPU architectures can be performed. For a given GPGPU program, the best suited GPU architecture can be obtained. The objective can be number of cycles, execution time, power consumption, or energy consumption of the inspected GPU on the given workload. The GPU architectures can be simulated cycle-accurately with GPGPU-Sim, and the power or energy consumption can be obtained with GPUWattch.

SOG-DSE has been evaluated on a variety of different programs from three different benchmarks. Exemplary, four different Fermi™GPU architectures have been inspected. The configurations of the four architectures result in eleven actual NVIDIA®GPUs. Additionally, for each of the four architectures, a frequency scaling of the core clock and DRAM clock has been inspected. As a result, the most energy efficient GPU configurations have been identified for the given GPGPU programs.

For most of the programs, the GPU configuration with the fastest core clock speed and highest number of SPs is also the most energy efficient. However, there is no general rule for this as not all programs could make use of the maximum number of SPs or the maximum clock speed. For one program a GPU with only very few SPs has been identified as most suited. Overall, it could be shown that the SOG-DSE method can be used to automatically identify well suited GPU architectures for given GPGPU programs.

In Section 7.6 the SOG-DSE method is extended toward a DSE of high performance GPU architectures. And in Chapter 8 ideas from SOG-PSE and SOG-DSE are combined to a hybrid PSE and DSE approach.

7.6 Single-Objective Power-Aware Hardware Design Space Exploration of GPUs for HPC Systems with SOG-DSE

In this section SOG-DSE is extended toward a DSE of GPU architectures of High-Performance Computing (HPC) systems. High performance GPUs can be explored and best suited GPU parameters like the core clock and DRAM clock can be obtained regarding one or more objectives.

If a new HPC system should be equipped with GPUs, it would be beneficial to know ahead, which type of GPU is the most power or energy efficient for the GPGPU tasks that need to be solved while, at the same time, the desired processing speed can be achieved. This knowledge is especially advantageous to design a system for green computing.

For HPC systems other objectives are of interest than for desktop systems. In addition to the performance and energy consumption, the Thermal Design Power (TDP) of an HPC system is of importance. Either the cooling system of an HPC system needs to be designed appropriately to the TDP or the GPU has to fulfill the constraint of a certain maximum TDP.

In both cases it would be beneficial if the energy and power consumption of a GPU is known in advance, especially if an HPC system is planned long ahead and the GPUs for the system are not yet available on the market.

The following research issues are considered:

- Which performance measures are needed for HPC systems?
- How can typical server loads be modeled?

This section is based on the author's publication [LSW14].

7.6.1 Motivation

In HPC systems more and more GPUs are integrated and used for calculations resulting in GPGPU on HPC systems becoming more and more important [HTW+10]. However, choosing suitable GPUs is a difficult task as several, possibly conflicting objectives and constraints need to be fulfilled.

In context of the PAMONO sensor use case HPC systems are important if the PAMONO sensor is used as a mobile, hand-held device outdoors or in a hospital or at airports and the images are offloaded to an HPC system for processing. Related scenarios are the cloud computing scenario $\text{Scn4}_{\text{Cloud}}$ and the hospital scenario $\text{Scn5}_{\text{Hospital}}$ from Figure 7.2. In both of these scenarios an HPC server is used to process PAMONO sensor images.

Especially for a mobile use, the energy consumption on a hand-held device should be low to enable several measurements without the need to recharge the battery. To achieve this offloading of work to an HPC system can be used in order to save energy on the device. For a mobile use in hospitals or at airports, the energy consumption is not as crucial as for a mobile outdoor use but with offloading, a fast processing speed can be achieved with smaller and less powerful hand-held devices. In both use cases several PAMONO devices can share one HPC server and the server can be used to not only process PAMONO sensor data but also other tasks.

In this section the server side of these scenarios is inspected and in Section 8.4 the client side. Here, the HPC systems are evaluated for general use cases while in Section 8.3 the actual PAMONO use case is evaluated.

7.6.2 Methods

To extend SOG-DSE toward a DSE of GPUs for HPC, two extensions are necessary. First, the introduction of new fitness values, which are especially suited for HPC. Second, modeling of different server loads to simulate not only a single program.

The architectural design is the same as explained in Section 7.5.3. The modeling of high performance GPUs is not part of this section but part of Section 8.3.

Calculation of the Fitness Values

In Section 7.5.3 it has been shown how the fitness values power consumption, energy consumption, cycles, and execution time are calculated. Additionally, for HPC systems also other objectives might be of interest. Therefore, five additional fitness values are introduced: peak power consumption, cycles to power ratio, power to cycles ratio, energy delay product, and power delay product.

For the TDP of a system, the peak power consumption $\widetilde{P}^{\max}$ and the average power consumption $\widetilde{P}_{\text{GPU}}$ are of interest. Both values can be received from GPUWattch. GPUWattch uses McPAT and CACTI-D to calculate the peak power consumption while the average power consumption is calculated with the activation factors calculated by GPGPU-Sim. The peak power consumption is especially useful to determine the TDP of HPC systems for GPUs that do not yet exist.

With the used cycles $\widetilde{C}_{\text{GPU}}$, provided by GPGPU-Sim, and the estimated power consumption $\widetilde{P}_{\text{GPU}}$, cf. Equation (7.3), the cycles to power consumption ratio fitness value can be calculated as

$$\text{CyclesToPowerRatio} = \frac{\widetilde{C}_{\text{GPU}}}{\widetilde{P}_{\text{GPU}}}. \tag{7.10}$$

Analogously, the power consumption to cycles ratio fitness value can be calculated as

$$\text{PowerToCycleRatio} = \frac{\widetilde{P}_{\text{GPU}}}{\widetilde{C}_{\text{GPU}}}. \tag{7.11}$$

Another useful fitness value for HPC is the energy delay product, which permits a weighting of energy consumption and execution time. The energy delay product is calculated as [SCR+13]

$$\text{EnergyDelayProduct} = (\widetilde{E}_{\text{GPU}})^{\alpha} \cdot (\widetilde{T}_{\text{GPU}})^{\beta}, \tag{7.12}$$

with $\alpha, \beta \in \mathbb{R}^+$ as weights.

In the same manner the power delay product can be calculated as

$$\text{PowerDelayProduct} = (\widetilde{P}_{\text{GPU}})^{\alpha} \cdot (\widetilde{T}_{\text{GPU}})^{\beta}, \tag{7.13}$$

with $\alpha, \beta \in \mathbb{R}^+$ as weights.

Modeling of Typical Server Loads

On an HPC system usually several applications are executed. Each application might have different demands for the GPU hardware. This should be reflected in a DSE. Therefore, SOG-DSE is extended such that it can model typical server loads by using the same approach as for the optimization concerning several data sets in Section 7.2.4.

For each program a weighting factor can be defined. For $P \in \mathbb{N}$ evaluated programs, the weighted overall fitness value $f \in \mathbb{R}$ for all programs can be calculated as the sum over all weighted fitness values for each individual program as

$$f = \sum_{p=1}^{P} w_p \cdot f_p, \tag{7.14}$$

with $w_p \in \mathbb{R}$ as a weighting factor for the p-th program and $f_p \in \mathbb{R}$ as the fitness value for the evaluation of the p-th program. If no weighting factor is defined w_p is simply chosen as $\frac{1}{P}$ for each p.

7.6.3 Experiments

In HPC servers often simply the fastest GPU is chosen that fits the budget. However, if the HPC system uses mainly the CPU for processing and the GPU is only used by a few programs, the TDP of the GPUs can be more important than the speed. The overall TDP of an HPC system influences how the power supply and the cooling of the system is set up. If GPUs are integrated in an existing HPC system, they have to stay within the overall TDP.

Therefore, the experiment $\text{Exp5}_{\text{Power}}$ has been conducted that compares the fastest GPU with power efficient GPUs. Specifically, the fastest GPU configuration of the Fermi™architecture, an NVIDIA®GTX 480 GPU, has been compared to all other GPUs with Fermi™architecture. The evaluated architectures are the GF-108, GF-106, GF-104, and GF-100 as listed in Table 7.3. These GPU architectures are explored regarding their power consumption.

An excerpt of four programs from the Rodinia benchmark is used: hotSpot, k-Means, nn, and nw. A description of the programs is provided in Table 7.4.

The evaluation has been performed on a PC with four Intel®Xeon E5-2690, each CPU with eight cores, 64 GB RAM and Windows™7 Operating System (OS). An Intel®Atom D510 with Ubuntu™OS has been used as remote compiler for the OpenCL code to provide the PTX code.

7.6.4 Results and Discussion

The optimized power consumptions for experiment $\text{Exp5}_{\text{Power}}$ are shown in Figure 7.9 and the corresponding optimized GPU configurations are shown in Figure 7.10. The optimized results are in comparison to the NVIDIA®GTX 480 GPU with GF-100 architecture.

For hotSpot the GeForce™GT 420 GPU with GF-108 architecture has been found to be best. The identified hardware configuration of this architecture uses 48 SPs, a core clock rate of 700 MHz and a DRAM clock rate of 450 MHz. With this configuration the power consumption has been reduced from 49.7 W to 11.5 W, which is a reduction by 77 %. For nn the same GT 420 GPU with a slightly different configuration has been identified.

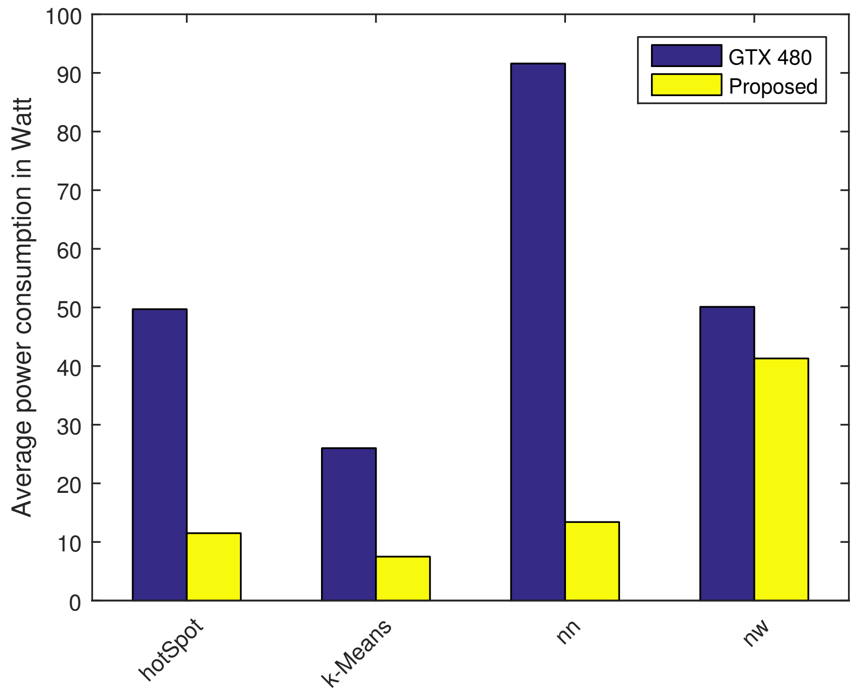


Figure 7.9: The average power consumption for four different programs (cf. Table 7.4) with the proposed method, compared to an NVIDIA®GTX 480 as baseline. Evaluated for the architectures GF-108, GF-106, GF-104, and GF-100. Lower values are better.

Only the DRAM clock rate is slightly lower. The power consumption for NN is reduced by 85 % for this configuration.

For the K-MEANS program a GEFORCE™GT 450 GPU with GF-106 architecture with a configuration with 115 SPs and low clock rates has been identified with a power reduction by 71 %.

Finally, for the NW program a much faster GTX 470 GPU with GF-100 architecture with 384 SPs has shown to be best but a configuration with low clock rates has been identified. As the configuration for NW is similar to the baseline, the power saving is only 18 %.

An interesting result can be found, by comparing the hardware configurations from this section with the configurations from Section 7.5.5. For the power-aware optimization, slower GPUs are identified in all cases, compared to the energy-aware optimization. The power-aware hardware configurations use less SPs and a reduced core clock and DRAM clock rate.

For example, for the HOTSPOT program a GT 420 GPU with GF-108 architecture with only 48 SPs has been identified in the power-aware optimization and a GTX 465 GPU with GF-100 architecture with 352 SPs in the energy-aware optimization. Also, the core clock is 700 MHz, compared to 780 MHz and the GDDR-3 clock rate is reduced to 450 MHz, compared to the 990 MHz of the GDDR-5 clock rate. The results for K-MEANS and NN are similar. Surprisingly, the NW tool uses the same GF-100 architecture and slightly more SPs in the power-aware optimization: 384 SPs, instead of 352 SPs. However, the clock rates are reduced from 780 MHz and 990 MHz to 690 MHz and 800 MHz.

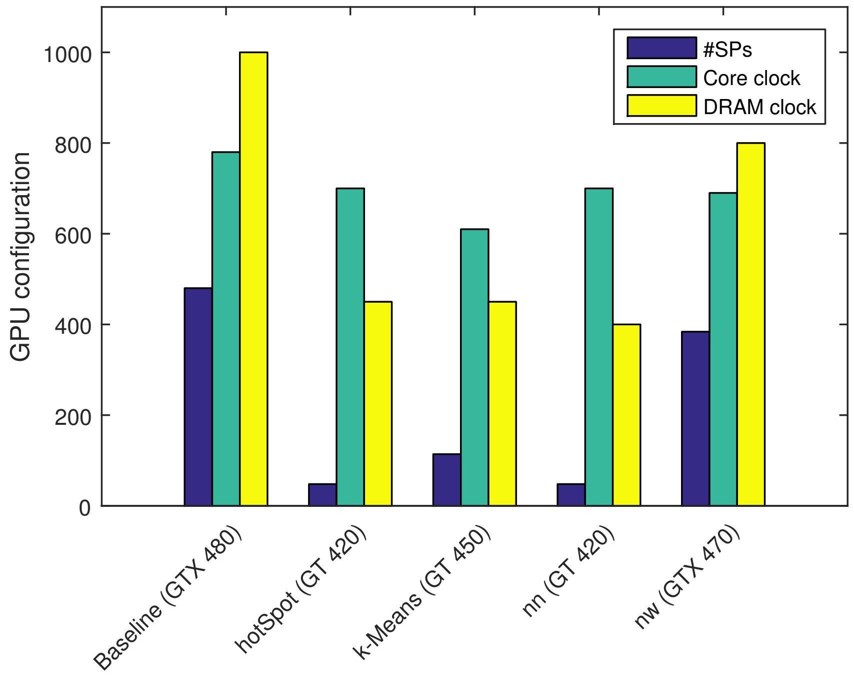


Figure 7.10: The most power efficient GPU configurations for different programs (cf. Table 7.4). The name of the identified GPU is provided in brackets. Evaluated for the architectures GF-108, GF-106, GF-104, and GF-100.

Usually, the power-aware optimization should identify slow hardware as the most power efficient hardware. But this is not true if the hardware demands of the software exceed the capabilities of the slow hardware. The most important demands are to the local and global memory. For example, some GPUs provide 64 kB local memory and others only 16 kB. If an OpenCL program uses 16 floating point values per thread and a work group size of 16×16 threads, the 16 kB local memory is already fully utilized.

7.6.5 Summary and Conclusion

In this section a single-objective energy- or power-aware DSE for HPC systems has been presented. SOG-DSE has been extended with the fitness values peak power consumption, cycles to power ratio, power to cycles ratio, energy delay product, and power delay product. These values are suited for HPC systems. The modeling of different server loads is another extension of SOG-DSE.

The evaluation has been done with power consumption as objective. A surprising result is that not always the slowest GPU with the least amount of SPs is identified as the most power efficient GPU. As can be seen for the NW and K-MEANS program, the hardware requirements of the software can limit the applicable design space.

This single-objective DSE for HPC systems is the basis for the multi-objective DSE for HPC systems in Section 8.3. Usually, power consumption is not the only objective for HPC systems. Performance and energy consumption are important and should be considered both. During the performed evaluation, this limitation of SOG-DSE has become very obvious. In the following all limitations of SOG-PSE and SOG-DSE are discussed.

7.7 Limitations of SOG-PSE and SOG-DSE

The SOG-PSE and SOG-DSE approaches have some limitations. Some of these were known during the design of the method and some did not become apparent until the performance of the experiments or until new aspects got into focus of research.

The most important limitation is that both approaches can only handle one single objective. This is well suited to optimize, for example, the detection quality or the energy consumption for a given program. However, to optimize for both objectives first the detection quality has to be optimized with SOG-PSE and then the energy consumption with SOG-DSE or the other way round. But this two pass optimization does not necessarily lead to a good result for both objectives.

A combined PSE and DSE for a hardware/software codesign is also not possible. SOG-PSE and SOG-DSE can be used sequentially, first optimizing the software and then the hardware parameters or vice versa, but this approach can prevent that an optimal solution can be found.

Another important issue is that the simulation of GPUs is very time-consuming. Running GPGPU programs on a GPU simulator can easily be by a factor of 4000 slower than running them on an actual GPU. This has been observed during the experiments for SOG-DSE. However, SOG-DSE lacks methods to speed up this time-consuming evaluation.

A related limitation is that distributed evaluation can be very important to achieve reasonable evaluation times. This was also not obvious during the design of SOG-DSE and is therefore not handled. If the evaluation is not distributed, the evaluation time is quickly limited by the performance of a single computer. Effortlessly integrating other available systems is not possible.

Additionally, the handling of dependencies in the design space has also not been solved particularly well. Especially for GPU architectures several dependencies exist that need to be modeled. Simply matching a parameter set to the nearest valid parameter set as in SOG-PSE and SOG-DSE introduces overhead that can be prevented.

Finally, the evaluation of ESs is not supported with SOG-DSE. This has to be integrated manually. The same applies to energy and performance measurements of actual GPU hardware and the integration of offloading methods.

7.8 Summary and Conclusion of the SOG-PSE and SOG-DSE Methods

In this chapter two methods have been presented, the single-objective PSE SOG-PSE and the single-objective DSE SOG-DSE. Also, a manually performed DSE has been conducted as a starting point for the automatic DSE methods.

The first method, SOG-PSE, is a single-objective PSE. This method can be used to optimize parameters for a given program regarding several objectives. The experiments provided valuable insights to the parametrization of the VirusDetectionCL use case. With SOG-PSE better detection results than with manual tuning of the parameters have been achieved. That the automatic SOG-DSE methods outperforms a manual tuning this clearly has not been expected at all. This is a highlight and shows the advantages of the SOG-PSE method.

The experiments also led to the insight that a global optimization of the parameters of VirusDetectionCL is needed as the parameters clearly influence each other. Simply optimizing one pipeline step after the other always led to worse results than with the global optimization.

Parameters for different noise sources and signal strengths could be explored automatically. This also provided insights about the difficulty of the inspected noise sources and how this noise is handled. It could be shown that the real sensor noise is more difficult to handle than synthetically added Poisson noise.

A manual DSE has been performed on mobile and desktop CPUs and GPUs as preliminary experiment for SOG-DSE. This manual DSE showed that for the VirusDetectionCL use case, the use of a GPU on a mobile system not only made real time processing possible but also saved energy compared to the use of the CPU only. This positive result motivated the development of SOG-DSE.

The second method, SOG-DSE, is an optimization method for single-objective DSE of GPU architectures. It integrates GPGPU-Sim and GPUWattch to enable a cycle accurate simulation of GPGPU programs and the application of the power model of GPUWattch based on the utilization provided by GPGPU-Sim. GPU hardware can be explored regarding several objectives such as energy consumption, power consumption, execution time, number of cycles, cycle to power ratio, power to cycle ratio, energy delay product, and power delay product. The seamlessly integration of simulation of GPUs into an automatic DSE method is clearly a highlight of this work.

The first experiment with SOG-DSE identified energy efficient GPUs for programs from three benchmarks suites. This includes also the actual GPU configuration with number of SMPs, core clock, DRAM architecture, and DRAM clock. The integration of simulation shows to be very beneficial as several GPU architectures can be explored without buying of GPUs and costly measurements.

The second experiment explored high performance GPUs for an HPC system. SOG-DSE can be used to model different server loads. Here, four programs were evaluated regarding their power consumption on the selected GPU hardware and the optimized GPU hardware has been compared to the fastest GPU. A surprising insight from this experiment is that not always the slowest GPU is the most power efficient for a given program. With this experiment also the need of a multi-objective DSE became clear. For the practical problem of identifying a high performance GPU for an HPC, more than one objective, e.g., performance, energy consumption, and maximum power consumption, needs to be considered.

Overall, three of the five PAMONO sensor scenarios, cf. Section 1.1 and Figure 1.1, have been considered in the performed experiments: the laboratory scenario with desktop and laptop systems, the server part for the cloud computing scenario, and the local processing part for the hospital and airport scenario.

The insights and learned lessons from this chapter strongly influenced the direction of research in the following. Both, SOG-PSE and SOG-DSE are the foundation for the multi-objective hybrid PSE and DSE approach that is presented in the next chapter. The goal of this method is to combine the advantages of SOG-PSE and SOG-DSE without their limitations.

Chapter 8

Hybrid PSE and DSE with MOGEA-DSE

Contents

8.1 Introduction to MOGEA-DSE 155

8.1.1 Motivation 155

8.1.2 Evaluated Scenarios 156

8.2 General Methods and Concepts of MOGEA-DSE 158

8.2.1 Architectural Design of MOGEA-DSE 158

8.2.2 Parameter Space and Design Space Definition 159

8.2.3 Multi-Objective Optimization with SPEA-2 and NSGA-II 164

8.2.4 GPUSimPow Simulator 166

8.2.5 Extension of GPUSimPow 167

8.2.6 Calculation of the Fitness Values 169

8.2.7 Summary and Conclusion 170

8.3 Multi-Objective Energy-Aware DSE of Simulated GPUs for Mobile, Desktop, and HPC Systems 170

8.3.1 Methods 171

8.3.2 Experiments 176

8.3.3 Results 179

8.3.4 Discussion 185

8.3.5 Summary and Conclusion 187

8.4 Multi-Objective Offloading-Capable Hybrid PSE and DSE of Simulated GPUs for Mobile Devices 188

8.4.1 CoPoMo LTE Simulator 189

8.4.2 Methods 190

8.4.3 Experiments 193

8.4.4 Results 194

8.4.5 Discussion 196

8.4.6 Summary and Conclusion 197

8.5 Multi-Objective Hybrid Hardware/Software PSE and DSE of an Embedded System 198

8.5.1 Methods 200

8.5.2 Experiments 206

8.5.3 Measurement Setup 208

8.5.4 Results 209

8.5.5 Discussion 223
8.5.6 Summary and Conclusion 226
8.6 Summary and Conclusion of the MOGEA-DSE Method 227

In this chapter the Multi-Objective GPGPU Energy-Aware Design Space Exploration (MOGEA-DSE) method is developed, which is a novel hybrid PSE and DSE approach for the exploration of GPGPU programs and GPU architectures. This approach combines PSE and DSE in one method, uses multi-objective optimization, is energy-aware, and can be conducted with actual or simulated GPU hardware as target systems. MOGEA-DSE can be used to optimize software and hardware parameters in a hardware/software codesign manner.

A unique feature of MOGEA-DSE is that target architectures for HPC systems, desktop systems, mobile systems, down to ESs can be explored. This makes it widely applicable as the size of explored target architecture can be scaled depending on the particular field of application. MOGEA-DSE is the most advanced PSE and DSE method within this thesis.

MOGEA-DSE is especially suited to explore CPSs, like those considered for the PAMONO sensor use case, toward a wide variety of application scenarios. Different application scenarios usually require well suited target hardware and a tuning of parameters toward the objectives that are important in this case. With MOGEA-DSE software parameters and target architectures can be automatically explored for several, possibly conflicting objectives. Hardware and software of the CPSs are considered as a unit and are explored jointly.

To show that MOGEA-DSE can be used to explore all kind of systems, it is evaluated on various physical, biological, and industrial applications and for several different target systems. The main focus is on the PAMONO sensor use case, as exploring CPSs for the several PAMONO scenarios is especially challenging.

The following research issues are investigated in this chapter:

- Is it beneficial to combine the PSE and DSE methods to enable a hardware/software codesign?
- How much energy and/or execution time can be saved if the hardware and software parameters are optimized jointly?
- How can dependencies in the parameter and design space be handled properly?
- Can the time-consuming DSE of simulated GPUs be accelerated without losing accuracy in the results?
- How can actual energy measurements be integrated?
- How can physical MPSoC hardware be explored?
- Can the demanding soft real-time limits of VirusDetectionCL be met on a small hand-held device?
- Can offloading of work to a server save energy or run time on mobile devices?
- How can approximate computing be used to save energy or execution time on ESs?

The structure of this chapter is as follows. Section 8.1 gives an introduction to MOGEA-DSE followed by general methods and concepts in Section 8.2. Afterward, three DSEs with MOGEA-DSE are presented: Section 8.3 presents a multi-objective DSE for different medical and industrial applications, explored on simulated GPUs. Section 8.4 presents a

DSE that integrates offloading using a mobile data connection. Section 8.5 presents a DSE for actual and not simulated hardware. Finally, Section 8.6 summarizes and concludes this chapter.

This chapter is based on the author's publications [LMS+14; LKD+14; NLE+15].

8.1 Introduction to MOGEA-DSE

MOGEA-DSE is a method for multi-objective, hybrid PSE and DSE of GPGPU programs with the possibility for hardware/software codesign and energy measurement of actual and simulated GPU hardware.

The first important feature of MOGEA-DSE is the combination of the SOG-PSE and SOG-DSE methods to a hybrid approach. This enables a hardware/software codesign of architectures and software.

The second important feature is that GPU target architectures for HPC systems, desktop systems, mobile systems, and ESs can be evaluated. This enables an exploration of target architectures from the largest down to the smallest systems. With MOGEA-DSE all target architectures for the PAMONO sensor use case can be explored.

The third important feature is that multiple objectives can be optimized and combined with additional constraints. This includes the objectives execution time, number of cycles, power to cycles ratio, cycles to power ratio, energy consumption, dynamic energy consumption, energy delay product, dynamic energy delay product, power consumption, dynamic power consumption, several quality measures for the VirusDetectionCL task, and generic objectives specified by the user. For the constraints, for example, an execution time limit can be defined to ensure that a soft real-time constraint can be met.

The fourth important feature of MOGEA-DSE is that offloading can be incorporated and evaluated, e.g. the best stage to offload intermediate results to a server can be determined automatically with respect to different objectives. If mobile and hand-held devices are low on battery or are too weak to perform processing by themselves, offloading of work to a powerful server or other devices nearby is useful.

The fifth important feature is that automatic measurements of actual systems can be performed with MOGEA-DSE. Within SOG-DSE and for the previous experiments for MOGEA-DSE only simulation is used. The measurements are performed for small ESs because for these systems, an accurate evaluation of execution time and energy consumption is especially important. Small errors in the simulation can cause the actual systems to miss deadlines or to consume more energy than expected.

8.1.1 Motivation

The new aspects that are considered with the MOGEA-DSE method and are motivated by the PAMONO sensor use case and the learned lessons from SOG-PSE and SOG-DSE. First, a combination of PSE and DSE into one method. Second, exploration of target architectures of all sizes, integration of multi-objective optimization, evaluation of offloading, and integration of actual measurements and not only simulations of hardware.

The importance of multi-objective hardware/software codesign became clear during the design of SOG-PSE and SOG-DSE and during the performance of the experiments for SOG-PSE and SOG-DSE. With SOG-PSE and SOG-DSE, a PSE and DSE can only be performed one after the other. Although two objectives can be optimized, this proceeding can make finding a global optimum impossible. For example, several parameter vectors might produce an optimal result in the PSE but only a few vectors might produce an optimal result for the considered hardware. If bad decisions are made in the PSE, the parameters can not be changed in the DSE and the result can be far away from the global optimum. With MOGEA-DSE the PSE and DSE are combined.

As have been shown in Section 4.1.3 to Section 4.1.5, for an efficient use of the GPU several, possibly conflicting constraints have to be fulfilled and at least two levels of memory hierarchy have to be considered. Consequently, an exploration of GPU architectures is not an easy task.

The need for exploring GPU target architectures from the largest down to the smallest systems became clear as new fields of use for the PAMONO sensor opened up with the development of VirusDetectionCL. Not only a laboratory or hospital use became possible but also a mobile use with laptops or ESs as target hardware and with the additional possibility of offloading work to an HPC system. As a consequence, the same application needs to be mapped to a variety of hardware each with other requirements.

The need of energy and performance measurements of actual GPU hardware became important for ESs as target systems for VirusDetectionCL. GPU simulators might deviate from actual hardware. For ESs small deviations between a simulation and the actual hardware might play an important role, e.g., if the simulated system meets deadlines and the actual system misses them or if the energy consumption of the actual system is higher than of the simulated.

With the hybrid PSE and DSE, the need for distributed evaluation became obvious. Evaluating several hundred configurations in a GPU simulator is time-consuming. With distributed computing large design spaces can be explored in a reasonable time.

Overall, the MOGEA-DSE method combines the advantages of SOG-PSE and SOG-DSE and resolves their most important limitations, cf. Section 7.7. Several new features are integrated into MOGEA-DSE that enable the exploration of all considered CPSs for the PAMONO sensor use case and also of many other applications.

8.1.2 Evaluated Scenarios

For the conducted experiments of MOGEA-DSE, all PAMONO scenarios from Section 1.1 and several industrial, biological, and physical use cases are considered. The evaluated PAMONO scenarios are shown in Figure 8.1: SCN1$_{\text{LAB}}$, SCN2$_{\text{STAND-ALONE}}$, SCN3$_{\text{LOCAL}}$, SCN4$_{\text{CLOUD}}$, and SCN5$_{\text{HOSPITAL}}$. For each scenario different requirements to the software, the QoR, and the GPU hardware have to be fulfilled.

For scenario SCN1$_{\text{LAB}}$, the laboratory use case, stationary, independent desktop systems are used to process PAMONO tasks. The virus detection is done, e.g., for the development of new drugs or to analyze samples that are sent to a laboratory. The main objective is the detection quality. The detection quality has already been inspected in Section 7.3. In this chapter the focus lies on identifying the best suited GPU hardware for these systems.

In scenarios SCN2$_{\text{STAND-ALONE}}$ to SCN5$_{\text{HOSPITAL}}$, PAMONO tasks are processed on mobile devices indoors and outdoors. As an example, it is conceivable that the virus detection is used as an early warning system for epidemics. In scenario SCN2$_{\text{STAND-ALONE}}$ a stand-alone hand-held device is used for the processing of PAMONO tasks. All data

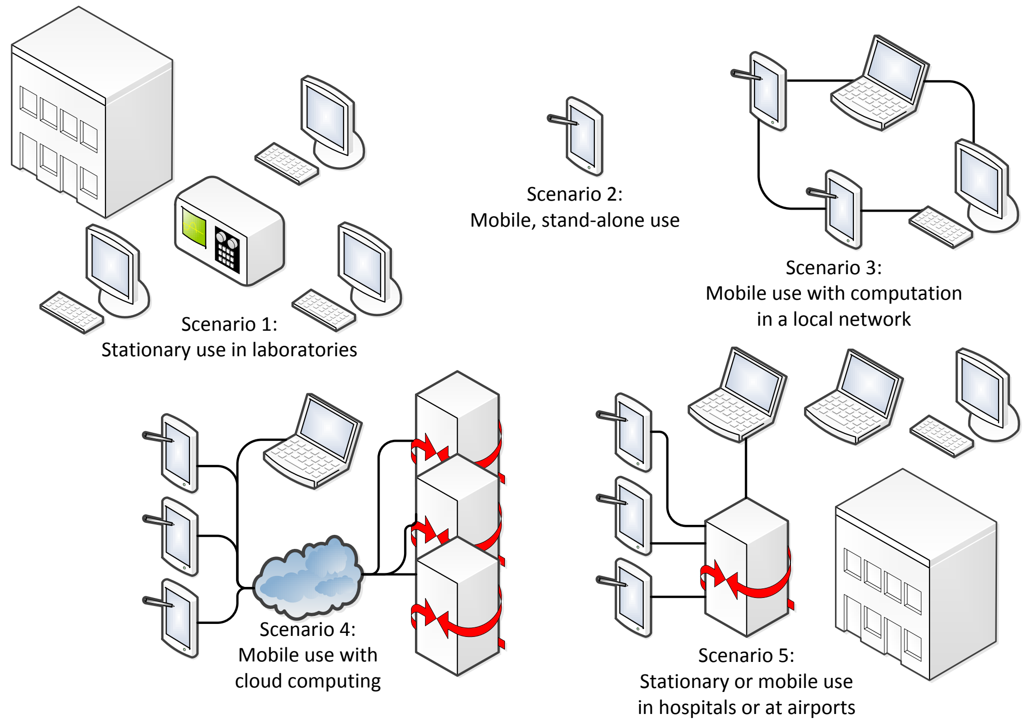

Scenario 1:
Stationary use in laboratories
Scenario 2:
Mobile, stand-alone use
Scenario 3:
Mobile use with computation in a local network
Scenario 4:
Mobile use with cloud computing
Scenario 5:
Stationary or mobile use in hospitals or at airports

Figure 8.1: Different scenarios for the PAMONO sensor use case inspected in this chapter.

has to be processed on the device. In scenario $\text{SCN3}_{\text{LOCAL}}$ and scenario $\text{SCN4}_{\text{CLOUD}}$, tasks can be partially or fully offloaded to other available devices. In $\text{SCN3}_{\text{LOCAL}}$ they are offloaded to other PAMONO devices in the area. An example is the deployment of several devices in the same area and they form a local network to share resources. The devices in the area might run on battery or have a power supply. For these two scenarios the energy consumption on the hand-held devices is especially important while execution time and detection quality are additional objectives. In $\text{SCN3}_{\text{LOCAL}}$ the offloading part is exemplary evaluated for offloading to a device with a powerful GPU and a power supply. The scenarios can dynamically change if new PAMONO devices are becoming available or leave the area.

In scenario $\text{SCN4}_{\text{CLOUD}}$ the cloud computing scenarios are shown. Mobile and hand-held devices can offload work to the cloud to save energy. One or more servers are available to process the offloaded tasks. Usually, the servers are processing different tasks and not only PAMONO tasks. A mobile data connection is used to connect the mobile devices to the cloud. Offloading to other devices in the area instead to the server is not part of this scenario. However, if the connection to the server is lost, scenario $\text{SCN4}_{\text{CLOUD}}$ changes to one of the scenarios $\text{SCN2}_{\text{STAND-ALONE}}$ or $\text{SCN3}_{\text{LOCAL}}$ depending on whether other PAMONO devices are in the area and if they can be used as an offloading target instead of the cloud.

In scenario $\text{SCN5}_{\text{HOSPITAL}}$ the hospital or airport scenarios are shown. Different devices are used, from ES, to laptops and desktop systems. A server is available for processing of data from mobile or hand-held devices. The server is used to process several samples in a short time and a fast local network is used, which is always available. The server part of these two scenarios has been evaluated in Section 7.6 but the evaluation is extended further in this chapter.

In addition to the PAMONO sensor use cases, a wide variety of industrial, biological, and physical applications are inspected for different scenarios. For a desktop system as target architecture, this includes blackScholes, histogram, quasiRandomGen, reduction, matrixMul, mersenneTwister, and vectorAdd. For a mobile system as target architecture, this includes AES, BFS, CP, LPS, NN, NQU, RAY, and STO. And for an HPC system use case this includes backprop, gaussian, hotspot, nn, and nw. For example, blackScholes is a model for price prediction, quasiRandomGen is a random number generator, AES is an encryption algorithm, NN is a neural network program on the GPU, and hotSpot is a temperature physics simulation for processors. These applications are explained in more detail in Section 8.3.2, cf. Table 8.1.

8.2 General Methods and Concepts of MOGEA-DSE

In this section the general methods and concepts of MOGEA-DSE are introduced. This includes the realization of important insights that were gained during the design of SOG-PSE and SOG-DSE. First, an introduction to the architectural design of MOGEA-DSE is given. Then methods for multi-objective optimization, parameter space and design space representation, and handling of dependencies in the parameter space and design space are presented. This is followed by a description of GPUSimPow and the extension of GPUSimPow, which plays an important role for MOGEA-DSE. Finally, the calculations of the fitness values are shown.

8.2.1 Architectural Design of MOGEA-DSE

The overall philosophy of the architectural design of MOGEA-DSE is a centralized, modular, and flexible main evaluation with an optionally decentralized quality evaluation. In the main evaluation the decisions can be made on a global basis and at the same time, the processing power of several heterogeneous systems can be used for the time-consuming quality evaluation.

Figure 8.2 shows the general architectural design of the MOGEA-DSE approach with a parallel local evaluation. The input of MOGEA-DSE consists of a definition of the parameter and design space, the objectives, a quality evaluator, program and file dependencies, program parameters, a configuration of the optimization, and the GPU architecture. These inputs are held by the main evaluation in databases.

Within the main evaluation the central management of the hybrid PSE and DSE takes place. The quality evaluation is strictly separated from the main evaluation. The main evaluation consists of the optimization, an evaluation manager, a GPU code compiler, and five databases for design space, dependencies, program parameters, GPU configurations, and evaluated results.

The optimization cycle is based on a heavily modified version of A Java-based Evolutionary Computation research system (ECJ) [LPB+14]. The cycle can be set up as a simple, single-objective GA or as a multi-objective optimization. ECJ provides methods for both cases.

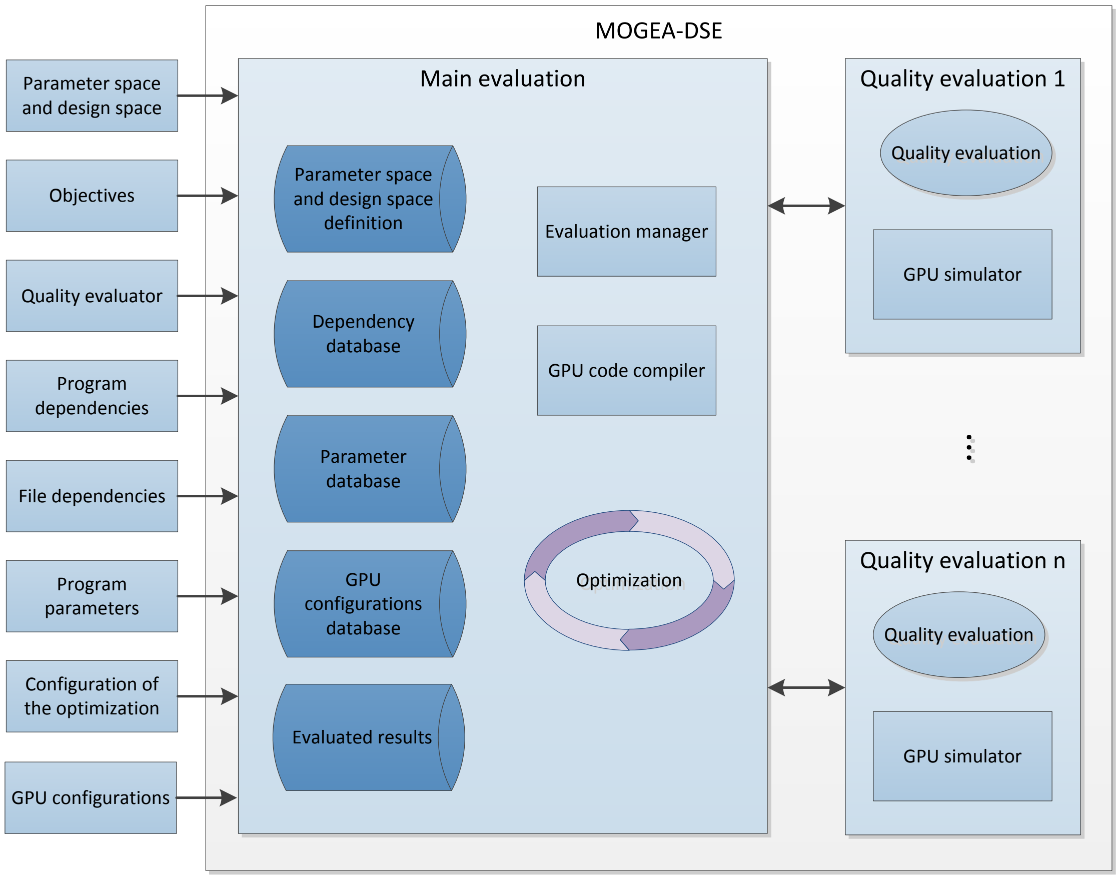


Figure 8.2: Schematic of the architectural design of MOGEA-DSE in a basic configuration.

In MOGEA-DSE software parameters of the program and hardware parameters of the GPU architecture are handled by loading all files that contain parameters into databases. The parameter files are automatically parsed to extract the names and values of the parameters. MOGEA-DSE supports files with name value pairs or eXtensible Markup Language (XML) files.

Each parameter is uniquely identified by its name and, if needed, by the filename of the corresponding parameter file. Parameters can easily be updated in the database. This includes the current allele of the parameters that are optimized and parameter dependencies that can be automatically resolved. The updated parameter files are then created within the main evaluation and transferred to the client on which the evaluation is performed.

8.2.2 Parameter Space and Design Space Definition

An important feature of a PSE and DSE is how the parameter spaces and design spaces is handled. This strongly influences which problems can be solved. In contrast to SOG-PSE and SOG-DSE, cf. Section 7.2.2, MOGEA-DSE can handle even very complex definitions of the parameter space and design space.

The input of MOGEA-DSE consists of one or more text files with all software and hardware parameters that are used and a formal description of how the parameter and design space can be spanned based on these parameters. From this input the definition of the parameter and design space is derived and finally coded as gene configuration for the EA.

Data Types

For the parameters several data types are supported. This includes the data types from SOG-PSE, SOG-DSE and all data types from ECJ. In addition, MOGEA-DSE supports some special gene types that are useful for DSE of GPUs and for optimizing GPGPU applications.

The following types are available for the gene values $i, j \in \mathbb{N}$ and $x \in \mathbb{R}$ and the output $o \in \mathbb{R}$:

- `INTEGER_TYPE` with $o := i$
- `FLOAT_TYPE` with $o := x$
- `INTEGER_POWER_OF_TWO_TYPE` with $o := 2^i$
- `FLOAT_POWER_OF_TWO_TYPE` with $o := 2^x$
- `INTEGER_MUL_`10^j`_TYPE` with $o := i \cdot 10^j$
- `INTEGER_DIV_`10^j`_TYPE` with $o := \frac{i}{10^j}$
- `GPGPU_SIM_CLOCK_DOMAINS_TYPE`
- `GPGPU_SIM_SHADER_CORE_PIPELINE_TYPE`

The first two types in this list are the regular ECJ types for integer and floating point numbers, whereas the following types are part of MOGEA-DSE.

Powers of two are often needed for buffer sizes or work group sizes in GPGPU programs. Thus, instead of optimizing the o in $o := 2^i$, the i in this equation is optimized.

The types where the integer value is multiplied with an integer power of ten are useful, for example, if frequency scaling is used. Clock rates only can be varied in 100 kHz steps on some devices. Therefore, simply the i value in the equation $i \cdot 10^j$ can be optimized, the j can be set to 2, and the minimum and maximum of the range for the optimization can be divided by 100.

The types with integer values divided by powers of ten are useful if the accuracy of floating point values is not needed. Instead of optimizing floating point numbers, integer values are optimized that are afterward converted to floating point numbers with the needed accuracy. Only numbers of type $\frac{i}{10^j}$ are considered. This approach is equal to using fixed point numbers with j fractional decimal digits. This has the advantage that very small changes in the gene values are not possible. These small changes would require a new evaluation but would not change the result.

Finally, two GPGPU-Sim types are introduced to handle the clock and shader core configuration of GPGPU-Sim that are stored in a proprietary format. In this format several numbers are packed, separated by colons, into one string that is parsed automatically by MOGEA-DSE.

Simple Parameters

Based on the data types, a formal description of the parameter and design space can be made. A reference to a parameter, the type of the parameter, a range of valid values, and a start value have to be given. This defines the parameter and design space if it is provided for all parameters. This definition can be directly translated to a gene configuration of the EA.

The following Listing 8.1 shows an excerpt of two parameters from the parameter space, which are coded into genes.

```
pop.subpop.0.species.segment.1.start = 3

pop.subpop.0.species.gene-name.3 = gaussImage
pop.subpop.0.species.gene-type.3 = INTEGER_TYPE
pop.subpop.0.species.min-gene.3 = 0
pop.subpop.0.species.max-gene.3 = 1
pop.subpop.0.species.gene-start-value.3 = 1

pop.subpop.0.species.gene-name.4 = gaussImageSigma
pop.subpop.0.species.gene-type.4 = INTEGER_DIV_1000000_AS_FLOAT_TYPE
pop.subpop.0.species.min-gene.4 = 1500000
pop.subpop.0.species.max-gene.4 = 3500000
pop.subpop.0.species.gene-start-value.4 = 1500000
```

Listing 8.1: Gene configuration

In the listing, first, a new segment that groups the genes to chromosomes is given. Then two genes are defined. The first gene is a flag if the Gaussian noise reduction, cf. Equation (5.11), should be enabled. This parameter has the name *gaussImage* and its boolean value is codes as the integer numbers 0 and 1. The second gene defines that the sigma from Equation (5.11) should be optimized. The parameter is named *gaussImageSigma*. This value is optimized as a fixed point number with 6 fractional decimal digits. The range of the optimization is from 1.5 to 3.5 and the start value for the first individual is 1.5 for the sigma.

The next section describes how parameter dependencies and more difficult ranges can be handled.

Modeling of Dependencies

For MOGEA-DSE a method for handling dependencies in the parameter and design space is introduced. This method is useful to handle complex parameter and design spaces and is also useful for search space pruning.

If several types of GPUs or GPU architectures are to be examined, several hardware parameters need to be varied. For example, in this work the following GPU parameters are varied: the number of SMPs, the number of SPs, the clock rates for core, shader, interconnect DRAM, and cache, the DRAM bus width, the number of Raster Operation Processors (ROPs), and the number of Texture Mapping Units (TMUs). Most of these parameters heavily depend on each other. The number of SMPs in a GPU affects the total number of SPs, ROPs, and TMUs. Additionally, some software parameters might also be dependent on the target architecture. Current approaches do not make use of these dependencies.

As simple genes can not model dependencies of values, the genes might express all valid but also some invalid configurations. In SOG-DSE dependencies of hardware parameters for the exploration of GPU architectures are handled by mapping invalid gene configuration to the nearest valid parameter set in the design space, cf. Section 7.5.3. This results in a design space that is only effectively smaller. The problem with this method is that it does not keep the overall design space small. Also, several invalid configurations might match to the same valid configuration. This introduces overhead and influences how the design space is sampled.

One solution to this problem is to enhance the process of creating new individuals in a way that only valid values are generated in the fitness function. However, this requires that the fitness function needs to be rewritten and recompiled if the parameter dependencies change. However, here, a different approach is chosen. The main objective is to keep the design space small. Therefore, the number of genes is reduced to a number of necessary genes and the rest of the parameters is derived from these genes. With this approach all kind of parameter dependencies can be handled.

Listing 8.2 shows an example of how simple parameter dependencies can be resolved. The work group width and height can be formulated as a power of two and the size of the work group as a dependency of the width and height that can be resolved by multiplying width and height. If parameters from different parameter files should be optimized, the file name can be appended to the parameter name.

```
pop.subpop.0.species.segment.14.start = 36
pop.subpop.0.species.gene-name.36 =
    #define_MARCHING_SQUARES_WORK_GROUP_WIDTH@config.h
pop.subpop.0.species.gene-type.36 = INTEGER_POWER_OF_TWO_TYPE
pop.subpop.0.species.min-gene.36 = 1
pop.subpop.0.species.max-gene.36 = 3
pop.subpop.0.species.gene-start-value.36 = 3
pop.subpop.0.species.gene-name.37 =
    #define_MARCHING_SQUARES_WORK_GROUP_HEIGHT@config.h
pop.subpop.0.species.gene-type.37 = INTEGER_POWER_OF_TWO_TYPE
pop.subpop.0.species.min-gene.37 = 1
pop.subpop.0.species.max-gene.37 = 3
pop.subpop.0.species.gene-start-value.37 = 3
pop.subpop.0.species.parameter-dependency-name.0 =
    #define_MARCHING_SQUARES_WORK_GROUP_SIZE@config.h
pop.subpop.0.species.parameter-dependency-type.0 = INTEGER_TYPE
pop.subpop.0.species.parameter-dependency-operation.0 = MULTIPLY_OPERATION
pop.subpop.0.species.parameter-dependency-operand0.0 =
    #define_MARCHING_SQUARES_WORK_GROUP_WIDTH@config.h
pop.subpop.0.species.parameter-dependency-operand1.0 =
    #define_MARCHING_SQUARES_WORK_GROUP_HEIGHT@config.h
```

Listing 8.2: Parameter dependencies

For simple dependencies the following operations are available for the operands $a, b, c \in \mathbb{R}$ and the output $o \in \mathbb{R}$:

- `IDENTITY_OPERATION` with $o := a$
- `ADD_OPERATION` with $o := a + b$
- `SUB_OPERATION` with $o := a - b$
- `MULTIPLY_OPERATION` with $o := a \cdot b$

- DIV_OPERATION with $o := a/b$
- MAD_OPERATION with $o := a \cdot b + c$
- POWER_OF_TWO_OPERATION with $o := 2^a$

The operands can be numbers or other parameters. The output is casted to the correct type of the parameter.

To model more complex dependencies, a JavaScript®interpreter is integrated into MOGEA-DSE. As a result, the full JavaScript®language can be used to calculate the dependencies. The JavaScript®interpreter can be addressed with EVAL_OPERATION as the operation of the dependency.

For example, if the NVIDIA®architectures GF-108 to GF-100, cf. Table 8.2, should be evaluated, the number of SPs per SMP is 48 for GF-108, GF-106, and GF-104 and 32 for GF-100. This needs to be reflected by the parameters. Because the architecture is uniquely identified by the number of SMPs, the number of SPs per SMP can be formulated as a dependency of the number of SMPs. Also, other parameters vary with the selected architecture and can therefore be derived from the number of SMPs. As a result, the design space and therefore the evaluation time can be smaller.

An example of some dependencies of the NVIDIA®architectures GF-108 to GF-100 is given in Listing 8.3 as pseudo code: the number of SPs per SMP, the DRAM bus width, the shader clock speed, the number of TMUs, and the number of ROPs.

```
1 num_SP_per_SM = (number_of_sm < 11) ? 48 : 32;
2 dram_buswidth = (number_of_sm >= 3) ? 4 : 2;
3 shader_clock = 2 * target_core_clockrate;
4 num_tmus = 4 * number_of_sm;
5 num_rops = 3 * number_of_sm;
```

Listing 8.3: Parameter dependencies as pseudo code

Listing 8.3 shows how hardware parameters for the GF-108, GF-106, and GF-104 architectures depend on each other. In line one for all architectures with less than 11 SMPs, the number of SPs per SMP is set to 48 and for the GF-100 with 11 or more SMPs it is set to 32. Additionally, the slower GPUs use slower DRAM. The DRAM bus width is four for all GPUs with three or more SMPs and two for all the others as stated in line two. The shader clock rate is twice the core clock rate, cf. line three. Finally, there are four TMUs on every SMP and three ROPs as described in the last two lines. This pseudo code can easily be translated to parameter dependencies in MOGEA-DSE.

```
pop.subpop.0.species.parameter-dependency-name.7 = num_SP_per_SM
pop.subpop.0.species.parameter-dependency-type.7 = INTEGER_TYPE
pop.subpop.0.species.parameter-dependency-operation.7 = EVAL_OPERATION
pop.subpop.0.species.parameter-dependency-operand0.7 =
    (@@number_of_sm@@ < 11) ? 48 : 32
pop.subpop.0.species.parameter-dependency-name.8 = dram_buswidth
pop.subpop.0.species.parameter-dependency-type.8 = INTEGER_TYPE
pop.subpop.0.species.parameter-dependency-operation.8 = EVAL_OPERATION
pop.subpop.0.species.parameter-dependency-operand0.8 =
    (@@number_of_sm@@ >= 3) ? 4 : 2
```

Listing 8.4: More complex parameter dependencies

Listing 8.4 shows how this is done. For both parameter dependencies first the name of the parameter is given, then the type, and third that it needs to be evaluated with the

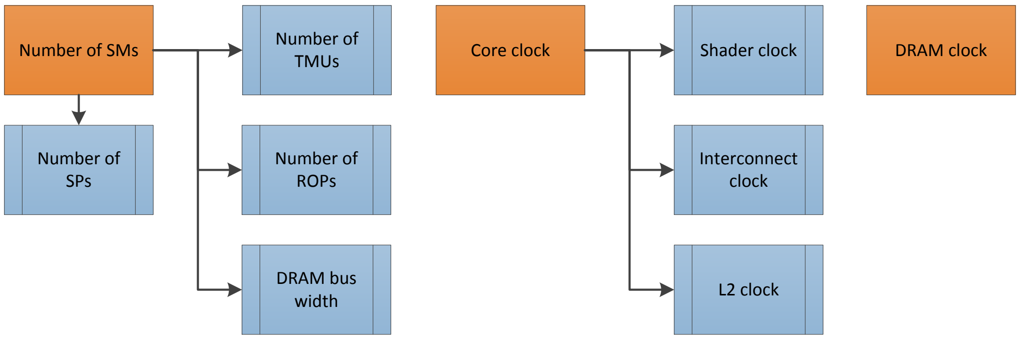


Figure 8.3: Example for optimized and derived parameters. The number of SMPs, the core clock, and the DRAM clock are the optimized parameters. All other parameters can be directly derived from these three parameters.

JavaScript®interpreter. In the last line the evaluation operation is given analogously to Listing 8.3 as JavaScript®code.

The first two lines from Listing 8.3 can be evaluated with the JavaScript®interpreter by defining them as `EVAL_OPERATION`. The multiplications can be formulated as a simple multiply operation with two operands similar to the multiply operation in Listing 8.2.

With `@@number_of_sm@@` a reference to another parameter is defined. Such references are resolved automatically by MOGEA-DSE with the current allele of the referenced parameters. This enables a very easy and flexible definition of the parameter dependencies.

An example, of how GPU parameters can be derived from each other is shown in Figure 8.3. Only three parameters need to be coded in genes: the number of SMPs, the core clock, and the DRAM clock. All other parameters can be derived unambiguously.

Overall, MOGEA-DSE can handle very complex parameter and design spaces. The use is not limited to the shown dependencies. New values can be derived with the `EVAL_-OPERATION` operation from other genes. For example, a lookup table can translate a simple integer gene to other values or some function can be evaluated on a floating point gene. For the latter, the input of a function is modeled as gene and the function is modeled as parameter dependency. The output of the evaluation is then used for the quality evaluation.

8.2.3 Multi-Objective Optimization with SPEA-2 and NSGA-II

A key feature of the MOGEA-DSE method is that it supports multi-objective optimization. The multi-objective optimization in MOGEA-DSE can be done with SPEA-2 [ZLT01] or NSGA-II [DAP+00; DPA+02]. Both are very well established methods for multi-objective optimization and part of ECJ [LPB+14]. SPEA-2 and NSGA-II are introduced in the following.

The overall evaluation cycle is similar to that of SOG-PSE and SOG-DSE, cf. Section 7.2.3. It also consists of evaluation, selection, recombination, and mutation. However, the main fitness evaluation is toward multiple objectives and the selection and recombination step is replaced with SPEA-2 or NSGA-II.

SPEA-2 has been developed by Zitzler, Laumanns, and Thiele. It uses a population and an additional archive. In each generation all non-dominated individuals from the current population and the previous archive are stored in the new archive. If the archive exceeds a given, fixed archive size, some individuals are truncated. If the number of individuals is less than the archive size, it is filled up with dominated individuals. Individuals for recombination are selected from the archive only with a tournament selection. A new population is then build after recombination and mutation on the selected individuals. [ZLT01]

The key factor of SPEA-2 is the fine grained fitness assignment strategy. First, a strength value is calculated for each individual. The strength value is the number of solutions in archive and population that the individual dominates, cf. Definition 49. Second, a raw fitness is calculated for each individual. The raw fitness is the sum of the strength values of all individuals that dominate the individual. In result, if more strong individuals dominate the current individual a higher raw fitness value is assigned. Thus, lower raw fitness values are better. Individuals on the Pareto front have a raw fitness value of zero. Third, in addition to the raw fitness value, a density value is considered. It calculates as the inverse of the distance to the k-th nearest neighbor in objective space, with k equal to the square root of the combined archive and population. Thus, if the k-th nearest neighbor of an individual is far away, the density value is small. Finally, the fitness calculates as sum of the raw fitness and the density value. As the density value is small, the overall fitness value is dominated by the raw fitness value and the density is used to discriminate between individuals with the same raw fitness. [ZLT01]

NSGA-II has been developed by Deb et al. to overcome problems with existing multi-objective optimization methods. It reduced the computational complexity, uses elitism, and does not rely on the concept of sharing. The method works as follows. First, a new children population is built from the parent population with tournament selection, recombination, and mutation. This children population is then evaluated and combined with the parent population. Second, NSGA-II uses a fast non-dominated sorting of the front to determine the fitness. This is done by successively building a set of non-dominated individuals. Only individuals in this set are compared for dominance, which reduces the complexity of this approach. If the set has been built, it contains only the non-dominated Pareto front. Individuals from this front get the rank one. These individuals are then removed and the sorting is repeated to achieve the individuals with rank two and so forth. Third, a crowding distance is calculated for every individual. This is done by determining, for each objective, the size of the largest cuboid enclosing this individual that does not include any other individual with the same rank. Finally, a partial order is calculated on all individuals. For individuals with unequal rank, a smaller rank is better. For individuals with equal rank, a larger crowding distance is better. The first N many individuals from this set form the next parent population. [DAP+00]

In summary, SPEA-2 and NSGA-II are both elitist EAs for multi-objective optimization. SPEA-2 uses ranking combined with density information to select individuals for breeding. In contrast, NSGA-II uses non-dominated sorting combined with a crowding distance for the selection of individuals. Both methods prefer individuals near the Pareto front and use geometric information in objective space, which favors a wider distribution of individuals.

8.2.4 GPUSimPow Simulator

To demonstrate that the MOGEA-DSE approach is a modular design and not dependent of the actual GPU simulator, GPUSimPow [LLA+13] is used as an alternative to GPGPU-Sim and GPUWattch for some evaluations.

GPUSimPow is a GPU simulator with an integrated power consumption model developed by Lucas et al. It uses the GPGPU-Sim [BYF+09] as GPU simulator and extends it with the GPU power model GPGPU-Pow. Like GPUWattch, GPGPU-Pow is an extension of McPAT [SAS+09] but has been developed independently of GPUWattch.

The input of GPUSimPow is PTX code of the GPU program and an actual execution of this GPU program. The output is the estimated chip area, the estimated peak dynamic power consumption, the total leakage, and also average power consumption and average execution time for every unique kernel. However, neither the estimated overall power consumption nor the estimated overall energy consumption is provided.

To simulate the power consumption of the SPs and the memory, McPAT [SAS+09] is used, which is an analytical model. For the address generation units, special function units, and other irregular components an empirical model is used. This model has been created by the authors with a measurement setup. The setup consists of a mainboard with a raiser card that contains the GPU that should be measured. The power consumption is measured with probes for voltage and current on the raiser card and on the external power connectors of the GPU. With additional data acquisition hardware and a measurement software, energy consumption and average power consumption can be measured for each kernel run. The measurement setup has also been used to validate GPUSimPow. [LLA+13]

In GPUSimPow the power consumption for switching circuits P_{circuit} is modeled as [LLA+13]

$$P_{\text{circuit}} = \alpha \cdot C \cdot V_{\text{dd}} \cdot \Delta f_{\text{clock}} + V_{\text{dd}} \cdot I_{\text{short-circuit}} + V_{\text{dd}} \cdot I_{\text{leakage}}, \tag{8.1}$$

with α as the activity factor, $CV_{\text{dd}}\Delta f_{\text{clock}}$ as the dynamic power, $V_{\text{dd}}I_{\text{short-circuit}}$ as the short circuit power, and $V_{\text{dd}}I_{\text{leakage}}$ as the static leakage power. The activity factor α is determined with GPGPU-Sim, which is used by GPUSimPow. The short circuit power $I_{\text{short-circuit}}$ and the dynamic power I_{leakage} are determined with the GPGPU-Pow power model, which is the main part of GPUSimPow.

The average dynamic power consumption $\widetilde{P}_{\text{dynamicAvg}} : \mathbb{N} \to \mathbb{R}^+$ of $n_k \in \mathbb{N}$ runs of the same kernel k is calculated in GPUSimPow by

$$\widetilde{P}_{\text{dynamicAvg}}(k) = \frac{1}{n_k} \cdot \sum_{i=1}^{n_k} \widetilde{P}_{\text{dynamic}}(k,i) \tag{8.2}$$

with $\widetilde{P}_{\text{dynamic}}(k,i)$ as the power consumption of the i-th run of the kernel k. The average execution time $\widetilde{T}_{\text{avg}} : \mathbb{N} \to \mathbb{R}^+$ is calculated by

$$\widetilde{T}_{\text{avg}}(k) = \frac{1}{n_k} \cdot \sum_{i=1}^{n_k} \widetilde{T}(k,i), \tag{8.3}$$

with $\widetilde{T} : \mathbb{N} \times \mathbb{N} \to \mathbb{R}^+$ as the execution time of the i-th run of the kernel k. These two equations are important for the extension of GPUSimPow in Section 8.2.5 and for the calculation of the fitness values in Section 8.2.6.

8.2.5 Extension of GPUSimPow

To integrate GPUSimPow into MOGEA-DSE, some modifications have been made to GPUSimPow. These modifications affect the output of GPUSimPow. As shown in the following, the estimated average power consumption output of GPUSimPow is not suitable to optimize the energy or power consumption of programs that call data dependent kernels several times like VirusDetectionCL does. In VirusDetectionCL several sensor images are streamed through the pipeline. Therefore, almost all kernels are called more than once. GPUSimPow does not handle these multiple calls appropriately.

The output of GPUSimPow is similar to the output of GPGPU-Sim and GPU-Wattch in Section 7.5.3 but there are differences. The output of GPUSimPow contains the static power consumption $\widetilde{P}_{\text{static}}$ but not the dynamic power consumption $\widetilde{P}_{\text{dynamic}}$. Instead of the overall dynamic power consumption, an averaged dynamic power consumption for every single kernel, averaged over the number of kernel runs, is given. The averaged power consumption $\widetilde{P}_{\text{dynamicAvg}}(k)$ of kernel k over n_k runs is calculated according to Equation (8.2) and the averaged execution time $\widetilde{T}_{\text{avg}} : \mathbb{N} \to R^{+}$ over n_k runs is calculated according to Equation (8.3). It should be noted that the intermediate results $\widetilde{P}_{\text{dynamic}}(k,i)$ in Equation (8.2) and $\widetilde{T}(k,i)$ in Equation (8.3) are not available in the output of GPU-SimPow. Only the averaged power consumption and averaged execution time is part of the output.

With the estimated static power consumption $\widetilde{P}_{\text{static}}$ and estimated average dynamic power consumption $\widetilde{P}_{\text{dynamicAvg}}(k)$ for every kernel k, the estimated power consumption $\widetilde{P}_{\text{GPUSimPow}}$ can be obtained as

$$\widetilde{P}_{\text{GPUSimPow}} = \widetilde{P}_{\text{static}} + \frac{1}{\sum_{k=1}^{K} n_k} \cdot \sum_{k=1}^{K} n_k \cdot \widetilde{P}_{\text{dynamicAvg}}(k), \tag{8.4}$$

with $K \in \mathbb{N}$ as the number of different kernels.

A major problem with this average power consumption in Equation (8.3) and average execution time in Equation (8.3) over more than one run is that the overall energy consumption can usually not be restored correctly from these two values even if the number of runs n_k for every kernel k is known. Hence, the output of GPUSimPow can not be used to calculate the energy consumption if at least one n_k is not one, i.e. one of the kernels is executed more than once.

Especially, it is usually *incorrect* to calculate the overall energy consumption with the following equation

$$\widetilde{E}_{\text{GPUSimPow}} = \widetilde{P}_{\text{static}} \cdot \sum_{k=1}^{K} n_k \cdot \widetilde{T}_{\text{avg}}(k) + \sum_{k=1}^{K} n_k \cdot \widetilde{P}_{\text{dynamicAvg}}(k) \cdot \widetilde{T}_{\text{avg}}(k), \tag{8.5}$$

with $K \in \mathbb{N}$ as the number of different kernels.

In VirusDetectionCL the execution time and energy consumption of most kernels is only dependent on the size of the images and not on the content, e.g., for the Gaussian filter. As the image size does not change, it would be appropriate to just calculate the power consumption based on averaged power consumptions of each kernel run with Equation (8.4) and Equation (8.5). However, for data dependent kernels like the segmentation in Section 5.5 or the Random Forest classification in Section 5.6.2, where the energy consumption and execution time varies considerably, it is inappropriate to use average values to calculate the overall energy consumption. In these cases it can lead to wrong results.

To demonstrate that the information of the overall energy consumption can get lost with averaging of the power consumption and averaging of the execution times, the following example should be given. For the kernel k with two kernel runs, the first run should have a power consumption of 100 W and an execution time of 0.1 s and the second kernel run should have a power consumption of 10 W and an execution time of 0.9 s. According to Equation (8.3) the averaged power consumption of kernel k is $\widetilde{P}_{\mathrm{avg}}(k) = \frac{1}{2} \cdot (100\,\mathrm{W} + 10\,\mathrm{W}) = 55\,\mathrm{W}$. According to Equation (8.3) the averaged execution time calculates as $\widetilde{T}_{\mathrm{avg}}(k) = \frac{1}{2} \cdot (0.1\,\mathrm{s} + 0.9\,\mathrm{s}) = 0.5\,\mathrm{s}$. These two values are available in the GPUSimPow output. If the number of runs n is known, two attempts can be done to calculate the actual energy consumption with these values.

First, it can be calculated directly from the two averaged values multiplied with the known number of runs resulting in $(2 \cdot 55\,\mathrm{W}) \cdot (2 \cdot 0.5\,\mathrm{s}) = 110\,\mathrm{J}$.

Second, it can be calculated by calculating the energy consumption of each kernel run and sum these up. As the power consumption and execution time of each kernel run is unknown it has to be assumed that each of the two kernel runs took half of the power and half of the execution time resulting in $(55\,\mathrm{W} \cdot 0.5\,\mathrm{s}) + (55\,\mathrm{W} \cdot 0.5\,\mathrm{s}) = 55\,\mathrm{J}$.

However, both of these attempts lead to wrong results. The actual energy consumption of kernel k is $(100\,\mathrm{W} \cdot 0.1\,\mathrm{s}) + (10\,\mathrm{W} \cdot 0.9\,\mathrm{s}) = 19\,\mathrm{J}$ and neither 110 J nor 55 J. This actual energy consumption of 19 J can not be calculated from the GPUSimPow output which consists of the averaged power consumption of 55 W and the averaged execution time of 0.5 s even if the number of runs is known. This is because the information of the power consumption of the single runs and the execution time of the single runs is lost if the values are averaged. Also, the averaged energy consumption and the overall power consumption can not be restored from the averaged power values. The actual averaged energy consumption is $\frac{19\,\mathrm{J}}{2} = 9.5\,\mathrm{J}$ and neither $\frac{110\,\mathrm{J}}{2} = 55\,\mathrm{J}$ nor $\frac{55\,\mathrm{J}}{2} = 27.5\,\mathrm{J}$.

In fact, the actual overall power consumption is $\frac{19\,\mathrm{J}}{1\,\mathrm{s}} = 19\,\mathrm{W}$, which is much lower than the averaged power consumption of 55 W calculated in GPUSimPow.

To sum up, neither the energy consumption nor the power consumption can be calculated correctly from the GPUSimPow output if kernels with varying execution time or energy consumption are called more than once.

To solve this problem, GPUSimPow has been modified in order to receive the correct values for the power and energy consumption of multiple kernel runs of the same kernels. All changes have been provided to the authors of GPUSimPow and were confirmed as correct.

The changes consist of the following three equations. First, instead of the averaged execution time, cf. Equation (8.3), here the overall execution time $\widetilde{T}$ of kernel k over n_k runs is calculated, given as

$$\widetilde{T}(k) = \sum_{i=1}^{n_k} \widetilde{T}(k,i), \tag{8.6}$$

with $\widetilde{T} : \mathbb{N} \times \mathbb{N} \to \mathbb{R}^{+}$ as the execution time of the i-th run of kernel k. Second, the sum over all K kernel execution times results in the estimated overall execution time $\widetilde{T}_{\mathrm{GPU}}$:

$$\widetilde{T}_{\mathrm{GPU}} = \sum_{k=1}^{K} \widetilde{T}(k). \tag{8.7}$$

Third, the dynamic power consumption $\widetilde{P}_{\text{dynamic}} : \mathbb{N} \rightarrow \mathbb{R}^+$ for kernel k is calculated as

$$\widetilde{P}_{\text{dynamic}}(k) = \frac{1}{\widetilde{T}(k)} \cdot \sum_{i=1}^{n} \widetilde{P}_{\text{dynamic}}(k,i) \cdot \widetilde{T}(k,i), \tag{8.8}$$

with $\widetilde{P}_{\text{dynamic}}(k,i)$ as the dynamic power consumption of the k-th kernel in the i-th run and $\widetilde{T}(k,i)$ as the corresponding execution time. This equation obtains the power consumption by calculating the energy consumption and then dividing it by the overall execution time of all runs. The energy consumption is calculated by summing up the dynamic power consumption of the kernel runs multiplied with their execution time.

As a result, Equation (8.8) calculates the actual power consumption and not an average power consumption as in GPUSimPow. In GPUSimPow the power consumption is weighted by the number of runs, whereas here it is weighted by the execution time. Based on this equation also the actual energy consumption can be calculated, simply, by multiplying it with the overall execution time as shown in the following section.

8.2.6 Calculation of the Fitness Values

For MOGEA-DSE the available fitness values of SOG-DSE are extended to execution time, number of cycles, power to cycles ratio, cycles to power ratio, energy consumption, dynamic energy consumption, energy delay product, dynamic energy delay product, power consumption, and dynamic power consumption as described in the following.

The same approach as in Equation (8.8) is used to achieve the estimated overall dynamic power consumption as $\widetilde{P}_{\text{dynamic}}$

$$\widetilde{P}_{\text{dynamic}} = \frac{1}{\widetilde{T}_{\text{GPU}}} \cdot \sum_{k=1}^{K} \widetilde{P}_{\text{dynamic}}(k) \cdot \widetilde{T}(k), \tag{8.9}$$

with K as the number of kernels. Combined with the static power consumption $\widetilde{P}_{\text{static}}$, the estimated overall power consumption $\widetilde{P}_{\text{GPU}}$ can be obtained. This can be achieved with Equation (8.4), which is repeated here for reasons of clarity:

$$\widetilde{P}_{\text{GPU}} = \widetilde{P}_{\text{static}} + \widetilde{P}_{\text{dynamic}}. \tag{8.10}$$

By multiplying the overall power consumption $\widetilde{P}_{\text{GPU}}$ with the overall execution time, the estimated energy consumption $\widetilde{E}_{\text{GPU}}$ can be obtained:

$$\widetilde{E}_{\text{GPU}} = \widetilde{P}_{\text{GPU}} \cdot \widetilde{T}_{\text{GPU}}. \tag{8.11}$$

If only the dynamic energy $\widetilde{E}_{\text{dynamic}}$ is of interest, it can be calculated as

$$\widetilde{E}_{\text{dynamic}} = \widetilde{P}_{\text{dynamic}} \cdot \widetilde{T}_{\text{GPU}}. \tag{8.12}$$

With the execution time $\widetilde{T}_{\text{GPU}}$ in seconds and the known shader clock rate S in Hertz, the number of cycles can be calculated as

$$\widetilde{C}_{\text{GPU}} = \widetilde{T}_{\text{GPU}} \cdot S. \tag{8.13}$$

With the used cycles $\widetilde{C}_{\text{GPU}}$ and the power consumption $\widetilde{P}_{\text{GPU}}$, the cycles to power consumption ratio and the power consumption to cycles ratio fitness values can be calculated according to Equation (7.10) and Equation (7.11).

Finally, the energy delay product can be calculated with Equation (7.12) and the dynamic energy delay product is analogously calculated as

$$\text{DynamicEnergyDelayProduct} = (\widetilde{E}_{\text{dynamic}})^{\alpha} \cdot (\widetilde{T}_{\text{GPU}})^{\beta}. \tag{8.14}$$

with $\alpha, \beta \in \mathbb{R}^{+}$ as weights.

8.2.7 Summary and Conclusion

In this section the general methods and concepts for MOGEA-DSE have been introduced. The first important aspect is a parallel and possibly distributed evaluation. This is needed to achieve a reasonable low evaluation time as the simulation of GPUs is slow and the search space is usually large.

The second aspect is how the parameter space and design space can be defined properly. The search space should be kept small. Therefore, a method for handling dependencies in the parameter and design space have been developed. The overall search space can be kept smaller by evaluating floating point parameters only with the needed degree of quality and by deriving parameters from other parameters and genes.

The third aspect is the integration of multi-objective optimization. For many problems it is crucial to optimize toward more than one objective, e.g., toward a low execution time and low energy consumption.

The fourth aspect is the integration of GPUSimPow GPU simulator and power model as an alternative to the GPGPU-Sim GPU simulator and GPUWattch as power model. To achieve this changes had to be made on the calculation of the output of GPUSimPow.

Finally, the calculation of fitness values has been extended. This makes the results from GPUSimPow available for MOGEA-DSE.

In the next section these general methods and concepts lead to a more concrete use of MOGEA-DSE and the first experiments and evaluations.

8.3 Multi-Objective Energy-Aware DSE of Simulated GPUs for Mobile, Desktop, and HPC Systems

In this section MOGEA-DSE is extended toward a multi-objective, energy-aware DSE on simulated GPUs. The range of the explored systems is from bigger mobile devices like laptops, to regular desktop systems, up to HPC servers. The main focus in this section is on making use of the general methods from Section 8.2. Additionally, methods for a faster evaluation are introduced.

As more and more medical, physical, or industrial applications are using GPUs for processing, it is necessary to identify fast and/or energy-efficient GPUs for these systems. Several, possibly conflicting objectives have to be fulfilled and additional constraints have to be met. If a system for a specific medical, physical, or industrial application should be equipped with GPUs to increase the processing power of the system and/or to keep the energy consumption low, it is beneficial to know in advance which GPUs meet the requirements of the system. With MOGEA-DSE suitable GPU systems can be identified for given programs. Several programs can be weighted to model more complex work loads.

The evaluated PAMONO sensor scenarios from Figure 8.1 are $\text{S}\textsc{cn}1_{\text{Lab}}$, $\text{S}\textsc{cn}3_{\text{Local}}$, $\text{S}\textsc{cn}4_{\text{Cloud}}$, and $\text{S}\textsc{cn}5_{\text{Hospital}}$. Only the not hand-held devices from these scenarios are considered and the communication is also not part of this section.

For VirusDetectionCL a soft real-time constraint has to be fulfilled if it processes images directly from the camera. Therefore, the execution time is particularly important. If a certain processing has to be met, the most energy efficient solution might not be feasible. The execution time can also be taken as a constraint to the optimization process or as a further objective function, naturally leading to multi-objective optimization. In the context of the PAMONO sensor all three inspected device sizes are of interest: mobile devices for a flexible use of the PAMONO sensor in various indoor or outdoor environments, in hospitals, and at airports, desktop devices for a use in laboratories or hospitals, and servers for a central processing of tasks.

The following research issues are examined in this section:

- How can MOGEA-DSE be used to identify GPU hardware for HPC servers?
- Which is the most energy-efficient GPU for a mobile device that complies with the soft real-time limits?
- Can the possibly contrary objectives, fast processing and energy-efficiency, be achieved at the same time?
- Is it possible to identify best suited GPUs for given programs without buying and testing them?
- Can upcoming GPUs, that do not yet exist, be explored with a DSE?

Especially the first research issue, identifying GPU hardware for HPC servers, is worth to be considered, as more and more compute servers and HPC systems will be equipped with GPUs in the future. Energy awareness is of huge importance for these systems. There is also a huge potential of research in this field [RSR+08].

This section is based on the author's publication [LMS+14].

8.3.1 Methods

In the following, methods for a fast, multi-objective, energy-aware exploration of GPU architectures are introduced. GPU architectures of different sizes are simulated. The main challenge for the MOGEA-DSE method is to keep evaluation run time low as simulation of GPUs is time-consuming and the explored design space is usually not small. The task to keep the evaluation time low is closely related to the task of keeping the design space small without sacrificing accuracy in the objective function.

The structure of the section is as follows. First, an introduction to the architectural design of MOGEA-DSE is given that integrates remote evaluation. Then three methods for a faster evaluation are presented. The first method that simply reduces the input size will be shown to be not suitable for an accurate evaluation and will lead to the next two methods.

Architectural Design of MOGEA-DSE for Remote Evaluation

The general architectural design of MOGEA-DSE is extended to run locally or distributed on a wide variety of homogeneous or heterogeneous compute systems. The main target

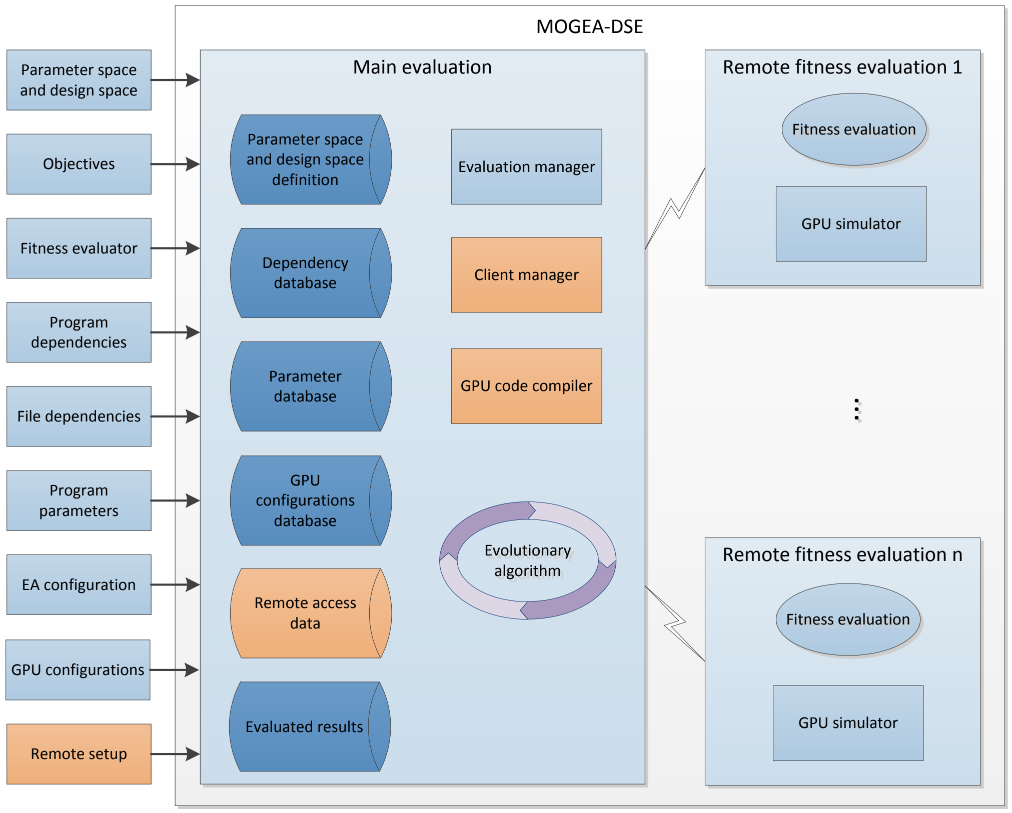


Figure 8.4: Distributed evaluation process. A master PC controls the EA and distributes evaluation tasks to the cloud nodes. On each node several evaluations can be calculated in parallel. Each evaluation is done, e.g., with the GPUSimPow GPU simulator.

systems for distributed computing are single desktop computers, a heterogeneous network of desktop computers, compute servers, and heterogeneous compute clouds.

In Figure 8.4 the changes in the architectural design are shown. Same as in SOG-DSE the evaluation manager is responsible for running the evaluation. However, in the MOGEA-DSE approach the evaluation manager runs on a master PC and can distribute the evaluations to the quality evaluations 1 to n, as shown in Figure 8.4. The quality evaluations can be done on the same local device or on remote devices. Each quality evaluation runs the given GPGPU program on the current configuration out of the design space. Every node in the network can evaluate numerous evaluations at the same time using multiple instances of the GPU simulator or by using actual GPU hardware. The GPGPU program can be executed on a GPU simulator. Afterward, the quality of the result is determined and transmitted back by the main evaluation.

As GPU simulators either GPGPU-Pow or GPGPU-Sim in combination with GPU-Wattch are used in MOGEA-DSE. The GPU simulators are only loosely coupled with the fitness evaluation. Therefore, GPGPU-Pow or GPGPU-Sim can be easily replaced by other simulators.

Reduction of the Input Size

A major problem for the simulation of complex GPU programs is the evaluation time. Especially for a large design space, the evaluation time of a single run is crucial because hundreds or even thousands of runs have to be performed to achieve good sampling. A typical solution to keep the evaluation time small would be to reduce the input size of the simulated GPU program. However, simply reducing the size of the input is not permitted or has to be done very carefully because the utilization of the GPU SMPs heavily depends on the input size. A good utilization of the SMPs is a crucial factor of how fast and energy efficient a GPU program runs. Simulating unrealistic utilization due to too small input sizes can have unwanted side effects on the evaluation results.

This can be shown with the following example. An image of size 1024 px × 1024 px should be processed on four GPUs with 12 to 15 SMPs. The pixels should be processable independently from each other. This is a very simple task and even for this task the input size is important. For the partitioning of the input image, a local work group size of, e.g., 16 × 16 can be considered. With this the image is partitioned in 4096 sub-images each of size 16 × 16. Each sub-image is then processed by on one SMP on the GPU.

The GPU with 15 SMPs needs 273 steps to process 15 sub-images, 4095 of the 4096 sub-images, with full utilization of the multiprocessors and then one last step for the one remaining sub-image with about 6.7 % utilization as 14 SMPs are idle. As a result, the overall utilization of the GPU is 99.7 %. This is a very good utilization of the GPU. If now the input size is simply decreased to speed up the evaluation, e.g., from 1024 px × 1024 px down to 128 px × 128 px, only 64 sub-images need to be processed. The GPU with 15 SMPs can be fully utilized for 4 steps and in the last step 11 of the 15 SMPs are active. The overall utilization of the GPU drops from 99.7 % to 85.3 %.

A similar reduction in the utilization can be observed for the other GPUs. For a GPU with 14 SMPs, the utilization drops from 99.9 % to 91.4 %. For a GPU with 13 SMPs, the utilization drops from 99.7 % to 98.5 %. And for a GPU with 12 SMPs the utilization drops from 99.8 % on the full input to 88.9 % on the reduced input. For the four considered GPUs, the one with 14 SMPs shows the best utilization on the full input image. However, on the reduced input the GPU with 13 SMPs shows the best utilization.

Not only the utilization changes but also the execution time. For the full input image, the four GPUs need 274, 293, 316, and 342 steps, and for the reduced input image the GPUs need 5, 5, 5, and 6 steps. If all four GPUs operate at the same clock speed and the memory operations are performed with the same speed, the execution time on the full input image is shorter for the GPUs with more SMPs but on the reduced input image it is identical for three of the four GPUs. In consequence, it is possible that, wrongly, a GPU with, e.g., 13 instead of 14 or 15 SMPs is identified as the best GPU.

In this example, only a very simple task, with no dependencies between the threads, has been considered. Each thread and each work group can work independently of each other. Even for this simple task, it is difficult to determine the influence of a changed input size. To determine for a given application a change of input size that has no effect on the result, is more difficult or even not possible.

Therefore, a reduction of the input size is not used in MOGEA-DSE. Instead, two approaches have been developed to circumvent the shown problems. The approaches are called *checkpointing* and *quick start* approach and are presented in the following.

The Checkpointing Approach

To reduce the evaluation time, the *checkpointing* approach can be used. This approach can handle initialization phases or calculations of programs that are time-consuming if simulated but that should not be measured anyway.

One example is the initialization of VirusDetectionCL. Some arrays and queues has to be initialized or filled, e.g., the queues for the noise reduction, before the actual virus detection can run.

A second example is the neural network program NN. If a neural network program is used, e.g., in an industrial process control, this program is initialized once and then repeatedly used to classify several requests. Here, also the initialization phase is of no interest for the overall energy consumption.

Usually, an initialization phase of an algorithm can not be skipped as this would affect the objectives that should be measured or even keeps the program from running properly. Especially, data dependencies on not properly initialized buffers can influence the run time in the remaining parts. In VirusDetectionCL the noise reduction influences how many viruses are found or missed and this influences the run time of other pipeline elements.

The *checkpointing* approach is used to reduce the evaluation time without influencing the program logic or the calculated results. It works as follows. In a preparation step all memory buffers at a pre-defined state are stored on the hard drive to create a checkpoint. This step can be done in one single GPU run, either on actual hardware or simulated hardware. During the simulation this pre-calculated memory state is restored. Afterward, the remaining data can be processed.

To use the *checkpointing* approach with VirusDetectionCL, two additional functions are used to store and restore all needed GPU memory buffers in VirusDetectionCL. To store the device memories, a host memory buffer of sufficient size is allocated. For every device memory buffer, the OpenCL function `clEnqueueReadBuffer` is used to download the buffer from device to the host memory. The host buffer is then stored in a binary file. Additional information about name of the buffer, buffer type, buffer dimensions, bit depth, number of bits per pixel, current frame number, byte endianness, and information about the data set are stored in a sidecar file. To load a checkpoint the information in the sidecar file is used to find out the types and sizes of the OpenCL device buffers. The stored binary files are loaded into host memory for each buffer. The OpenCL function `clCreateBuffer` is used to create the device memory of the correct type and size. By providing a pointer of the host memory to the OpenCL function, the host memory is used to initialize the device buffer. Additional error checks are done to make sure that the restored checkpoint matches the data set and the frame number.

As a result, the initialization phase of VirusDetectionCL, which usually needs to process dozens of frames, can be skipped and replaced by loading a few stored buffers with the *checkpointing* method.

Additionally, for streaming applications like VirusDetectionCL that consist of an initialization phase and then thousands of images are processed the same way, the *checkpointing* approach can be used to safely reduce the number of processed images. If the characteristic of the data stays the same after the initialization phase, only a reduced amount of data has to be processed to extrapolate the execution time and energy consumption to process the whole data. For VirusDetectionCL, e.g., only a few hundred frames of a data set of 4000 frames need to be processed to receive accurate results.

One limitation of this method is that the restoring introduces overhead that has to be considered in the measurement. The restoring has to be excluded from the measurements or the measurements of the restoring have to be subtracted from the overall measurements afterward.

To sum up, the *checkpointing* approach can be used to reduce the evaluation time for many types of programs. This method is especially useful if data dependencies exist that require perfectly initialized buffers. For the VirusDetectionCL use case, e.g., the first 100 images do not have to be processed. Instead, the initialized queues and buffers can be loaded.

The Quick Start Approach

To even further reduce the evaluation time, the *checkpointing* approach is extended to the so called *quick start* approach. This approach can initialize memory buffers very fast by using approximation and making use of the queue structure of some buffers.

The fastest possible initialization of memory buffers is to not initialize them at all. If it is known that some parts of the GPU program do not depend on the actual data in the buffers, these buffers can simply be kept uninitialized. Then they just contain random values. For example, the execution time of a filter kernel applied to an image usually does not depend on the actual content of the image. If this is the case, the calculation can be done on random values in the memory without effecting the execution time or energy consumption. The evaluation time for the *checkpointing* step or a full initialization can be saved.

However, the execution time of, e.g., a sorting algorithm usually strongly depends on the content that should be sorted. Also, it is necessary to ensure that no later pipeline element depends on the data in the buffer that should be kept uninitialized. Therefore, most applications do not allow a random initialization of the buffers.

These considerations lead to the *quick start* approach. With the *quick start* approach, a fast initialization is possible even in the presence of data dependencies. The *quick start* approach has been tailored for the VirusDetectionCL program but can also be used for other programs.

The approach works as follows. For VirusDetectionCL buffers can not be kept uninitialized as the segmentation and classification would not work on random data. For the PAMONO sensor images, it can be assumed that they are constant if no viruses appear and if the sensor noise is ignored. This is done by replicating the first image and use this image for the initialization of the queues of the noise reduction queues. By using the first noise reduced image for the initialization of the signal model queues, all pipeline stages can be initialized with only a single image. The replication of the image is done within the OpenCL kernels and not for the pipeline input. As a result, for every OpenCL kernel in the pipeline only a single kernel call is needed. With the *quick start* approach, the necessary initialization can be reduced, e.g., from processing 100 frames down to processing only one frame. With larger queue sizes the savings is even higher. Because this method can produce some boundary artifacts, e.g., by darker areas in the first image, the results from detection and classification are calculated but ignored for a few frames.

A major drawback of the *quick start* method is that it requires modifications in the OpenCL kernels, which is not always possible. Also, some kind of queue-based data structure is needed to benefit from the *quick start* method.

8.3.2 Experiments

The method has been evaluated with a wide variety of industrial, biological, and physical applications for different scenarios. Three experiments are conducted for desktop, mobile, and HPC target architectures and named accordingly: $\text{Exp6}_{\text{Desktop}}$, $\text{Exp7}_{\text{Mobile}}$, and Exp8_{HPC}.

The evaluated programs are shown in Table 8.1. VirusDetectionCL is evaluated on all target architectures. As data set for VirusDetectionCL $\text{Ds7}_{\text{Poly200nm-1}}$ is used, cf. Section 2.2. In addition, programs of the following three benchmark suites are evaluated to show that the MOGEA-DSE method generalizes. The programs make use of both OpenCL and CUDA as programming languages and cover a wide variety of industrial, physical or biological applications like AES encryption, denoising of radar data, solving systems of linear equations, simulation of temperature distributions on processors or ray tracing simulations.

The programs are matched to the three scenarios as close as possible. For the desktop scenario in $\text{Exp6}_{\text{Desktop}}$, programs from the NVIDIA®CUDA SDK benchmark [NVI15a] are used. For the mobile scenario in $\text{Exp7}_{\text{Mobile}}$, some lightweight benchmark programs from GPGPU-Sim [BYF+09] are used. And for the HPC scenario in Exp8_{HPC}, the programs from the demanding Rodinia benchmark [CBM+09] are used.

For VirusDetectionCL the following scenarios from Figure 8.1 are covered. The laboratory scenario Scn1_{Lab}, the desktop and laptop devices from scenario $\text{Scn3}_{\text{Local}}$, the laptop and server devices of the cloud computing scenario $\text{Scn4}_{\text{Cloud}}$, and the laptop, desktop and server devices of the hospital or airport scenario $\text{Scn5}_{\text{Hospital}}$.

The experiment $\text{Exp6}_{\text{Desktop}}$ covers a desktop scenario where different desktop GPUs are used as target architecture. For VirusDetectionCL this covers Scn1_{Lab}, $\text{Scn3}_{\text{Local}}$, and $\text{Scn5}_{\text{Hospital}}$. The experiment $\text{Exp7}_{\text{Mobile}}$ covers a mobile scenario where different laptop GPUs are the target architecture. This covers Scn1_{Lab}, $\text{Scn3}_{\text{Local}}$, $\text{Scn4}_{\text{Cloud}}$ and $\text{Scn5}_{\text{Hospital}}$. The experiment Exp8_{HPC} covers a high performance scenario where different HPC GPUs are the target architecture. This covers $\text{Scn4}_{\text{Cloud}}$, and $\text{Scn5}_{\text{Hospital}}$.

Table 8.2 shows the three sets of evaluated GPU architectures for the three experiments. For $\text{Exp6}_{\text{Desktop}}$ four Fermi™architectures, for $\text{Exp7}_{\text{Mobile}}$ three mobile Fermi™architectures, and for Exp8_{HPC} four devised high performance architectures. The high performance architectures were announced by NVIDIA®but they did not exist while conducting the experiments and no specifications were available. The idea of selecting these as target architectures is that it can be beneficial to know in advance which GPU is best suited for the demands. For example if an HPC server system should be equipped with GPUs in the future, a preliminary DSE can be performed to find out if it is beneficial to delay a purchase until the identified best suited GPU is available or to produce early cost estimates of the target system. Also, bottlenecks in the software design can be identified and remedied early.

The three sets of architectures from Table 8.2 are explained the following. For $\text{Exp6}_{\text{Desktop}}$ the Fermi™architectures GF-108, GF-106, GF-104, and GF-100 are evaluated.

Table 8.1: Evaluated programs for the multi-objective optimization from benchmark suits CUDA-SDK, GPGPU-Sim, and Rodinia. Adapted from [LMS+14].

Suite	Program	Language	Description
none	VirusDetectionCL	OpenCL	Real-time computer vision application on biological sensor data, see Chapter 5
CUDA-SDK	blackScholes	OpenCL	Price prediction of European options
CUDA-SDK	histogram	OpenCL	Histogram calculation
CUDA-SDK	quasiRandomGen	OpenCL	Random number generator
CUDA-SDK	reduction	OpenCL	A parallel sum reduction on large arrays
CUDA-SDK	matrixMul	OpenCL	Parallel matrix multiplication
CUDA-SDK	mersenneTwister	OpenCL	Random number generator
CUDA-SDK	vectorAdd	OpenCL	Parallel vector addition
GPGPU-Sim	AES	CUDA	AES encryption algorithm
GPGPU-Sim	BFS	CUDA	A breadth first search algorithm for graph traversal
GPGPU-Sim	CP	CUDA	Calculation of the Coulomb potential physics simulation
GPGPU-Sim	LPS	CUDA	Laplace discretization on a 3D structured grid
GPGPU-Sim	NN	CUDA	A Neural Network on GPU
GPGPU-Sim	NQU	CUDA	N-Queens Solver
GPGPU-Sim	RAY	CUDA	Raytracing simulation for rendering of light effects
GPGPU-Sim	STO	CUDA	MD5 hash calculation
Rodinia	backProp	OpenCL	Back propagation algorithm in neural networks
Rodinia	hotSpot	OpenCL	A temperature physics simulation for processors
Rodinia	gaussian	OpenCL	Gaussian elimination for solving systems of linear equations
Rodinia	nn	OpenCL	Nearest neighbor location calculation on hurricane data
Rodinia	nw	OpenCL	A nonlinear global optimization on DNA sequences with the Needleman-Wunsch algorithm

Table 8.2: Evaluated GPU architectures for the experiments Exp6$_{\text{Desktop}}$, Exp7$_{\text{Mobile}}$, and Exp8$_{\text{HPC}}$. For each architecture the number of SMPs, number of SPs, core clock rate, DRAM type, and DRAM clock rate are listed. Columns for number of TMUs, ROPs, and shader clock rate are shown in Table A.2. Adapted from [LMS+14].

	GPU architecture	SMPs	SPs	Core clock	DRAM	DRAM clock
Desktop	Fermi GF-108	1-2	48,96	700-810 MHz	GDDR-3	400-450 MHz
	Fermi GF-106	3-4	144,192	590-790 MHz	GDDR-5	450-1000 MHz
	Fermi GF-104	6-7	288,336	650-675 MHz	GDDR-5	850-950 MHz
	Fermi GF-100	11-15	352-480	610-780 MHz	GDDR-5	800-1000 MHz
Mobile	Fermi GF-108M	1-2	48,96	600-740 MHz	GDDR-3	800-900 MHz
	Fermi GF-116M	3-4	144,192	590-775 MHz	GDDR-5	900-1250 MHz
	Fermi GF-114M	7-8	336,384	575-620 MHz	GDDR-5	1500 MHz
HPC	Pascal GP-107^1	4-5	1024,1280	1200 MHz	GDDR-5	1250-1450 MHz
	Pascal GP-104^1	6-8	2304-3072	820-1040 MHz	GDDR-6	1500-1750 MHz
	Pascal GP-110a^1	12-15	4608-5760	860-880 MHz	GDDR-6	1500-1750 MHz
	Pascal GP-110b^1	14-15	5376-5760	840-890 MHz	GDDR-6	1500-1750 MHz

^{1} Devised architectures that did not exist while conducting the experiments.

The use case for these architectures are typical desktop applications for example the VirusDetectionCL program in a stationary laboratory or hospital setup as shown in Figure 8.1.

For Exp7$_{\text{Mobile}}$ the mobile Fermi™architectures GF-108M, GF-116M, and GF-114M are evaluated. The use case for these architectures is an independent, mobile, battery driven processing. Several runs of the same program are performed on a mobile device. I.e., the evaluation of several PAMONO samples with VirusDetectionCL on a laptop as shown in Figure 8.1.

For Exp8$_{\text{HPC}}$ the high performance the devised Pascal™architectures GP-107, GP-104, GP-110a, and GP-110b are evaluated. As already mentioned, the actual Pascal™architectures did not exist during the performance of the experiments. A typical scenario for these architectures is the server side of the cloud computing, hospital, or airport scenario in Figure 8.1. Several devices are connected to the server and a few fixed applications are executed on the server multiple times. The Pascal™architecture shows what a new high performance architecture might look like in the near future. All specifications are derived, no specifications were available from NVIDIA®during performance of the experiments. It is assumed that the number of available SPs, ROPs and TMUs will double, compared to today's Kepler™architectures GK-107, GK-104, and GK-110. Also, it is assumed that the current GDDR-5 RAM will be replaced by a GDDR-6 RAM with doubled memory bandwidth. However, the modeling of the new architecture has been kept simple as only a few changes have been made compared to the Fermi™architecture. For example, the Pascal™architecture may be equipped with HBM2 DRAM, which is 3D stacked memory, and a unified virtual memory link. This has only been modeled with a doubled memory bandwidth, which is not very accurate. Therefore, the experiments with these derived architectures should only be seen as a proof of concept. If more reliable results are needed, more effort has to be spend in the modeling of new architectures.

As explained in Section 8.2.2, not all GPU parameters listed in Table 8.2 need to be coded in genes in order to optimize these. Instead, most of them can be modeled as dependencies. This has the two advantages that the design space is kept small and no invalid configurations are generated.

For example, for the FERMI™architectures GF-108, GF-106, and GF-104 architectures only three parameters need to be coded in genes: the number of SMPs, the core clock, and the DRAM clock. All other parameters can be derived unambiguously.

The evaluation has been performed on a compute sever with four AMD OPTERON™6272 CPUs and 256 GB RAM. Each of the OPTERON™CPUs has 16 cores, resulting in 64 cores in total. Because the GPUs are simulated, the compute server does not need to have GPU hardware installed. As operating system LINUX®UBUNTU™12.10 for servers is used, which is executed in a KVM virtual machine. SPEA-2 [ZLT01] for the multi-objective evaluation is configured with a vector mutation pipeline with mutation probability 0.01 and a tournament selection of size two.

8.3.3 Results

In this section results for MOGEA-DSE are presented for a multi-objective energy-aware hardware DSE. The results are divided into three parts: First, results for the experiments EXP6$_{\text{DESKTOP}}$, EXP7$_{\text{MOBILE}}$, and EXP8$_{\text{HPC}}$. Second, results for the *checkpointing* approach. Third results for the *quick start* approach.

Results for the Conducted Experiments

The evaluation of MOGEA-DSE has been done for a variety of GPU architectures and a variety of GPGPU programs from different benchmark suites with focus on the evaluation of the VirusDetectionCL use case. Besides VirusDetectionCL 20 other programs have been evaluated for the FERMI™, mobile FERMI™, and PASCAL™target architectures. The evaluation time for a single GPU program depended on the complexity of the program and varied from 20 minutes to five days. The average evaluation time over all programs was 19.8 hours.

The following results are structured according to the three experiments EXP6$_{\text{DESKTOP}}$, EXP7$_{\text{MOBILE}}$, and EXP8$_{\text{HPC}}$. As VirusDetectionCL is the main use case, the results for VirusDetectionCL are particularly highlighted and presented separately of the three experiments.

To provide an overview, in Figure 8.5 Pareto fronts for the energy-aware, multi-objective evaluation of different programs and for different target architectures are shown for the three experiments. The evaluation for VirusDetectionCL is shown in more detail in Figure 8.5a. Figure 8.5b, Figure 8.5c, and Figure 8.5d shows the evaluations for EXP6$_{\text{DESKTOP}}$, EXP7$_{\text{MOBILE}}$, and EXP8$_{\text{HPC}}$. Figure 8.6 shows the core clock rate and the number of SMPs of the identified GPU configurations in an overview for all experiments and then structured by the experiments. Finally, Table 8.3 shows the identified GPU architectures, number of SPs, and core clock rate for the Pareto fronts of every program. The different architectures and specific GPU configurations for these architectures are given as comma separated entries. An extended version of this table with columns for SMPs and DRAM can be found in Table A.3.

The Pareto fronts for the VirusDetectionCL use case are shown in Figure 8.5a and the corresponding configurations are shown in Figure 8.6a and Table 8.3. VirusDetectionCL has been evaluated with a reduced pipeline configuration against all architectures, indicated by (1) to (3) in the figure and table, and also in the most complex pipeline configuration against the high performance architecture, indicated by (4).

In Figure 8.5a four Pareto fronts for VirusDetectionCL with objectives execution time per frame and overall energy consumption are given. The GF-1xx FERMI™architecture, indicated by (1), is the best choice with respect to energy consumption. As expected, the GP-1xx high performance architecture, indicated by (3), is the best choice with respect to the execution time. A somehow surprising result is that the mobile FERMI™architecture, indicated by (2), is dominated in both objectives by the FERMI™architecture. In the most complex pipeline configuration and the GP-1xx high performance architecture, indicated by (4), the lowest overall energy consumption is 41 J with an execution time of 4.4 ms per frame. The fastest execution time was achieved with 1.6 ms per frame with an overall energy consumption of 170 J. Overall, the GF-100 FERMI™architecture could be identified as a good compromise between energy consumption and execution time. In detail, the GEFORCE™GTX 465, 470 and 480 were identified as actual GPUs of the GF-100 FERMI™architecture, with the GTX 465 as the most energy efficient GPU. Only if the fastest run time is needed a high performance architecture has to be chosen.

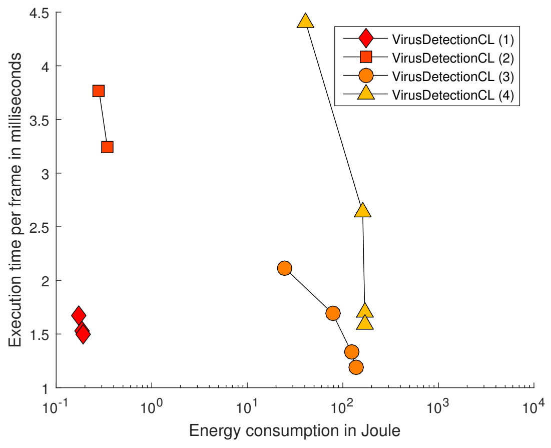
For the experiment EXP6$_{\text{DESKTOP}}$, seven programs are evaluated on the four FERMI™architectures GF-108, GF-106, GF-104, and GF-100. The Pareto fronts and the dominated points are shown in Figure 8.5b. Because the execution time and energy consumption are in a wide range, the Pareto fronts in the figure are only visible for some programs. Therefore, the Pareto fronts are also shown in Table 8.3. The corresponding configurations are visualized in Figure 8.6b. The execution time of the evaluated programs is in a range of 0.015 ms to 77.82 ms and the energy consumption is in a range of 0.0018 J to 9.33 J.

For all seven programs the GF-100 dominates the architectures GF-108, GF-106, and GF-104. The actual configuration of GF-100 varied from eleven to fifteen SMPs, with a core clock rate from 630 MHz to 780 MHz.

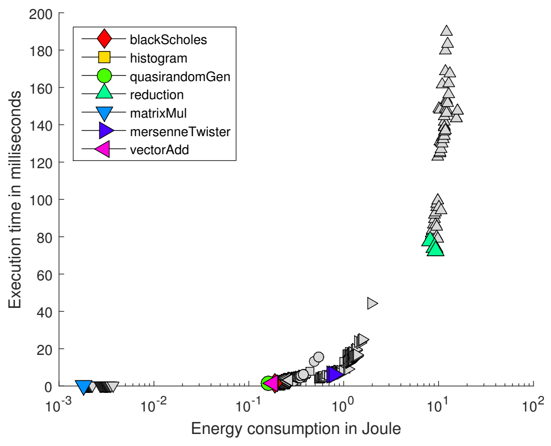
A few trade offs could be identified. For the random number generator MERSENNETWISTER, the energy consumption can be reduced by 7 % with only a slight increase in execution time of 0.2 % by selecting the according point on the front. For the parallel sum reduction on large arrays REDUCTION, three points are on the Pareto front. The most energy efficient solution uses 12 % less energy while spending 7 % more in execution time than the fastest solution. Generally, the differences are small as the GF-100 is the only architecture on the front and only the actual configurations of this target architecture differ.

For the experiment EXP7$_{\text{MOBILE}}$, eight programs were evaluated for the mobile FERMI™architectures GF-108M, GF-116M, and GF-114M. The Pareto fronts are shown in Figure 8.5c and the configurations are shown in Figure 8.6c and Table 8.3. The execution time of the evaluated programs is in a range of 0.006 ms to 8.84 ms and the energy consumption is in a range of 0.000 14 J to 0.67 J.

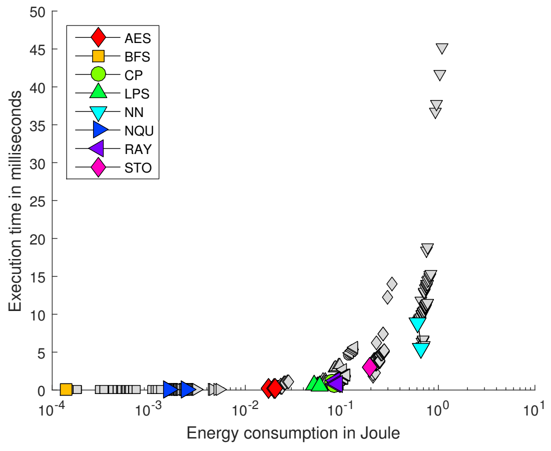
Overall, a wide variety of architectures has been identified with one to eight SMPs. The core clock rate is in a range from 615 MHz to 770 MHz. In contrast to the experiment



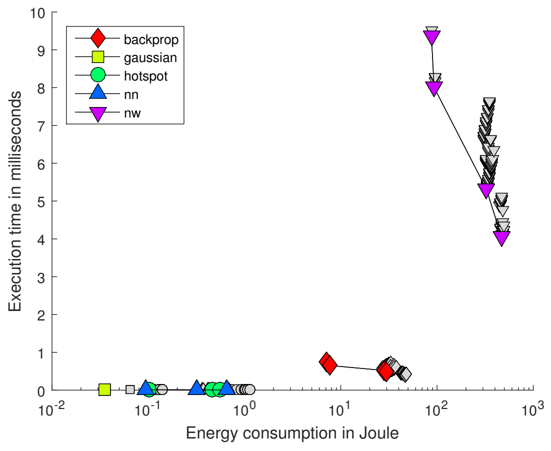
(a) Tradeoff between energy consumption and execution time. Detailed view of four Pareto fronts for the VirusDetectionCL program. The symbols indicate different target configurations.



(b) Tradeoff between energy consumption and execution time for EXP6_{DESKTOP} with target GPU architectures FERMI™ GF-100, GF-104, GF-106, and GF-108. Points from different Pareto fronts are shown. The symbols indicate different programs.

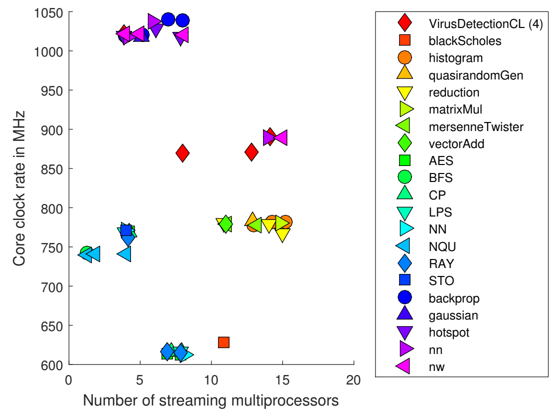


(c) Tradeoff between energy consumption and execution time for EXP7_{MOBILE} with target mobile GPU architectures FERMI™ GF-114M, GF-116M, and GF-108M. Points from different Pareto fronts are shown. The symbols indicate different programs.

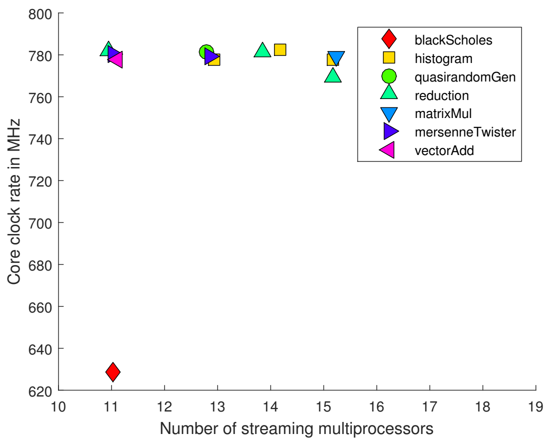


(d) Tradeoff between energy consumption and execution time for EXP8_{HPC} with target GPU architectures PASCAL™ GP-110a, GP-110b, GP-104, and GF-107. The GPUs are derived and did not exist during performance of the experiments. Points from different Pareto fronts are shown. The symbols indicate different programs.

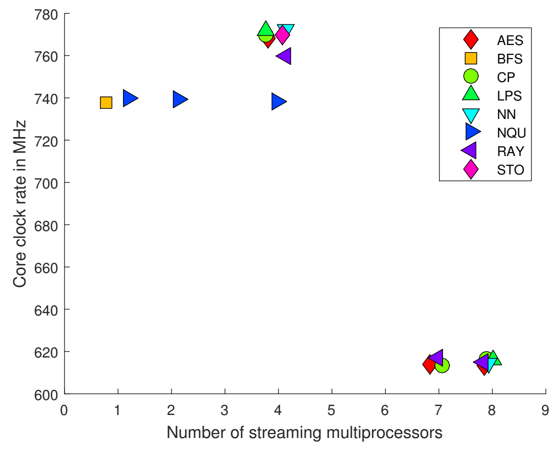
Figure 8.5: Energy-aware multi-objective evaluation for different programs, cf. Table 8.1, and for different GPUs, cf. Table 8.2. For every program a Pareto front of non-dominated points is shown. Dominated points are plotted smaller and without color. Adapted from [LMS+14].



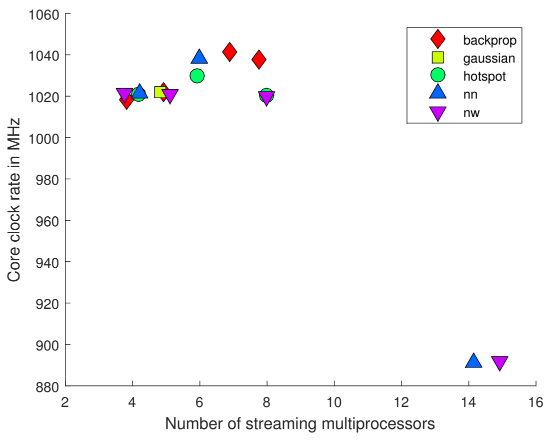
(a) Best GPU configurations for all evaluated programs of their associated target architecture.



(b) GPU configurations for the FERMI™ GPUs GF-100, GF-104, GF-106, and GF-108.



(c) GPU configurations for the FERMI™ GPUs GF-114M, GF-116M, and GF-108M.



(d) GPU configurations for the PASCAL™ GPUs GP-110a, GP-110b, GP-104, and GF-107.

Figure 8.6: The best GPU configurations with respect to the objectives energy and execution time. Integer values are slightly jittered to separate the points. See also Table 8.3. Adapted from [LMS+14].

with the FERMI™ architectures, where the GF-100 is the only architecture on the front, here every architecture is at least on one of the fronts.

For the neural network program NN, the fastest solution saves 37 % execution time with an increase in energy consumption of 8 %. For the ray tracing program RAY, the fastest solution also saves 37 % execution time while at the same time the energy consumption is increased by 7 % if the GF-114M architecture is used instead of the GF-116M. For the CP program, that calculates Coulomb potentials similar values could be found. A saving of 37 % execution time could be traded against an increase of 6 % in energy consumption. For the Laplace discretization of grids program LPS, a saving of 30 % execution time results in an increase of 2 % in energy consumption if GF-114M architecture is used instead of the GF-116M.

Table 8.3: The best GPU configurations for every program. Columns for number of SMPs and DRAM are shown in Table A.3. Extended version of [LMS+14].

Program	GPU architectures	SPs	Core clock in MHz
VirusDetectionCL (1)	GF-110	352,448,480	780,780,780
VirusDetectionCL (2)	GF-114M	192,336	760,590
VirusDetectionCL (3)	GP-107^1,104^1,110b^1	1280,2304,4608,5760	1020,1040,870,890
VirusDetectionCL (4)	GP-107^1,104^1,110b^1	1024,3072,4992,5376	1020,870,870,890
blackScholes	GF-100	352	630
histogram	GF-100	416,448,480	780,780,780
quasiRandomGen	GF-100	416	780
reduction	GF-100	352,448,480	780,780,770
matrixMul	GF-100	480	780
mersenneTwister	GF-100	352,416	780,780
vectorAdd	GF-100	352	780
AES	GF-116M,114M	192,336,384	770,615,615
BFS	GF-108M	48	740
CP	GF-116M,114M	192,336,384	770,615,615
LPS	GF-116M,114M	192,384	770,615
NN	GF-116M,114M	192,384	770,615
NQU	GF-108M,116M	48,96,192	740,740,740
RAY	GF-116M,114M	192,336,384	760,615,615
STO	GF-116M	192	770
backProp	GP-107^1	1024,1280	1020,1020
	GP-104^1	2688,3072	1040,1040
gaussian	GP-107^1	1280	1020
hotSpot	GP-107^1,104^1	1024,2304,3072	1020,1030,1020
nn	GP-107^1,104^1,110b^1	1024,2304,5376	1020,1040,890
nw	GP-107^1	1024,1280	1020,1020
	GP-104^1,110b^1	3072,5760	1020,890

^{1} Devised architectures that did not exist during performance of the experiments.

Finally, for the experiment Exp8$_{\text{HPC}}$, five programs were evaluated on the high performance Pascal™architectures GP-107, GP-104, GP-110a, and GP-110b. It should be emphasized that the results for these evaluations have to be interpreted very carefully. As explained in Section 8.3.2, the architectures are derived from the Fermi™architectures with marginal knowledge of the actual target architectures. As a result, the absolute values are not supposed to be accurate and the evaluation of these architectures is provided as a proof of concept. Also, the kernel execution times are too short to give reliable results for the absolute energy consumption. However, the relative proportions and the identified target architectures still yield valuable insights.

The Pareto fronts are shown in Figure 8.5d and the configurations are shown in Figure 8.6d and Table 8.3. The execution time of the evaluated programs is in a range of 0.003 ms to 9.38 ms and the energy consumption is in a range of 0.0349 J to 466.9 J.

In contrast to the FERMI™architecture, here again the Pareto front contains a wide variety of architectures. The number of SMPs is in a range from four to fifteen. The core clock rate is 1020 MHz to 1040 MHz for most of the programs with an outlier for NW with 890 MHz. For all programs the GP-107 architecture was part of the Pareto fronts and for some programs additionally GP-104 and for the NW program also GP-110b. Overall, the programs NW and VirusDetectionCL are the only ones with the high performance GP-110 as target architecture. Several interesting trade offs could be identified. For the BACKPROP tool for neural networks, three points are on the front. For the most energy efficient solution, an increase in execution time by 42 % saves 74 % energy.

For the Needleman-Wunsch optimization of DNA sequences with the NW tool, the points on the front are far apart, the fastest solution with GP-110b as target architecture takes 4.06 ms and consumes 466.9 J and the most energy efficient solution with GP-107 as target architecture takes 9.38 ms and consumes 87.43 J. As a result, an increase in execution time by 131 % saves 81 % energy. For NN the nearest neighbor location on hurricane data, the fastest solution with GP-110b as target architecture takes 0.006 ms and consumes 0.64 J and the most energy efficient solution with GP-107 as target architecture takes 0.01 ms and consumes 0.09 J. The absolute savings of a single run are negligible but if the application should be executed several million times on an HPC server, a speedup of 1.67 or 86 % saved energy does make a difference.

Results for the Checkpointing and Quick Start Approach

Three different OpenCL kernels GAUSS FILTER, PRE-PROCESSING, and SLIDING MEDIAN from the VirusDetectionCL program have been inspected. The *checkpointing* approach has been evaluated by predicting the energy consumption for 50 frames based on the processing of 1, 5, 10, and 20 frames. For GAUSS FILTER the simulated energy consumption is 14.26 J and the prediction is 14.26 J with 0.0006 % relative error. For PRE-PROCESSING 0.159 J have been simulated and 0.162 J are predicted with a relative error of 1.37 %. Finally, SLIDING MEDIAN took 3.188 J in the simulation while 3.187 J are predicted. As a result the relative error is 0.047 %. The evaluation time could be reduced from 31 h down to 6 h.

The *quick start* approach has been evaluated by predicting the energy consumption of processing 80 frames based on the processing of 1, 5, 10, 20, and 40. For the first kernel, GAUSS FILTER, the actual simulated energy consumption is 14.28 J for 80 frames. The average predicted energy consumption is 14.26 J with a relative error of 0.019 %. Even if only one instead of 80 frames is processed, the relative error is merely 0.02 %. For the second kernel, PRE-PROCESSING, the simulated energy consumption is 0.159 J and the average predicted energy consumption is 0.157 a relative error of 1.37 %. If only one frame is processed, the relative error is 3.6 %. Errors can be reduced further by processing more frames. The third kernel, the SLIDING MEDIAN calculation, has heavy data dependencies and is therefore not well suited for the *quick start* approach. The simulated energy consumption is 2.37 J and the predicted energy 0.88 J. This results in a relative error of 62.8 %.

8.3.4 Discussion

With the conducted experiments all research issues raised in this section could be answered. It has been shown how MOGEA-DSE can be used to identify GPU hardware for desktop, mobile, and HPC systems for various programs.

The most important advantage of simulation of GPU architectures could be shown: GPUs that need to evaluated do not need to be bought and tested and do not even need to exist. Suitable GPUs can be identified early in the design phase of a system. Additionally, possible bottlenecks in the software can be identified, e.g., if a system should be equipped with the next generation of upcoming GPUs.

One important feature of MOGEA-DSE is that the explored design space can be kept small. This could be achieved by making use of parameter dependencies, which led to a reduced evaluation time without losing accuracy in the objective functions.

In the following a detailed discussion for the conducted experiments, the devised architectures, and for the *checkpointing* and *quick start* approach is given.

Discussion of the Conducted Experiments

The multi-objective evaluation has shown interesting results. For most evaluated programs not one but up to four GPU architectures could be identified to be on the Pareto front for the two objectives energy consumption and execution time.

Some architectures tend to be more efficient for the objectives energy consumption and execution time than others. These architectures include the GF-100 for the FERMI™architectures, the GF-116M and the GF-114M for the mobile FERMI™architectures, and the GP-107, GP-104, and the GP-110 for the PASCAL™architectures.

A very surprising result is that the GF-100 dominated all other FERMI™architectures. For the mobile FERMI™architecture, the GF-116M or GF-114M is best for most of the programs. And for the PASCAL™architecture, the GP-107 was always one solution in the Pareto fronts.

Another interesting result is that the high performance GP-110 architecture with up to 5760 SPs has been only for NN, NW and VirusDetectionCL on the Pareto front. This indicates clearly that all other programs, evaluated for the PASCAL™architecture, can not utilize the increased number of cores of the GP-110 architecture. As the GP-107 and GP-104 architecture can be operated at a higher clock speed, it is not beneficial to use the GP-110 architecture with more cores running at lower speed for programs that can not utilize the additional cores. Here, less cores running at a higher clock rate outperform the more powerful architecture. For the energy consumption it is also not beneficial to integrate more cores if they are not utilized.

Discussion on the Devised Architectures

During the performance of the experiment EXP8$_{\text{HPC}}$, the inspected PASCAL™architectures did not exist. They had been announced by NVIDIA®but no specifications was available. Meanwhile, NVIDIA®released eight GPUs based on the PASCAL™architecture [NVI16a] namely the GTX 1050, the GTX 1050 Ti, the GTX 1060, the GTX 1070, the GTX 1080, a new version of the Titan X, and the TESLA®P-100. This opens up the opportunity, to compare the devised architectures in Table 8.2 with the actual GPUs as provided in the following.

The GTX 1050 Ti is based on the GP-107 architecture. It is equipped with 6 SMPs, 768 SPs, and GDDR-5 memory [NVI16a]. It runs with a core clock rate of 1392 MHz. The GTX 1050 is also based on the GP-107 architecture but uses only 5 SMPs and is clocked slightly faster with 1455 MHz. The devised GP-107 architecture in Table 8.2 has been assumed to be more powerful than the actual architecture. The main difference is that for the devised architecture each SMP has been assumed to have 256 SPs, whereas the actual architecture uses 128 SPs per SMP. The DRAM type has been predicted correctly, and the core clock speed is slightly higher on the actual GPUs. Thus, the GP-107 would match more closely to a GP-108 architecture in the table, which had not been considered.

The GTX 1060 is based on the GP-106 architecture. It is equipped with 10 SMPs, 1280 SPs, and GDDR-5 memory [NVI16a]. It runs with a core clock rate of 1506 MHz and a DRAM clock of 2000 MHz. The derived GP-107 architecture in Table 8.2 matches more closely to the actual GP-106 architecture than to the actual GP-107. Overall, the actual architectures cover a wider range than the derived architectures. Therefore, the GP-106 architecture has been compared to the derived GP-107 in the following. Both the number of SPs and the DRAM type have been predicted correctly. For the devised architecture only 5 SMPs, each with 256 SPs, have been assumed, whereas the actual architecture uses 10 SMPs, each with 128 SPs. The overall number of SPs is the same. As the local memory and the warp scheduler can be kept smaller it is probably cheaper to build more less powerful SMPs than fewer more powerful. For the clock rate the GTX 1060 runs at 1506 MHz whereas for the derived architecture 1200 MHz has been assumed as the maximum core clock. Similarly, the DRAM clock rate has been assumed too low with 1450 MHz as maximum.

The GTX 1070 is based on the GP-104 architecture. It is equipped with 15 SMPs, 1920 SPs, and GDDR-5 memory [NVI16a]. It runs with a core clock rate of 1506 MHz and a DRAM clock of 2000 MHz. The GTX 1080, also based on the GP-104 architecture, is equipped with 20 SMPs, 2560 SPs, and GDDR-5X memory. It runs with a core clock rate of 1607 MHz and a DRAM clock of 2500 MHz. The derived GP-104 architecture uses 6 to 8 SMPs with 2304 to 3072 SPs. The SPs match for the GTX 1080. The GTX 1070 is in between the derived GP-107 and GP-104 architecture. This holds also for the DRAM type as the GTX 1070 uses GDDR-5 memory as the derived GP-107 architecture whereas the GTX 1080 uses faster GDDR-5X memory. The clock rates were underestimated for both GPUs and for core clock and DRAM clock.

The Titan X is based on the GP-102 architecture. It is equipped with 28 SMPs, 3584 SPs, and GDDR-5X memory [NVI16a]. It runs with a core clock rate of 1417 MHz and a DRAM clock of 2500 MHz. The actual GP-102 architecture is in between the derived GP-104 architecture and the derived GP-110a architecture.

Finally, the TESLA®P-100 [NVI16b] is based on the GP-100 architecture. It is equipped with 56 SMPs, 3584 SPs, with a core clock rate of 1126 MHz, and HBM2 DRAM high bandwidth memory. This GPU is specifically designed for HPC. Therefore, the TESLA®P-100 has additionally 1792 SPs for double precision floating point arithmetic. For this GPU architecture, which is named GP-110a and GP-110b in the table, 4608 to 5760 SPs were assumed, which is higher than for the actual GPU. But as the TESLA®P-100 only uses 56 SMPs of the 60 possible SMPs and also has additionally double processing SPs the maximum overall number of SPs of the GP-100 architecture is with 5760 equal to the assumed maximum. Similar to the other architectures fewer SMPs were assumed each with more SPs. The core clock rate of 1126 MHz was assumed as 840 MHz to 890 MHz. Overall, the TESLA®P-100 is the only out of the five GPUs with the stacked 3D memory, named HBM2, which has been assumed for the GP-104, GP-102, and GP-110 architecture.

To conclude the discussion of the actual architectures in comparison with the devised architectures, the devised architectures have been fairly accurate to the actual architectures. Most insights that were gained, e.g., that not all programs can utilize the high number of SPs, still hold for the actual GPUs. If however insights were gained that are dependent of the number of actual SMPs, for example work group sizes, this would have to be evaluated again if more information about the desired GPU architecture is available.

Discussion of the Checkpointing and Quick Start Approach

If the *checkpointing* or *quick start* approach is useful to reduce the evaluation time depends on whether the OpenCL programs has data dependencies or not. For programs without data dependencies, the *quick start* approach is well suited to reduce the evaluation time. For example, the energy consumption was predicted accurately for GAUSS FILTER and PRE-PROCESSING with good error rates of 0.019 % and 1.37 %.

For programs with data dependencies, the *quick start* approach is usually unsuitable. For these programs the *checkpointing* approach is well suited. For example, the SLIDING MEDIAN calculation exhibits heavy data dependencies. As a consequence, realistic input data is needed. Therefore, an initialization with random data as in the *quick start* approach is not sufficient. This is reflected particularly in the error rate of 62 %. For this type of programs, the *checkpointing* approach is perfectly suited. The computation can be done on initialized buffers, which are loaded from the hard disk, without the initialization overhead. As a result, the relative error dropped from 62 % to 0.047 %. The evaluation time was reduced by the *checkpointing* approach from one day and seven hours down to six hours and with a very low error.

8.3.5 Summary and Conclusion

In this section important methods for MOGEA-DSE have been developed. The contributions are the integration of remote evaluation, the *checkpointing* approach, and the *quick start* approach. All three contributions address the problem that simulation of GPUs is time-consuming and how a PSE and DSE method can be accelerated without losing accuracy in the result.

It could be shown that MOGEA-DSE can be used to explore several programs, several GPU architectures, and for target systems of different size. Particularly, MOGEA-DSE has been used for the exploration of 21 programs and 11 GPU architectures with mobile, desktop, and HPC systems as target systems. Additionally, with the GPU simulator, GPUs that do not yet exist could be evaluated. This is useful to identify suitable hardware without the need to buy several GPUs for testing, which is both time-consuming and expensive. Also, bottlenecks can be identified early making an optimization of software toward upcoming GPUs possible.

Regarding the *checkpointing* and *quick start* method, it has been shown that a reduction of the input result can lead to wrong results and should be done very carefully. With the *checkpointing* method buffers can be initialized faster without affecting the results by loading a checkpoint with pre-initialized buffers. With the *quick start* method, which is not suitable for all programs, an even faster initialization can be achieved by using approximation and making use of available pipeline structure. The approximation in *quick start* has only small effects on the result.

8.4 Multi-Objective, Offloading-Capable Hybrid PSE and DSE of Simulated GPUs for Mobile Devices

One use case of the PAMONO sensor is a rapid measurement of spreading of diseases. In case of a break out of a new disease, a fast, mobile virus detection can be crucial to detect the origin of the outbreak and to prevent further spreading. PAMONO devices can be used as an early warning system for epidemics. A CPS with a mobile PAMONO sensor, the real-time capable VirusDetectionCL method, and a mobile processing hardware like a laptop would be very well suited for such a task.

If several PAMONO devices cooperate they can share information like spreading of viruses in the current area or virus load of tested patients, which helps to guide further actions. The PAMONO device would then act as a Distributed Cyber-Physical System (DCPS). However, new challenges arise for DCPS. Each PAMONO device should be able to provide a high throughput of samples, to process individual samples quickly, to provide a long battery life, and be able to share information.

To share information a mobile data connection with a high data throughput is needed. Here, the Long Term Evolution (LTE) network is used. LTE is a telecommunication standard for fast, wireless data transfer with mobile phones. A mobile data connection with high data throughput also enables the possibility to offload work to a server instead of processing the work on the mobile device itself. This can be especially useful to save energy on the mobile device.

The following research issues are examined:

- How can offloading capability be integrated into MOGEA-DSE?
- Are the two offloading strategies of full offloading or no offloading the only useful strategies?
- What is the effect of partial offloading with different offloading strategies?
- How do GPU architecture and offloading strategy influence each other?

The structure of this section is as follows. First, the COntext-aware POwer consumption MOdel (CoPoMo) [DIC+13] simulator is introduced in Section 8.4.1, which can simulate communication costs, energy consumption, and transfer time of an LTE connection. Second, the MOGEA-DSE approach is extended to simulate effects of offloading computations to a server or other devices in the area in Section 8.4.2. Afterward, an experiment with offloading is conducted in Section 8.4.3. Then, results and discussion of this experiment is provided in Section 8.4.4 and Section 8.4.5. Finally, summary and conclusion are provided in Section 8.4.6.

This section is based on the author's publication [LKD+14].

8.4.1 CoPoMo LTE Simulator

To offload work from mobile PAMONO devices to a server, the LTE network is very well suited. It is available in many regions and is able to transfer sufficient amount of data in a reasonable time.

To integrate offloading via the LTE network into MOGEA-DSE, the CoPoMo simulator [DIC+13; LKD+14] from Dusza et al. is used. CoPoMo is a stochastic power consumption model for LTE devices. It is used to simulate an LTE modem. The LTE modem is used to transfer data from the mobile device to a server and the other way round.

The actual use cases of CoPoMo are to simulate the transfer of either PAMONO sensor images, pre-processed PAMONO sensor images, per-pixel classification, or unclassified polygons to a server and calculate transfer time and energy consumption. For each of these use cases, the amount of data that needs to be transferred varies. The server is used to calculate the polygon classification and transfers back the result. With CoPoMo the transfer time and energy consumption can be calculated for a given amount of data.

However, the transfer time and energy consumption is not only influenced by the amount of data but also influenced by the environment. For example, the available transfer time and energy consumption can be particularly influenced by: distance of the device to the next base station, number of devices using the base station, and occlusion of the base station by buildings or other objects. As the PAMONO sensor can be used in various scenarios, the influence of the environment should be considered. In CoPoMo several system parameters and information about the environment and context can be included into the calculation. In the most advanced setup, a three-dimensional map of the environment and positions of the device during the measurement can be used. In the following the CoPoMo power model is described.

An approximation $\widetilde{P}_{\text{LTE}}$ of the actual power consumption can be calculated with the equation [DIC+13]

$$\widetilde{P}_{\text{LTE}} = \sum_i p_i(\vec{s}, \vec{c}) \cdot \widetilde{P}_i(\vec{s}), \tag{8.15}$$

with $\widetilde{P}_i$ as the estimated power consumption. The term $\widetilde{P}_i$ is calculated with two curve pieces of linear functions with the up-link transmission power. The slopes, offsets, and transitions of the two linear functions are derived empirically. Lastly, the term p_i is given as a state probability of a Markovian model M. [DIC+13]

The Markovian model M consists of four states, which model the power states of the LTE device: *idle*, *low*, *high*, and *max*. The transition probabilities from a lower power state to a higher power state in the Markovian model depend on application arrival rate and channel conditions. On the other hand, the transition probabilities from a higher power state to a lower power state depend on the file transfer duration, which depends on file size and available throughput. [DIC+13]

The throughput is determined by measurements of different channel conditions with a radio channel emulator and by determining a target SNR with help of ray-tracing simulations on different scenarios, e.g., urban, suburban, and rural. Three-dimensional maps of different areas, with base station locations and user trajectories are used for this simulation. [DIC+13]

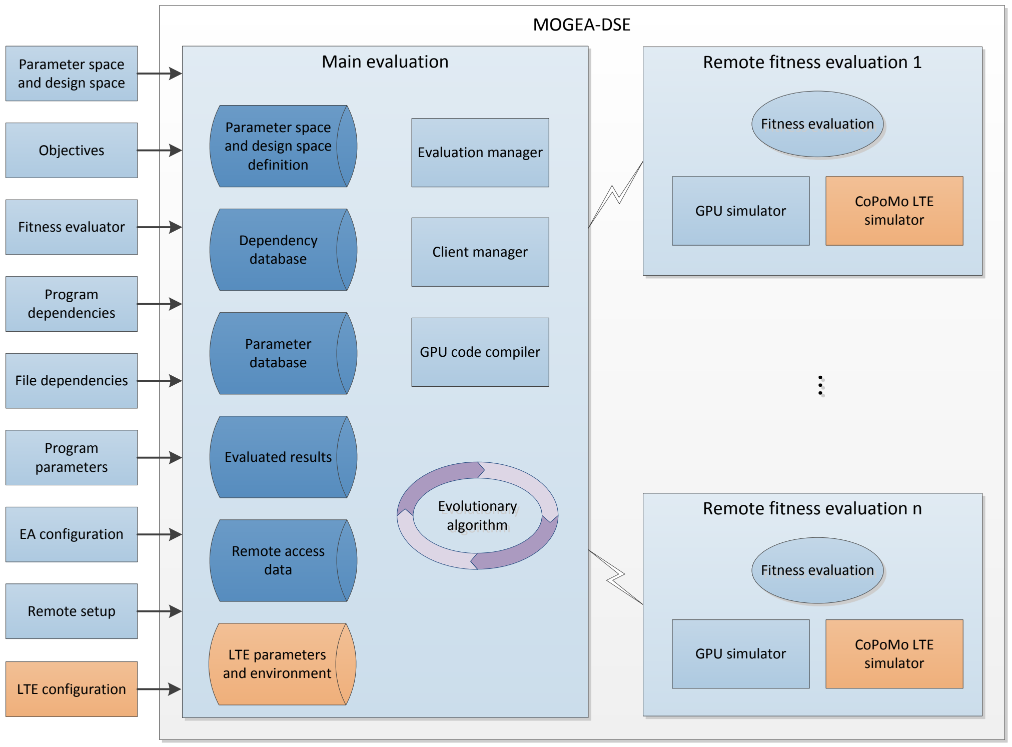


Figure 8.7: Schematic of the architectural design of the offloading capable extension of MOGEA-DSE.

The CoPoMo simulator has been verified with an independent system simulation. Different traffics and different cell environments were evaluated. The CoPoMo results and the system simulation results match very well. [DIC+13]

8.4.2 Methods

To explore offloading via the LTE network with MOGEA-DSE, the CoPoMo LTE simulator is integrated and results from CoPoMo are combined with the results from MOGEA-DSE as explained in the following. VirusDetectionCL is extended to support offloading as well. This includes different offloading points that are integrated in VirusDetectionCL. Depending on the chosen offloading point, different amount of data is processed locally or transferred to the server.

Architectural Design of MOGEA-DSE for Offloading Capable PSE and DSE

Figure 8.7 shows the offloading capable extension of MOGEA-DSE. The main difference is the CoPoMo LTE simulator that is used within the remote fitness evaluation. As input for this simulator the default LTE configuration and the current cell environment have to be provided. Both are stored in a database.

MOGEA-DSE automatically distributes CoPoMo to the nodes for the fitness evaluation in beforehand. This includes also the model of the cell environment. During the evaluation the current LTE parameters are provided to the fitness evaluations. Within the main evaluation the results from the LTE simulator are combined with other results as explained in the following.

Combination of the Models

To seamlessly integrate the results from CoPoMo in to MOGEA-DSE, the calculations of energy consumption and execution time in the fitness evaluation need to be extended. Further, the battery lifetime of the mobile device is added as a new fitness value.

The overall estimated power consumption is simply calculated by adding the power consumption $\widetilde{P}_{\text{LTE}}$, cf Equation (8.15), of the LTE device:

$$\widetilde{P} = \widetilde{P}_{\text{GPU}} + \widetilde{P}_{\text{CPU}} + \widetilde{P}_{\text{LTE}}. \tag{8.16}$$

To calculate the overall estimated energy consumption $\widetilde{E}$, the power consumption of the GPU ($\widetilde{P}_{\text{GPU}}$), the CPU ($\widetilde{P}_{\text{CPU}}$) and the LTE device ($\widetilde{P}_{\text{LTE}}$) are multiplied with the associated run time or transfer time in seconds and summed up as [LKD+14]

$$\widetilde{E} = \widetilde{P}_{\text{GPU}} \cdot \widetilde{T}_{\text{GPU}} + \widetilde{P}_{\text{CPU}} \cdot \widetilde{T}_{\text{CPU}} + \widetilde{P}_{\text{LTE}} \cdot \widetilde{T}_{\text{LTE}}, \tag{8.17}$$

with $\widetilde{T}_{\text{LTE}}$ as the transfer time determined with the CoPoMo model.

The power and energy consumption of the server are excluded from these calculations, as only the power and energy consumption of the mobile device is of interest.

Along with the energy consumption, the overall execution time is of particular interest. The approximation of the overall execution time $\widetilde{T}$ is calculated as [LKD+14]

$$\widetilde{T} = \widetilde{T}_{\text{GPU}} + \widetilde{T}_{\text{CPU}} + \widetilde{T}_{\text{LTE}} + \widetilde{T}_{\text{Server}} - \widetilde{T}_{\text{Parallel}}, \tag{8.18}$$

where $\widetilde{T}_{\text{GPU}}$, $\widetilde{T}_{\text{CPU}}$ and $\widetilde{T}_{\text{LTE}}$ is the execution time of the GPU and the CPU and the transfer time of the LTE device. The term $\widetilde{T}_{\text{Parallel}}$ is the time that can be saved by parallel calculation on GPU and CPU and by transferring data in parallel. In contrast to Equation (8.17), the execution time of the calculation on the server T_{Server} is included because the objective is the execution time until the final result is calculated. The server time is determined by measurements of actual runs of the different pipeline configurations. The time to record the images, about two minutes, is not included because no energy and execution time can be saved on this task.

Finally, the estimated battery lifetime $\widetilde{L}_{\text{Batt}}$ is of interest. It can be calculated with the equation [LKD+14]

$$\widetilde{L}_{\text{Batt}} = \frac{C_{\text{Batt}}}{\widetilde{P}_{\text{GPU}} + \widetilde{P}_{\text{CPU}} + \widetilde{P}_{\text{LTE}}}, \tag{8.19}$$

with C_{Batt} as the battery capacity.

Offloading Capable Extension of VirusDetectionCL

To enable offloading in VirusDetectionCL, the processing pipeline is extended to a client and server application. This works as follows. The pipeline is bisected at a certain point in the pipeline. The client processes all data up to this point, transfers the needed buffers to the server, and the server processes the rest of the pipeline. Client and server can use the same VirusDetectionCL software but set up only the needed pipeline elements.

Table 8.4: Offloading strategies.

Offloading point	Description
$\text{OFF0}_{\text{NONE}}$	No offloading, the images are processed locally on the mobile device.
$\text{OFF1}_{\text{COMB}}$	Offloading after the polygons are calculated. Only the final polygons have to be transferred, once the data set is processed.
$\text{OFF2}_{\text{UNCOMB}}$	Offloading of the uncombined polygons for every frame.
$\text{OFF3}_{\text{CAND}}$	Offloading after the pixel candidates are calculated. For every frame a compressed bit-mask of the pixel candidates is transferred.
$\text{OFF4}_{\text{NOISE}}$	Offloading after the constant background and the noise in the images is removed.
$\text{OFF5}_{\text{SIGNAL}}$	Offloading after the constant background in the images is removed. The transferred images contain less redundant information.
$\text{OFF6}_{\text{FULL}}$	Full offloading of the unmodified sensor images, without use of the local GPU.

Buffers are transferred by using the GStreamer [GSt16] streaming library with User Datagram Protocol (UDP) and Real-time Transport Protocol (RTP). If full offloading should be used, the pipeline is bisected after the images are received from the camera or loaded from the hard drive. All processing is then done on the server. This is especially useful for small ESs that can not process the images locally in a reasonable time.

For the use with CoPoMo, the offloading does not actually have to be performed as the offloading is only simulated. To enable this the VirusDetectionCL pipeline has been modified to calculate only the needed parts of the pipeline for a given offloading strategy. The pipeline is simply exited after the needed calculations for the current frame are done, drops the results, and processes the next frame. Seven different offloading strategies are considered. For each strategy the pipeline ends at a different point.

Table 8.4 shows and explains the conducted offloading strategies $\text{OFF0}_{\text{NONE}}$ to $\text{OFF6}_{\text{FULL}}$. With $\text{OFF0}_{\text{NONE}}$ no offloading is done and all the calculation is performed on the local GPU. With $\text{OFF6}_{\text{FULL}}$ a full offloading is performed. The sensor images are transferred to the server without any processing on the local GPU. Between these two extreme points, five points in the pipeline were identified where preliminary results are available that can be offloaded.

First, with $\text{OFF1}_{\text{COMB}}$ only the final polygons are transferred. This is done only once after the whole data set has been locally processed. It should be noted that if per-pixel features are needed for the classification, also the features needed to be transferred. Second, instead of transferring the final polygons, the preliminary polygons can be transferred for every frame with $\text{OFF2}_{\text{UNCOMB}}$. Third, the pixel segmentation can be transferred as a bit mask with $\text{OFF3}_{\text{CAND}}$. Also, no per-pixel features are taken into account here. Fourth, the virus signal that has been processed with the noise reduction can be transferred as image with $\text{OFF4}_{\text{NOISE}}$. Fifth, the virus signal that still contains the noise can be transferred with $\text{OFF5}_{\text{SIGNAL}}$.

Table 8.5: Transferred data for different offloading strategies.

Offloading strategy	Data size	Data type
$\text{OFF0}_{\text{NONE}}$	0 B	-
$\text{OFF1}_{\text{COMB}}$	26.3 kB	Compressed polygons
$\text{OFF2}_{\text{UNCOMB}}$	199.3 kB	Compressed polygons
$\text{OFF3}_{\text{CAND}}$	25.4 kB	Compressed bit masks
$\text{OFF4}_{\text{NOISE}}$	1.4 MB	Compressed images
$\text{OFF5}_{\text{SIGNAL}}$	1.4 MB	Compressed images
$\text{OFF6}_{\text{FULL}}$	1.6 MB	Compressed images

As the offloading is only simulated, the amount of data that has to be considered in the CoPoMo model needs to be determined. This is measured in an offline step. Depending on the current offloading strategy, the correct amount of data is provided to the CoPoMo model. In MOGEA-DSE this dependency is modeled as a dependent parameter as explained in Section 8.2.2.

Table 8.5 lists the amount of transferred data for different offloading strategies. Transferred polygons and bit masks are compressed with the common lossless Lempel–Ziv–Markov chain algorithm. Images are compressed using the lossless portable network graphics algorithm [Bou97]. The time to process the remaining data on a server is also measured in beforehand in an offline step.

8.4.3 Experiments

To answer the research issues, experiment $\text{EXP9}_{\text{OFFLOADING}}$ is conducted. For this experiment an important use case for VirusDetectionCL is inspected: a mobile virus detection setup with the PAMONO sensor and a laptop. Different mobile GPUs and the possibility for offloading at different points in the VirusDetectionCL pipeline are explored. Execution time and energy consumption are the objectives for this experiment.

Four different mobile NVIDIA®GEFORCE™GPUs are evaluated, as listed in Table 8.6: the GF-520M, GF-540M, GF-560M, and GF-580M. The most powerful GF-580M GPU, has eight SMPs, GDDR-5 DRAM, and runs at the slowest core clock of 620 MHz. This is used as a baseline. The least powerful GPU, the GF-520M, however has only one SMP and uses only GDDR-3 DRAM but it runs at the highest core clock of 740 MHz. The GF-540M and the GF-560M are in between with two and four SMPs.

For VirusDetectionCL the data set $\text{DS7}_{\text{POLY200NM-1}}$ is used, as explained in Section 2.2. The data set contains 160 images of size 1024 px × 128 px. The VirusDetectionCL parameters for the evaluation have been previously optimized to detect all particles without any false positives.

The CoPoMo parameters for the evaluation are listed in Table 8.7. As mentioned before, the amount of offloaded data varies depending on the offloading point.

For this experiment a single bandwidth and a fixed environment are considered. It is possible to vary these values and perform additional experiments with MOGEA-DSE so that offloading strategies for different bandwidths and different environments can be

Table 8.6: Overview of the evaluated mobile GPUs. Extended version from [LKD+14].

GPU	SMPs	SPs	Core clock	DRAM	DRAM clock	TMUs	ROPs
GF-520M	1	48	740 MHz	GDDR-3	800 MHz	8	4
GF-540M	2	96	672 MHz	GDDR-3	900 MHz	16	8
GF-560M	4	192	760 MHz	GDDR-5	1250 MHz	32	16
GF-580M	8	384	620 MHz	GDDR-5	1500 MHz	64	32

determined. A mobile device can then dynamically select a suitable offloading strategy based on current conditions.

For the calculation of the estimated energy consumption, cf. Section 8.4.2, the energy consumption of the CPU for the mobile devices is neglected as energy consumption of the CPU does not vary much between the different use cases and also no proper CPU energy model was available for the used CPU. Because almost every calculation is done either on the GPU or on the server, the CPU has only to decode the images and has to process OpenCL host code. However, in other scenarios, like energy efficient computation on ESs, the energy consumption of the CPU is not negligible. This is considered in Section 8.5 where ESs are inspected and the energy consumption of the CPU is included.

The evaluation has been performed on a compute sever with four AMD Opteron™6272 CPUs and 256 GB RAM. To measure the execution time of the different pipeline configurations, an Intel®Q9550 CPU with 8 GB RAM and a GTX 480 GPU was used.

8.4.4 Results

In experiment Exp9$_{\text{Offloading}}$, different mobile GPU architectures and different offloading points in the VirusDetectionCL pipeline have been inspected. Seven different offloading strategies and four different GPU architectures have been evaluated. The offloading strategies are described in Table 8.4 and the GPU architectures in Table 8.6.

The results for Exp9$_{\text{Offloading}}$ are shown in Figure 8.8. This figure shows four Pareto fronts as described in the following. The values for this figure are listed in Table 8.8 in detail.

For Figure 8.8 the numbers next to the points correspond to the numbers of the offloading points Off0$_{\text{none}}$ to Off6$_{\text{full}}$. However, only four of the seven offloading point from Table 8.5 are part of the four Pareto fronts. For example, it is not beneficial to offload the calculated polygons for classification as the classification can be easily done on the mobile device and the overhead does not pay off.

The four individual Pareto fronts for each of the GPUs are shown with differently shaped point symbols. The upper left solution with full offloading uses only 3.51 J but takes 27.82 s to complete. The other extreme is the solution without offloading and a local calculation on the GeForce™GF-560M. This solution only takes 0.51 s to complete, which is a speedup of 54.5 but with 35.72 J it uses ten time as much energy.

Beside these two extreme solutions, offloading only parts of the calculation gives interesting results. In other offloading approaches [LWX01; XLL07; KL10; TC13] the offloading decision is only binary, either everything is offloaded or nothing. Here however, multiple offloading points were inspected and it could be shown that it can be beneficial

Table 8.7: CoPoMo parameters used for the evaluation.

CoPoMo parameter	Value
alphaLow	0.0048
alphaHigh	0.068
betaLow	1.6
betaHigh	0.79
gamma	12
consMaxPower	2.35
powIdle	0.04
snrTarget	13
transmissionPowerForAWE	40
channel	AWGN
frameLength	10
activeTTIFrame	2
mode	nrt
protocol	UDP
constNbPRB	1
rtTargetDatarate	500 000
arrivalRate	0.01
serviceRate	0.0083
nBPrbNRT	25
maxPowerDatarateDegradationFactor	0.5
maxRTSNRDegradation	5

to offload only parts of the calculation. Of particular interest is the case, where a slight increase in execution time from 0.51 s to 0.87 s by using the offloading point $\text{OFF3}_{\text{CAND}}$ and the GF-560M instead of the GF-580M reduces the energy consumption by 77 %. If the GF-580M is kept with the same offloading point $\text{OFF3}_{\text{CAND}}$, an increase in execution time from 0.56 s to 0.85 s reduces the energy consumption by 64 % for the GF-580M.

A further decrease in energy consumption rises the execution time to larger values. For example, a decrease in energy consumption by another 38 % for the GF-560M, increases the execution time not only from 0.51 s to 0.87 s but from 0.87 s to 24.59 s. This solution is not very useful, as with a further moderate increase in execution time from 24.59 s to 27.82 s by choosing the solution with full offloading, it is possible to save even more energy. With full offloading the energy consumption drops from 9.47 J to a very good value of only 3.51 J. This saves 95 % of the energy consumption compared to the GF-580M without any offloading.

The overall Pareto front over all different GPUs is shown as a dashed line in Figure 8.8 and with bold font in Table 8.8. The overall Pareto front consists of the following solution: no offloading with an NVIDIA®GEFORCE™GF-560M, offloading of the pixel candidates with a GF-560M and a GF-580M, offloading of the images after background and noise are

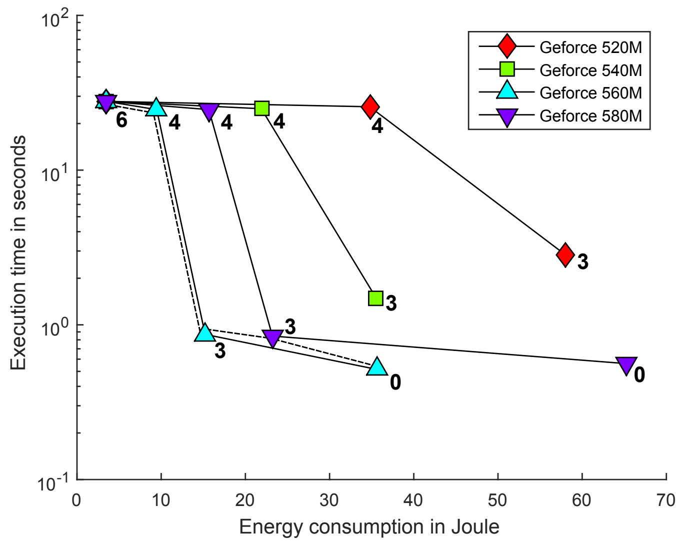


Figure 8.8: Pareto fronts for four GPUs. The overall Pareto front is shown as dashed line. The numbers next to the points correspond to the offloading strategies in Table 8.4. Zero corresponds to no offloading, three to offloading of the pixel candidates, four to offloading of the noise reduced images, and six to full offloading. See Table 8.6 for details. Adapted from [LKD+14].

removed, and finally a full offloading solution where the local GPU is not used and the unmodified images are offloaded to the server.

As already mentioned, the offloading point with offloaded polygons does not show up on any of the Pareto fronts. This solution is dominated by the points without offloading. Additionally, the points for no offloading are dominated on the local Pareto front for GF-520M and GF-540M.

8.4.5 Discussion

Some insights could be gained from the conducted experiment. A surprising result is that the most powerful GPU, the GF-580M, only provides one point to the overall Pareto front. This indicates that the program can not fully utilize the 384 cores of this GPU. The GF-560M with fewer cores but increased clock speed outperforms the GF-580M in all but this one point. An explanation to this is the relatively small image size of 1024 px × 128 px for the used data set. With larger images the GF-580M is considered to outperform the GF-560M. However, for images with 1024 px × 128 px, the GF-560M is the better choice. This result clearly shows why a DSE is important. Simply choosing the most powerful GPU can waste energy and in this case even is slightly slower than using the GF-560M with fewer cores but increased clock speed.

Another surprising result is that the most power efficient GPUs in the example, GF-520M and GF-540M, are not on the Pareto front at all. Although the faster GPUs use more power, the result is calculated faster and thus they need less energy.

By inspecting the local Pareto fronts of the single architectures, another interesting observation could be made. Only for two GPUs, the GF-560M and the GF-580M, the solution without offloading is part of the local Pareto front. For the GF-520M and the GF-

Table 8.8: Energy consumption and execution time for different offloading points and different GPUs. The transfer time and the time for processing the data on the server is included in the execution time. Savings in energy consumption are in relation to the use of the GF-580M without offloading. Points of the overall Pareto front are highlighted in bold font. Adapted from [LKD+14].

Offloading point	GPU	Energy consumption	Savings	Execution time
OFF0$_{\text{NONE}}$	**GF-560M**	**35.72 J**	45 %	**0.51 s**
	GF-580M	65.18 J	0 %	0.56 s
OFF3$_{\text{CAND}}$	GF-520M	58.0 J	11 %	2.82 s
	GF-540M	35.54 J	45 %	1.49 s
	GF-560M	**15.22 J**	77 %	**0.87 s**
	GF-580M	**23.21 J**	64 %	**0.85 s**
OFF4$_{\text{SIGNAL}}$	GF-520M	34.76 J	47 %	25.67 s
	GF-540M	21.89 J	66 %	24.94 s
	GF-560M	**9.47 J**	85 %	**24.59 s**
	GF-580M	15.66 J	76 %	24.60 s
OFF6$_{\text{FULL}}$	**None**	**3.51 J**	95 %	**27.82 s**

540M, the offloading of the pixel candidates dominates the solutions with local processing. For these two GPUs it is always beneficial to offload some amount of work.

8.4.6 Summary and Conclusion

In this section MOGEA-DSE has been extended to consider full or partial offloading of tasks from mobile devices to a server via the LTE wireless network. The CoPoMo model has been integrated into MOGEA-DSE and VirusDetectionCL has been extended to support offloading.

To answer the research issues, an experiment with seven different offloading points in the VirusDetectionCL pipeline and four different GPUs has been performed. It could be shown that the use of offloading can save up to 95 % of energy compared to simply using the fastest GPU without any offloading. A full offloading saves the most amount of energy but at the same time has the longest execution time. Using only local processing is the fastest solution but uses the most amount of energy.

Of particular interest are the solutions with partial offloading. By selecting the right offloading point and a suited GPU for this offloading point, the energy consumption can be reduced by 77 % and at the same time the execution time is still very low. This also shows that considering more offloading options than full offloading and local processing can be very beneficial. If a full offloading takes too long, a partial offloading can still save large amounts of energy without taking much longer than local processing.

Additionally, an influence of the chosen offloading point to the GPU architectures and the other way around could be shown. If the offloading point is given, different sets of GPUs are on the Pareto front. For example, only for one particular offloading point the most powerful GPU is on the Pareto front. If the architecture is given, also not all offloading points are on the Pareto front. For example, for the two slowest GPUs it is not beneficial to process the task locally.

As future work, the offloading can be extended to also use other devices in the local area or devices in other areas to process data. If one device has low battery power, insufficient computing capacity or the server is currently unavailable, the work can be offloaded to another mobile device, which has more battery power and sufficient computing capacities. Needed execution time and energy consumption of the other mobile device can be considered to make an offloading decision. Additionally, different data rates and location scenarios for the LTE communication can be evaluated. With this, offloading strategies can be explored for various external conditions. On the device the offloading strategy can then be adapted dynamically.

8.5 Multi-Objective Hybrid Hardware/Software PSE and DSE of an Embedded System

In this section a multi-objective, energy-aware, hybrid hardware/software PSE and DSE of an ES is presented as last and most complex PSE and DSE within this work. The benefits and difficulties of a hybrid PSE and DSE are especially in focus. The objectives energy consumption, execution time, and QoR are explored on an ES by incorporating actual hardware measurements. In addition, the use of approximate computing is explored as a method to save even more energy or to speed up the execution by trading QoR against improvements in one or both of the other objectives.

If a program should be deployed on an ES, determining a good software and hardware parametrization is usually difficult, especially if the objectives are conflicting and additional constraints have to be fulfilled. To achieve this, MOGEA-DSE is extended in several aspects compared to the aforementioned versions in the previous sections:

- ES target hardware is integrated in the hybrid PSE and DSE.
- A global hardware/software codesign is conducted.
- Actual hardware is measured instead of simulated.
- The energy consumption of CPU, GPU, and RAM is considered.
- The hardware parameters of CPU and GPU are part of the optimization.
- The measurement software is synchronized with the software that should be measured to enable precise measurements.
- Approximate computing is integrated in the PSE and DSE.

Particularly, the presented extensions of MOGEA-DSE are targeted toward the overall goal that should be achieved with this thesis: to identify GPU hardware for all PAMONO scenarios from HPC systems, desktop systems, mobile systems, down to small, hand-held ESs, to enable a rapid, reliable detection and counting of viruses in PAMONO sensor data.

Therefore, all not yet inspected parts of the scenarios in Figure 8.1 are in focus. In particular, the use of hand-held devices in the scenarios $\textsc{Scn2}_{\textsc{Stand-alone}}$, $\textsc{Scn3}_{\textsc{Local}}$, $\textsc{Scn4}_{\textsc{Cloud}}$, and $\textsc{Scn5}_{\textsc{Hospital}}$. Additionally, it should be shown that MOGEA-DSE is not only suited to be used with a GPU simulator but also with measurements of actual hardware.

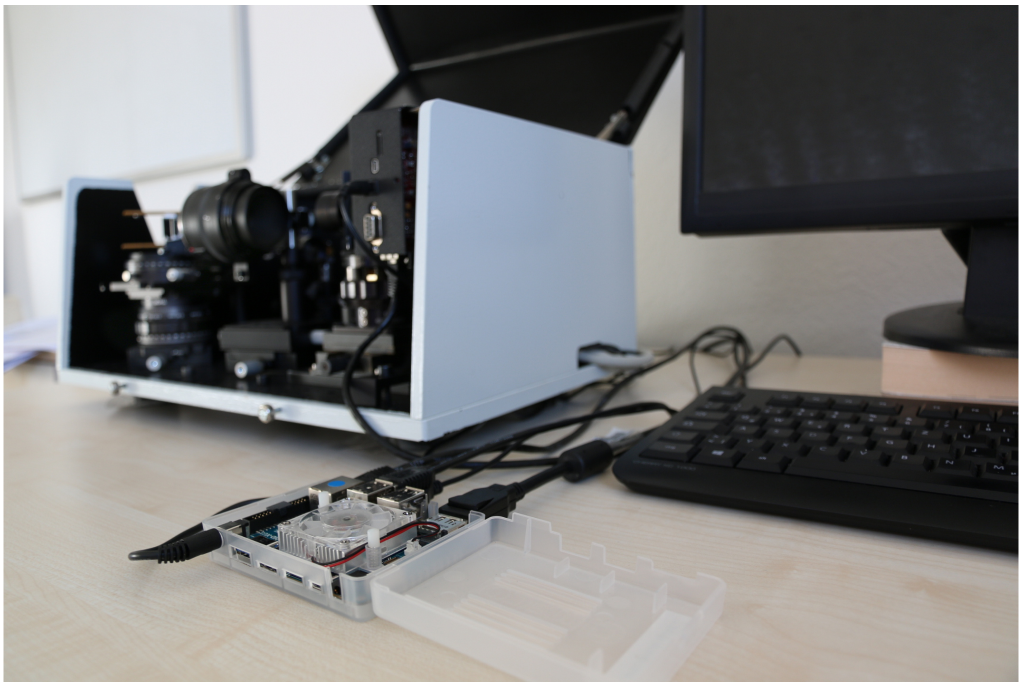


Figure 8.9: The Odroid platform and the PAMONO sensor. Example of how the size of the mobile processing device in the front compares to that of the PAMONO sensor on the left. The shown PAMONO sensor is a prototype, which will be further miniaturized in the future.

The following research issues are examined:

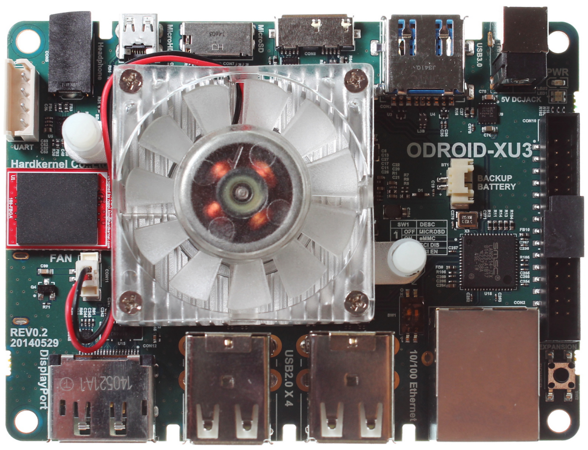
- How can actual energy measurements be integrated in MOGEA-DSE?
- How can MPSoC ES hardware be explored?
- Can the demanding soft real-time limits of VirusDetectionCL be met on a small hand-held device?
- What are the effects of a hardware/software codesign compared to considering only optimization of the hardware or software?
- How can approximate computing be used to save energy or execution time on ESs?
- Is a model-based prediction of the objectives useful?

The structure of this section is as follows. First, the Odroid platform is introduced in Section 8.5. Then, the methods for measuring the Odroid system is provided in Section 8.5.1. In Section 8.5.2 three experiments with the Odroid system as target architecture are designed and in Section 8.5.3 the measurement setup for these experiments is given. Results for these experiments are provided in Section 8.5.4 and discussed in Section 8.5.5. Finally, the section closes with summary and conclusion in Section 8.5.6.

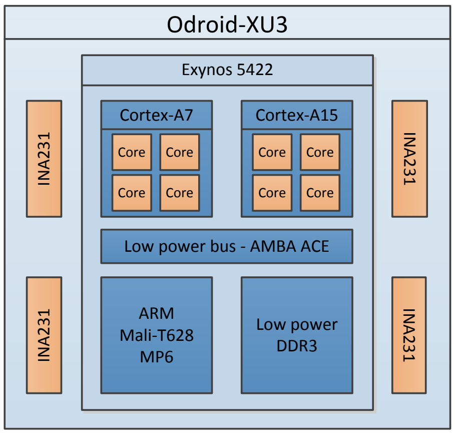
This section is based on the author's cooperative publication [NLE+15]. The first two authors contributed equally for this publication.

Odroid Platform

As target platform the HARDKERNEL®Odroid-XU3 Revision 0.2 [Har15] was chosen, as depicted in Figure 8.10a. The Odroid-XU3 is an MPSoC single board computer with a



(a) The Odroid-XU3 platform. The Odroid board uses an Exynos 5422 chip and is equipped with an eMMC module. The size of the board is 94 mm × 70 mm × 18 mm.



(b) Schematic illustration of the Odroid-XU3 platform, with the Cortex-A7 and Cortex-A15 CPUs, a Mali GPU, and low power RAM. The INA231 can be used for energy measurements.

Figure 8.10: The Odroid-XU3 MPSoC platform and a schematic illustration of it.

heterogeneous Arm®big.LITTLE architecture and a Mali GPU to enable Heterogeneous Multi-Processing (HMP).

The Odroid ES is perfectly suited for a mobile PAMONO sensor. A combination of a mobile PAMONO sensor, the VirusDetectionCL software, and the Odroid results in a CPS that can be used mobile indoors and outdoors. The Odroid offers a very small size, low power consumption and enough processing power to process PAMONO images as fast as they arrive from the camera. The size of the board is 94 mm × 70 mm × 18 mm. A power consumption of less than 7 W has been measured for the processing of PAMONO images with VirusDetectionCL.

Odroid uses the Samsung®Exynos 5422 chip [Sam15], as shown in the schematic in Figure 8.10b. The Exynos is an octa core chip, with four Arm®Cortex-A15 and four Arm®Cortex-A7 CPU cores. The Cortex-A15 cores can be clocked up to 2 GHz and the Cortex-A7 at up to 1.4 GHz. The Exynos also consists of a Mali-T628 GPU, a low power RAM, and four INA231 [Tex13] current and voltage sensors.

The Mali GPU is clocked at 600 MHz and supports OpenCL 1.1 full profile. Due to support of the full profile, the unmodified OpenCL kernel from the desktop and laptop systems can be used on the ES. As memory a 2 GB low power DDR-3 RAM clocked at 933 MHz is used, which has a memory bandwidth of 14.9 GB s^{-1}. The INA231 sensors are described in Section 8.5.1.

8.5.1 Methods

In this section the extensions of MOGEA-DSE and VirusDetectionCL toward actual energy measurements of ES are described.

The structure is as follows. First, the architectural design of MOGEA-DSE is shown. Second, the actual energy measurements on the Odroid are described. Third, dynamic

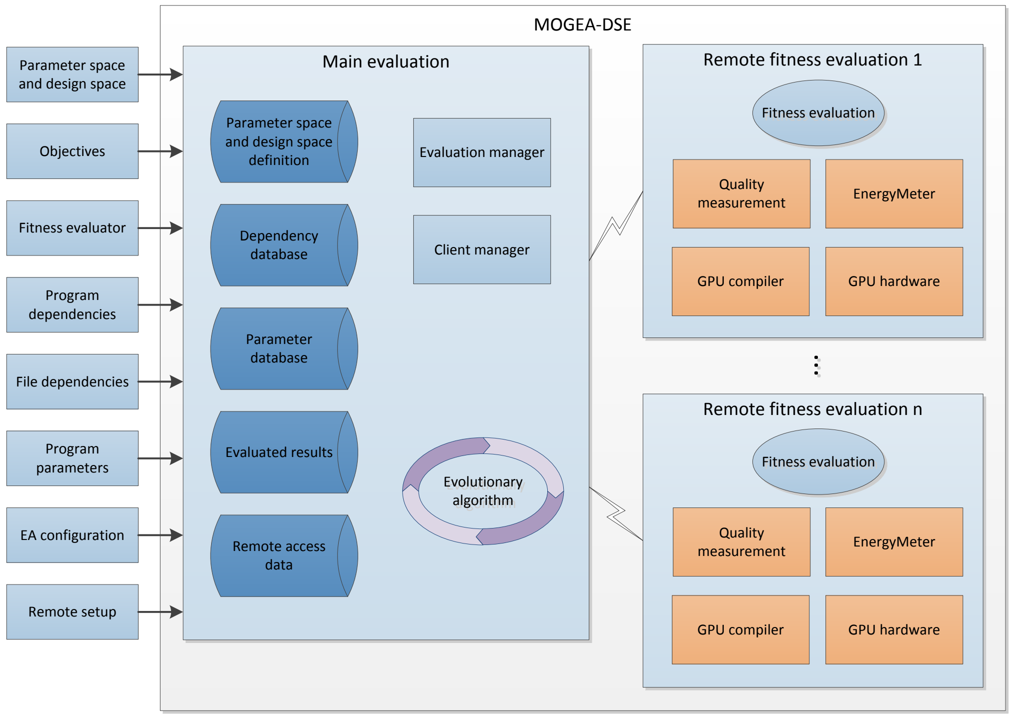


Figure 8.11: Schematic of the architectural design of MOGEA-DSE for measurement of actual MPSoC ES hardware.

frequency scaling is introduced. Fourth, the concept of approximate computing is presented. Finally, an early capitulation method is described, which can speed up the evaluation.

Architectural Design of MOGEA-DSE for Measuring the Embedded System

A schematic of how the measurement of actual ES hardware is included in the architectural design of MOGEA-DSE is shown in Figure 8.11. The remote fitness evaluation is directly performed on the evaluated hardware. For the measurements of actual MPSoC ES hardware, the EnergyMeter is used, which is explained in Section 8.5.1. A GPU compiler is used to generate the compiled code from the provided OpenCL code. The program that should be evaluated is then run on the GPU hardware. Afterward, a quality measurement is performed.

MOGEA-DSE can perform measurements on multiple ESs in parallel. As parallel measurement on the same device can result in unreliable measurements for some objectives, the measurements are performed one after the other on each device and in parallel on several devices to speed up the evaluation. The main evaluation is running on another system and distributes the needed parameters, OpenCL code, programs, files, and the EnergyMeter to the connected ESs. This ensures that the same data and code base is used on all the target platforms. It also prepares the ESs for the measurement.

The evaluation works as follows. New software and hardware parameter files are generated within the GA and are automatically distributed to the target platforms. The fitness evaluation is performed on the target device. This includes a run of the program that should be evaluated and measurements of energy consumption and execution time.

These measurements can be repeated several times to increase the accuracy of the result. The detection quality is calculated after the measurements. This has to be done only once as it does not change if a measurement is simply repeated. All fitness results are gathered and transferred back to the main evaluation. Then, a new evaluation is scheduled on the target device. If a generation is finished, the new generation is created on the master PC, a checkpoint and all partial results are stored and the evaluation for the next generation is executed.

Odroid Energy Measurements

To measure the energy consumption of the Odroids, the Energymeter software has been developed cooperatively [NLE+15] by Neugebauer and the author and has been made publicly available [NL15]. It includes an interface to the INA231 measurement hardware of the Odroid and provides an accurate timing mechanism to measure only desired parts of software as described in the following.

The INA231 current-shunt and power monitor [Tex13] from Texas Instruments®is used to measure the power consumption of the Cortex-A15 or Cortex-A7, the Mali-T628, and the low power DDR-3 RAM. There are four INA231 built into the Odroid-XU3, as shown in Figure 8.10b.

The INA231 stores the values of shunt voltage, bus voltage, current, and power in data output registers from which the values are read into the Energymeter measurement software. Measurements of the INA231 sensors are read periodically by accessing the provided INA231 registers and storing the power values for CPUs, GPU, and RAM in a list. If the measurement is finished, the average power consumptions are calculated. As the execution time is known, the energy consumptions in Joule are calculated by multiplying the average power consumptions in Watt with the execution time in seconds. The overall energy consumption is then simply calculated as sum of the energy consumption values. [Tex13]

The Energymeter is designed to be controlled by signals. It can be started, paused, and stopped by signals and sends a signal if it is ready for measurements. The signals for communication in both directions are sent and received with named pipes under Linux®. The VirusDetectionCL software is extended to communicate with the Energymeter. It waits for a ready signal from the Energymeter and sends signals if the measurement should be started or stopped. With these extensions the measurement of the energy consumption of the desired parts of VirusDetectionCL can be controlled from within VirusDetectionCL itself. If other software should be measured with the Energymeter, it is also possible to trigger the measurement without a modification of the software that should be measured.

A measurement of VirusDetectionCL works as follows. First, any not finished instances of VirusDetectionCL and the Energymeter are stopped. The fan of the Odroid is set to full speed to keep the temperature of CPU and GPU low during the measurement. This is especially important, as Backes, Rico, and Franke determined that the chip temperature is a limiting factor for some applications as the chip frequency needed to be scaled down if the chip exceeds 85 °C [BRF15]. Not needed services on the Odroid are stopped and the default governors are loaded. The pipes are flushed to remove any not consumed signals from previous, incomplete measurements.

Then, VirusDetectionCL and the Energymeter are started in signal mode. The processes are synchronized with the ready signal. VirusDetectionCL and the Energymeter both take some time to initialize. After VirusDetectionCL is ready it waits for the ready signal from the Energymeter, and after the Energymeter is ready it enqueues the ready signal into the pipe. The ready signal is kept in the pipe until it is consumed by VirusDetectionCL. This is important if the Energymeter is ready before VirusDetectionCL is ready.

VirusDetectionCL then continues to run and the Energymeter waits for the start signal to start the measurement. If in VirusDetectionCL a code part is reached that should be measured, VirusDetectionCL sends the start signal, which is received by the Energymeter. The Energymeter then keeps track of the energy consumption. If the code part that should be measured finishes, VirusDetectionCL sends the pause signal, which pauses the energy measurement.

Finally, if the last code part finishes, VirusDetectionCL sends the stop signal. The Energymeter then outputs the results of the measurements and is finished. Afterward, the generated output can be processed by MOGEA-DSE.

The initialization phase of VirusDetectionCL pipeline is excluded from the measurements on purpose. The initialization only needs to be done once and then several samples can be analyzed without starting the software again. Excluding the initialization phase has the advantage that fewer frames need to be processed to receive an accurate measurement. The energy consumption for processing all frames of a data set can easily be approximated from the processing of a few hundred frames. For example, it can be enough to process 200 frames instead of all 2000 frames of a data set and then simply multiply the energy consumption with ten. In MOGEA-DSE multiplicative factors can be specified for each objective. However, if the initialization of the pipeline is not excluded, always a full data set needs to be processed to receive an accurate result for the main processing. The ratio between images needed for the initialization and images processed fully need to be kept the same.

Dynamic Frequency Scaling

DVFS, Dynamic Frequency Scaling (DFS), and Dynamic Voltage Scaling (DVS) are key methods to save energy on CPUs. The Linux®OS kernel offers with the tool cpufreq an interface for a DFS of the Arm®big.LITTLE CPU. In the following paragraphs an overview of the strategies is given, based on [Kro07; Bro15; NLE+15].

For DFS through the Linux®OS kernel, two main policies are possible: the CPU can be set to a fixed frequency or the frequency can be adapted dynamically depending on the current requirements. For each of the two policies, different strategies can be used. These strategies can be set with so called governors. The available governors are:

- ONDEMAND
- PERFORMANCE
- POWERSAVE
- USERSPACE
- INTERACTIVE
- CONSERVATIVE

The PERFORMANCE, POWERSAVE, and USERSPACE governors are simple and are using static strategies. All three governors simply set the frequency to a fixed value within given limits: the PERFORMANCE governor sets the frequency to the highest frequency, the POWERSAVE sets the frequency to the lowest frequency, and USERSPACE governor sets the frequency to a user defined frequency. [Kro07; Bro15]

The ONDEMAND, INTERACTIVE, and CONSERVATIVE governors are using dynamic strategies. All three governors are changing the frequency depending on the current work load on the CPU. The ONDEMAND governor increases the frequency if the average work load on the CPU exceeds a given threshold and decreases the frequency if the average work load drops. The ONDEMAND governor can react quickly to changes in the work load. As a result, the performance of the system is not much affected. A fast switching of frequencies is not possible on all CPUs so the latency to switch from one frequency to another can be high. The behavior of the ONDEMAND governor can be adjusted by the user. The sampling rate, at which the current CPU usage is checked by the kernel, can be changed. Also, the up threshold, down-sampling factors, and the power save bias can be changed. The INTERACTIVE governor changes the frequencies more aggressively toward the maximum frequency and therefore it reacts faster to an increasing demand of computing power. The CONSERVATIVE governor changes the frequencies more gracefully than the ONDEMAND governor and therefore it is more suitable in a battery driven environment. With the default settings the frequency increases in 5 % chunks of the maximum frequency. This can be changed with the frequency step parameter. The CONSERVATIVE governor also has a threshold for decreasing the frequency and down-sampling factors as parameters. [Kro07; Bro15]

Approximate Computing

For software that produces a result of a certain quality, it is often possible to trade QoR against some other objective. With approximate computing [Mit16] calculations and results are allowed to be less accurate. Less accurate results might be calculated faster, with less energy, or both.

For VirusDetectionCL the detection quality, cf. Section 4.3, can be traded against energy consumption and execution time by choosing the parameters accordingly. The following parameters and options directly or indirectly influence the QoR:

- noise reduction parameters
- feature parameters
- segmentation parameters
- classification parameters
- used algorithms for the pipeline stages
- queue sizes for noise reduction
- queue sizes for features

In addition to these already available options to influence the QoR, VirusDetectionCL has been extended for approximate computing in the following parts:

- input perforation by dropping frames in the stream
- not fully processing every frame in the pipeline
- processing only a part of each frame

- reducing the number of possible polygon points
- not evaluating all trees in the Random Forest classification
- using fast relaxed math in OpenCL

However, to determine which parameters generate the desired QoR and at the same time save the most amount of execution time or energy is not an easy task. With MOGEA-DSE, approximate computing can be integrated in the optimization process. The desired QoR can then be selected from the Pareto front of the MOGEA-DSE output. Available performance measures in MOGEA-DSE include accuracy (Definition 34), precision (Definition 35), recall (Definition 36), F_1 score (Definition 37), positive agreement (Definition 38), and Matthews correlation coefficient (Definition 39).

Early Capitulation Method

To keep the overall evaluation time of MOGEA-DSE low, individuals which are time-consuming to evaluate should be handled properly. Some of these individuals are time-consuming to evaluate because their parameters are not very well suited. For example, if for VirusDetectionCL detection thresholds are chosen too low this can produce a lot of false positive detections, which then slows down all the following steps. This causes a long evaluation time but is not useful for finding a good solution. Because of the high dependencies between different parameters in the pipeline, individuals with a low quality can be generated quite often in the optimization process especially at the beginning with random values for the initialization. If these individuals are detected early, a lot of evaluation time can be saved.

With the *early capitulation* method these individuals do not have to be evaluated fully and can be marked as invalid and excluded from the breeding step. The *early capitulation* method works as follows. For actual measurements each individual is measured, e.g., three times in order to obtain an accurate execution time and energy consumption. If the first run of the individual produces bad results for the objectives it is considered as not useful. Depending on how the values for the objectives perform, the individual is handled differently. For individuals for which all objectives are far away from the considered optimum, the remaining runs are skipped. For individuals for which energy or execution time are good, the detection quality is further inspected. If the detection quality is very low, also the remaining runs are skipped. A very low detection quality is useless in practice regardless of the achieved energy consumption or execution time. All other individuals are evaluated fully.

Additionally, with the *early capitulation* method in the first run, additional checks can be done to detect a bad detection quality even earlier. As soon as the number of detections exceeds, e.g., four times the number of viruses in the sample, the experiment can be canceled. These solutions would not be able to obtain a good F_1 score above. This solution can be used for simulation-based approaches or for approaches on actual GPUs but without measurements. Additional overhead is introduced for these checks. As the actual detection quality, execution time, and energy consumption is not obtained if the run is canceled, these individuals are marked as invalid and are not considered in the breeding.

For measurement-based evaluation additional checks of the found viruses require access to the GPU memory and therefore influence the measurement. This must be avoided.

This problem can be solved with *early capitulation* method by performing a preliminary quality measure instead of performing additional checks. In a first run, e.g., only a quarter of the images can be processed. If the quality measure is obtained on the reduced input, low quality values can be quickly recognized. If the quality measures exceed certain values or the execution time or energy consumption are reasonable, the actual measurements are performed afterward. A certain quality can be guaranteed to be missed if a measurement is dismissed and the thresholds are chosen carefully. However, this introduces some additional overhead, which has to be considered if the *early capitulation* method is used this way.

8.5.2 Experiments

Three experiments, named Exp10_{HW}, Exp11_{SW}, and $\text{Exp12}_{\text{HW\&SW}}$, are conducted to evaluate hardware/software codesign for a hand-held CPS. This CPS consists of a mobile PAMONO sensor, the VirusDetectionCL software, and the Odroid-XU3 ES.

In experiment Exp10_{HW}, MOGEA-DSE automatically optimizes only the hardware related parameters of the system, in Exp11_{SW} it optimizes only the software parameters, and in $\text{Exp12}_{\text{HW\&SW}}$ it optimizes both software and hardware parameters. The three objectives for the multi-objective optimization are energy consumption, execution time, and F_1 score detection quality. The experiments are introduced in more detail in the following.

As data sets for VirusDetectionCL, training and testing data from $\text{Ds5}_{\text{11Apr13-1}}$ is used, as explained in Section 2.2. The data sets contain particles from real sensor signal. The particles have been synthetically injected into empty PAMONO sensor images. Both data sets have been generated with the SynOpSis approach [SLW+14] as described in Section 4.5 and made publicly available [SZS+14d] under the Open Database License. The training data is used for the optimization with MOGEA-DSE and the previously unseen testing data is used only for validation after the optimization is completed. The F_1 score is calculated by matching the found polygons with the labeled data.

The considered data sets are typical for the PAMONO sensor. Both, training and testing data, consist of 1000 images with 100 viruses and are 706 px × 167 px in size. The number of viruses correspond to a sample with a low virus concentration and the image size is typical for the newest sensor setup. All measurements are done for the processing of 900 images of this data set, as 100 of the 1000 images are used for the initialization phase, which is excluded from the measurements.

For Exp10_{HW} and $\text{Exp12}_{\text{HW\&SW}}$, hardware related parameters are optimized. The first hardware related parameter is the used governor for the Cortex-A15 CPU of the Odroid: PERFORMANCE, POWERSAVE, USERSPACE, INTERACTIVE, ONDEMAND, and CONSERVATIVE as explained in Section 8.5.1. The governor on the Odroids is changed with the ENERGYMETER software. The governor is only changed for the Cortex-A15 CPU cores because the Cortex-A7 cores are used to run the ENERGYMETER and are therefore excluded from the measurements.

The design space of the ENERGYMETER parameters consists of five different governors and 19 discrete values of frequencies for the Cortex-A15 core from 200 MHz to 2 GHz in 100 MHz steps.

The next optimized hardware related parameters are the work group sizes of all pipeline elements in the VirusDetectionCL software. These parameters influence how the work is partitioned and physically distributed on the SMPs of the Mali GPU. In total, 24 work group parameters are optimized.

The work group sizes have a major impact on how fast work can be processed on the GPU and how much energy the GPU consumes. Usually, work group sizes are optimized manually by the application designer with performance as the only objective. As usually different target platforms have to be matched, the optimization is done conservatively to avoid over fitting to one target. Here however, the MOGEA-DSE approach enables an automatic optimization and different target platforms can be addressed in advance or directly on the target platform in an automatic calibration step.

Finally, the last optimized hardware related parameter is the amount of memory, allocated on the GPU for storing of detected polygons. The polygon sizes are optimized for sizes of 8, 16, and 32 points. Overall, the explored parameter space for the hardware parameters exceeds the size of 3×10^9.

For $\textsc{Exp11}_{\text{SW}}$ and $\textsc{Exp12}_{\text{HW\&SW}}$, software parameters are optimized. The software parameters include the strength of the noise reduction, various detection parameters and the overall configuration of the detection pipeline. The configuration of the pipeline covers different algorithms, e.g., for the noise reduction and the pixel-based features. For each exchanged algorithm a different parameter set has been optimized. In total, 21 software parameters are optimized:

- $\textit{temporalNoiseReductionOption} \in \{0, 1\}$
- $\textit{currentImageRefs} \in \{1, \ldots, 40\}$
- $\textit{backgroundImageRefs} \in \{1, \ldots, 40\}$
- $\textit{gaussImage} \in \{0, 1\}$
- $\textit{gaussImageSigma} \in [1.5, 3.5]$
- $\textit{brightnessCorrection} \in \{0, 1\}$
- $\textit{closeDetections} \in \{0, 1\}$
- $\textit{closeDetectionsCircleRadius} \in [1.0, 5.0]$
- $\textit{openDetections} \in \{0, 1\}$
- $\textit{openDetectionsCircleRadius} \in [1.0, 5.0]$
- $\textit{detectOverexposedSpots} \in \{0, 1\}$
- $\textit{timeSeriesOption} \in \{14, \ldots, 17\}$
- $\textit{segmentationThreshold} \in [0.05, 1.0]$
- $\textit{timeSeriesDistancePatternSize} \in \{8, \ldots, 32\}$
- $\textit{timeSeriesDistancePatternSlopeRelaxation} \in \{0, \ldots, 8\}$
- $\textit{timeSeriesDistanceMinStep} \in [0.0005, 0.1]$
- $\textit{timeSeriesDistanceMaxStep} \in [0.01, 0.2]$
- $\textit{timeSeriesDistanceMinNegativeStep} \in [-0.1, -0.01]$
- $\textit{timeSeriesDistanceMaxNegativeStep} \in [-0.1, -0.01]$
- $\textit{mergingMaxDistance} \in [2.0, 12.0]$
- $\textit{mergingMaxFrameDistanceForPolygon} \in \{5, \ldots, 100\}$

This selection of parameters is based on a previous optimization so that the number of optimized parameters could be kept low. All floating point parameters, e.g., the segmentation thresholds, are converted to fixed point numbers and coded as integer genes. For each floating point parameter, the accuracy of the fixed point number is chosen to meet but not exceed the accuracy requirements. This saves some amount of evaluation

Table 8.9: INA231 configuration. Adapted from [NLE+15].

Parameter	Value
Number Of Averages	16
Bus Voltage Conversion Time	4.156 ms
Shunt Voltage Conversion Time	4.156 ms
Operation Mode	continuous
Update Period	132.992 ms

time. Even with this reduced number of parameters and discretized floating point values, the parameter space easily exceeds the size of 6×10^{21}.

The mutation rate for the NSGA-II algorithm in MOGEA-DSE is set to 0.1 with a likelihood of 1.0. The crossover is set to a tournament selection with a likelihood of 0.9.

Finally, in addition to the three experiments, a baseline measurement is performed where parameters have been optimized previously on a desktop system regarding F_1 score detection quality but have not been optimized for the Odroid hardware. This baseline measurement shows the use case, where a regular desktop application is ported to an embedded system without to make use of a hardware/software codesign regarding energy consumption and execution time.

8.5.3 Measurement Setup

An Intel®NUC D54250WYKH is used as master PC to run the main part of the MOGEA-DSE and to distribute work to nodes in the network.

All energy consumption and execution time measurements are performed on two identically configured Odroid-XU3. As the measurement is not limited to two Odroids, it can easily be accelerated by adding more Odroids. Even remotely available Odroids can be added to the measurement, as the Odroids can be set up automatically across the network connection and the connection is encrypted with Secure Shell.

The OS for the Odroids is a modified Ubuntu™14.04 Long Time Support (LTS) with kernel version 3.10.63. All GUI elements of the OS are disabled. For the experiments where the governor is not optimized automatically, the governor is set to the default governor ONDEMAND.

An air-conditioned server room is used to keep the influence of the room temperature as low as possible and the fan of the Odroid is set to full speed. The room temperature has been logged during the measurements and was in a range of 22 °C ± 1 °C.

The used settings for the INA231 sensors [Tex13] are listed in Table 8.9. All four INA231 sensors are operated in continuous mode, which averages the power values internally. Here, 16 values are averaged for both shunt and bus with a conversion times of 4.156 ms. Accumulation registers are used to sum up the intermediate measurements. As soon as the result is stored in the output register, the values are averaged. With a conversion time of 4.156 ms and 32 averaged values, the update period is 132.992 ms resulting in a frequency of 7.5 Hz. More samples per second are not possible due to limitations of the I^2C interface.

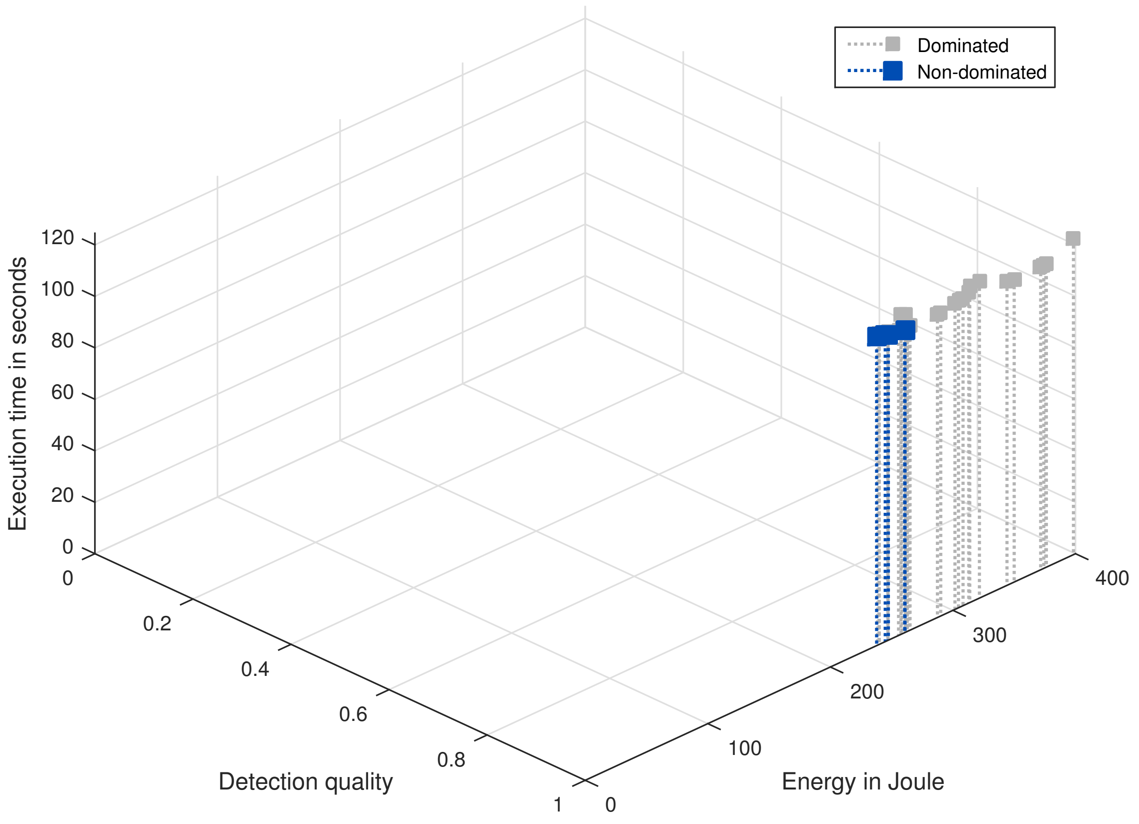


Figure 8.12: Pareto front and dominated points for Exp10_{HW}. Only hardware parameters were optimized for this experiment. The detection quality, F_1 score, is fixed at 1.0 for every point because parameters that can influence it were not modified.

8.5.4 Results

In this section the results for the three experiments Exp10_{HW}, Exp11_{SW}, and $\text{Exp12}_{\text{HW\&SW}}$ are presented.

The results are structured as follows. In Figure 8.12, Figure 8.13, and Figure 8.14 the Pareto fronts and the dominated points are plotted for the objectives energy consumption, execution time, and F_1 score detection quality. In Table 8.10 excerpts of the Pareto front are shown in detail. The points on the front are inspected with a linear regression model. Finally, in Figure 8.15 convergence plots for the three objectives are shown.

Table 8.10 shows the results from the evaluation. The first line contains the baseline measurement followed by an excerpt of ten points from the Pareto fronts of the three experiments Exp10_{HW}, Exp11_{SW}, and $\text{Exp12}_{\text{HW\&SW}}$. The values are discussed in the following.

Baseline Measurement

For the baseline measurement the VirusDetectionCL software has been measured on the Odroid systems with an unmodified parameter set. The parameter set has been previously optimized on a desktop system toward the detection quality as only objective. It has not

Table 8.10: Results of the three experiments EXP10_{HW}, EXP11_{SW}, and $\text{EXP12}_{\text{HW\&SW}}$. The table shows an excerpt of ten points from the Pareto fronts for each experiment. The objectives are F_1 score on a training data set, energy consumption, and execution time. In addition, the F_1 score for an unseen testing data set is shown. As baseline measurement a not optimized run is given in the first row, which was measured with an unmodified system and program. Extended version of [NLE+15].

Experiment	F_1 train.	F_1 test.	Energy	Savings	Exec.	Speedup	Frame rate
Baseline	1.000	0.995	370.0 J	0 %	119.8 s	1	7.5 fps
EXP10_{HW}	1.000	0.995	233.5 J	37 %	118.9 s	1	7.6 fps
	1.000	0.995	235.9 J	36 %	117.7 s	1	7.6 fps
	1.000	0.995	238.0 J	36 %	117.3 s	1	7.7 fps
	1.000	0.995	239.8 J	35 %	117.1 s	1	7.7 fps
	1.000	0.995	241.1 J	35 %	116.8 s	1	7.7 fps
	1.000	0.995	253.6 J	31 %	116.6 s	1	7.7 fps
	1.000	0.995	258.5 J	30 %	116.5 s	1	7.7 fps
	1.000	0.995	271.4 J	27 %	116.4 s	1	7.7 fps
	1.000	0.995	344.6 J	7 %	116.2 s	1	7.7 fps
EXP11_{SW}	1.000	0.995	87.2 J	76 %	29.7 s	4.0	30.3 fps
	0.985	0.931	64.6 J	83 %	22.7 s	5.3	39.7 fps
	0.953	0.883	59.7 J	84 %	20.6 s	5.8	43.7 fps
	0.870	0.844	50.5 J	86 %	17.4 s	6.9	51.7 fps
	0.830	0.734	48.6 J	87 %	15.8 s	7.6	57.0 fps
	0.753	0.723	43.9 J	88 %	17.4 s	6.9	51.7 fps
	0.750	0.693	46.3 J	87 %	14.7 s	8.1	61.3 fps
	0.684	0.612	39.2 J	89 %	13.8 s	8.7	65.1 fps
	0.519	0.413	36.4 J	90 %	12.0 s	10.0	75.2 fps
	0.413	0.400	34.4 J	91 %	10.4 s	11.5	86.8 fps
$\text{EXP12}_{\text{HW\&SW}}$	1.000	0.995	57.5 J	84 %	29.3 s	4.1	30.8 fps
	1.000	0.995	84.5 J	77 %	28.9 s	4.1	31.2 fps
	0.985	0.974	47.9 J	87 %	25.5 s	4.7	35.2 fps
	0.974	0.995	69.3 J	81 %	23.9 s	5.0	37.6 fps
	0.969	0.878	27.7 J	93 %	14.8 s	8.1	61.0 fps
	0.879	0.766	22.3 J	94 %	10.8 s	11.1	83.5 fps
	0.842	0.605	20.7 J	94 %	11.4 s	10.5	78.7 fps
	0.742	0.639	23.5 J	94 %	10.7 s	11.2	83.9 fps
	0.742	0.647	33.6 J	91 %	10.4 s	11.5	86.4 fps
	0.519	0.558	33.0 J	91 %	10.0 s	12.0	90.2 fps

been further optimized for the execution time or energy consumption of the Odroid. For the detection quality an F_1 score of 1.0 could be obtained for the training data set and 0.995 on the testing data set.

For this baseline measurement the results are shown in Table 8.10. The measured energy consumption was 370 J and the execution time 119.8 s. Regarding the frame rate, only 7.5 fps could be attained, which is too slow to meet the frame rate of the camera of 25 fps to 30 fps.

Optimizing Hardware Parameters

Figure 8.12 shows results from the experiment Exp10$_{\text{HW}}$ where only hardware related parameters were optimized. The shown Pareto front consists of 13 points with varying energy consumption and execution time. As the detection parameters were not modified in this experiment, the detection quality is fixed to a value of 1.0.

For the points on the front, less energy consumption results in higher execution time and vice versa. This has to be true as otherwise the points would not be on the front. However, for the dominated points such a relation is not always given: Figure 8.12 shows points with low energy consumption and high execution time, points with high energy consumption and low execution time, points with low energy consumption and low execution time, and points with high energy consumption and high execution time.

An excerpt of ten points of the Pareto front for Exp10$_{\text{HW}}$ is shown in Table 8.10 in more detail: The energy consumption varies from 233.5 J to 344.6 J. The execution time 116.2 s to 118.9 s, which corresponds to a frame rate of 7.6 fps to 7.7 fps. Compared to the baseline measurement, up to 3 % of execution time could be saved in the best case. The achieved savings for the energy consumption are in a range of 7 % to 37 %.

For the individual with the lowest energy consumption of 233.5 J and the longest execution time of 118.9 s, the governor POWERSAVE was used. The work group sizes varied from 8 to 64, which is relatively small, as sizes up to 256 were possible. For the maximal possible polygon sizes, also a small value of 16 points was chosen.

For the execution time no considerable speedup could be gained: the frame rate could only be increased from 7.5 fps to 7.7 fps. As expected, the individual with the fastest execution time of 116.2 s and the highest energy consumption of 344.6 J uses the PERFORMANCE governor. The work group sizes varied from 4 to 256, which comprises the full range of possible values.

To conclude, for all solutions on the front more or less energy could be saved, the execution time is at least slightly lower than in the baseline experiment, and the detection quality is the same as no software parameters were varied. Therefore, every solution from experiment Exp10$_{\text{HW}}$ dominates the solution from the baseline experiment.

Optimizing Software Parameters

For the second experiment Exp11$_{\text{SW}}$, only software parameters were optimized. Figure 8.13 shows results for this experiment. As the software parameters can influence all three objectives, the Pareto front is more pronounced compared to the results for Exp10$_{\text{HW}}$. The dominated points are also shown in the figure in order to visualize how dense the objective space was sampled. However, dominated points with very large values in execution time or energy consumption are cropped out for a clearer visualization. The missing points for

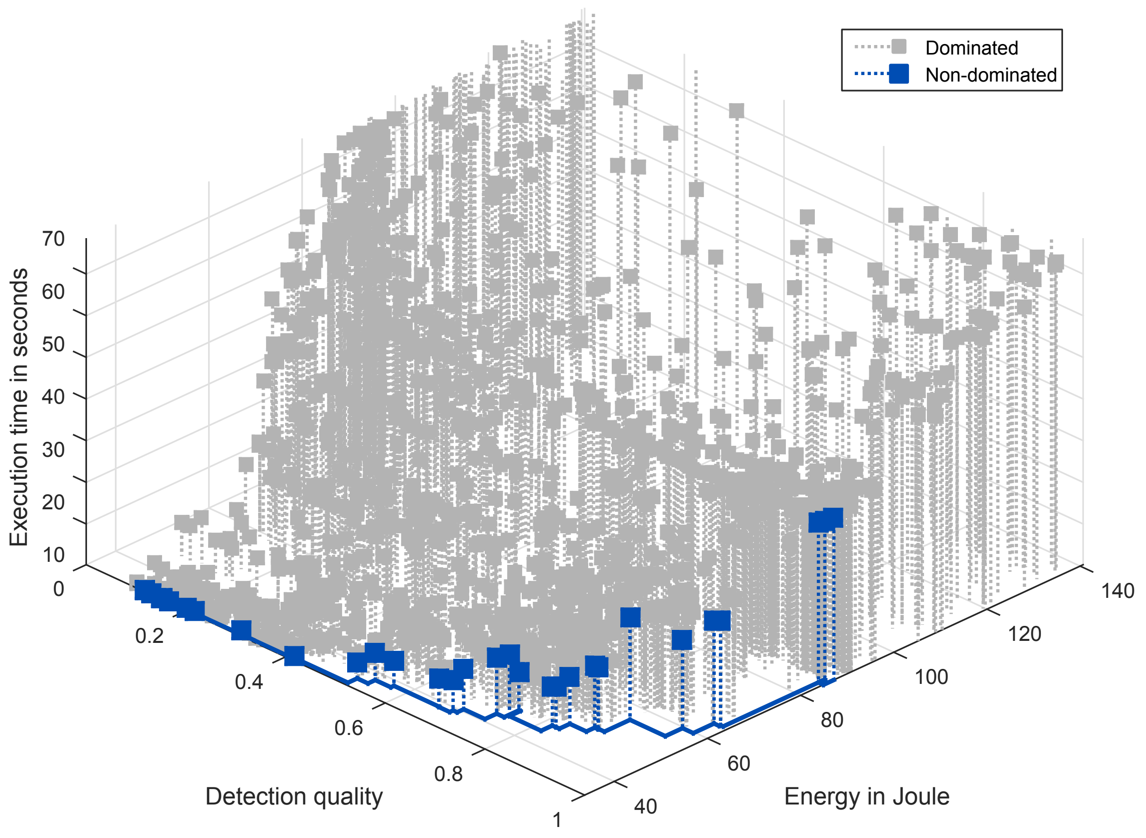


Figure 8.13: Pareto front of non-dominated points and most of the dominated points for $\text{Exp}11_{\text{SW}}$. Only software parameters were optimized for this experiment. For reasons of clarity, the Pareto front is plotted in front of the dominated points. [NLE+15]

F_1 scores below 0.113 are because of the early capitulation method, cf. Section 8.5.1, for individuals with useless F_1 scores.

The Pareto front consists of 32 points. The F_1 score on the training data set is in a range of 0.113 to 1.0, the energy consumption is in a range of 34.0 J to 90.3 J, and the execution time is in a range of 10.1 s to 29.7 s. For all individuals on the front, the soft real-time constraint is met. The frame rates range from 30.3 fps for the best detection quality to 88.7 fps for the lowest detection quality.

The front shows smooth transitions for all objectives. For the individuals on the front, the detection quality drops evenly as energy consumption and execution time are decreased. However, trading one objective against another while keeping the third objective fixed is only possible in a very limited range. In most of the cases, an improvement in one objective has to be traded against a deterioration of the two other objectives.

An excerpt of ten points of the front is shown in more detail in Table 8.10. The first point with an F_1 score of 1.0, an energy consumption of 87.2 J, and an execution time of 29.7 s dominates all points from $\text{Exp}10_{\text{HW}}$: the energy consumption could be reduced to 37.3 % of the best energy consumption for $\text{Exp}10_{\text{HW}}$, and at the same time the execution time could be decreased to 25.6 % of the best execution time. As expected, the F_1 scores on the testing data set, cf. Table 8.10, are slightly lower than on the training data set.

Overall, the F_1 scores on the testing data set show good results. This indicates that there is little overfitting to the training data set.

As an example, the solution with an F_1 score of 1.0 has been compared to the solution with an F_1 score of 0.953. This can be of interest if a camera with 40 fps should be used instead of a camera with 30 fps, as the solution with an F_1 score of 1.0 achieved a frame rate of 30.3 fps and the other one 43.7 fps. Both solutions differ in 15 parameters. The faster solution uses 40 frames for the temporal noise reduction and also a spatial noise reduction with the Gauss filter. Artifacts caused by too bright spots are detected and handled and a very low segmentation threshold has been chosen. The temporal feature extraction uses 31 frames and double detections are merged within a 17 frames window. The slower solution also uses 40 frames for the temporal noise reduction but no spatial noise reduction and no handling of artifacts caused by too bright spots. Instead, a larger segmentation threshold is chosen and only 26 frames are used for the temporal feature extraction. The window size for temporal merging of double detections was increased to 40 frames. The slower solution spends less effort on the noise reduction and compensates this by an adapted segmentation and feature extraction.

To sum up, the optimization of the software parameters in Exp11_{SW} outperforms the optimization of the hardware parameters in Exp10_{HW}. Not only a better solution for energy consumption or execution time is found but also all individuals from Exp10_{HW} are dominated. With a reduced QoR both the execution time and energy consumption can be reduced.

Optimizing Hardware and Software Parameters

For the experiment $\text{Exp12}_{\text{HW\&SW}}$, hardware and software parameters were optimized. As this is the most elaborated and most important experiment in this work, it is examined in more detail than the other experiments. Overall, the detection quality on the training and testing data set is similar to Exp11_{SW} but this experiment shows further improvements for the energy consumption and execution time.

Figure 8.14 shows results from the experiment $\text{Exp12}_{\text{HW\&SW}}$. The Pareto front of non-dominated points for the objectives execution time, energy consumption, and F_1 score detection quality is shown. In addition, the dominated points are plotted. For reasons of clarity, some dominated points far away from the front are cropped out.

The Pareto front consists of 89 points with the F_1 score on the training data set in a range of 0.165 to 1.0, the energy consumption in a range of 16.5 J to 84.5 J, and the execution time in the range of 9.7 s to 40.6 s. Same as in Exp11_{SW}, for all individuals on the front, the soft real-time constraint is met and the early capitulation method causes the missing points for F_1 scores below 0.16.

An excerpt of ten points of the Pareto front is also shown in Table 8.10 in more detail. Points with an F_1 score below 0.5 are not shown in the table as these are only useful for a few detection tasks. The energy consumption for the best detection quality could be reduced to 57.5 J, which saves 84 % energy compared to the baseline experiment. With a reduced but reasonable detection quality of 0.969, the energy consumption could be reduced by 93 % compared to the baseline and by more than 50 % compared to the solution from Exp11_{SW} with 0.953 detection quality.

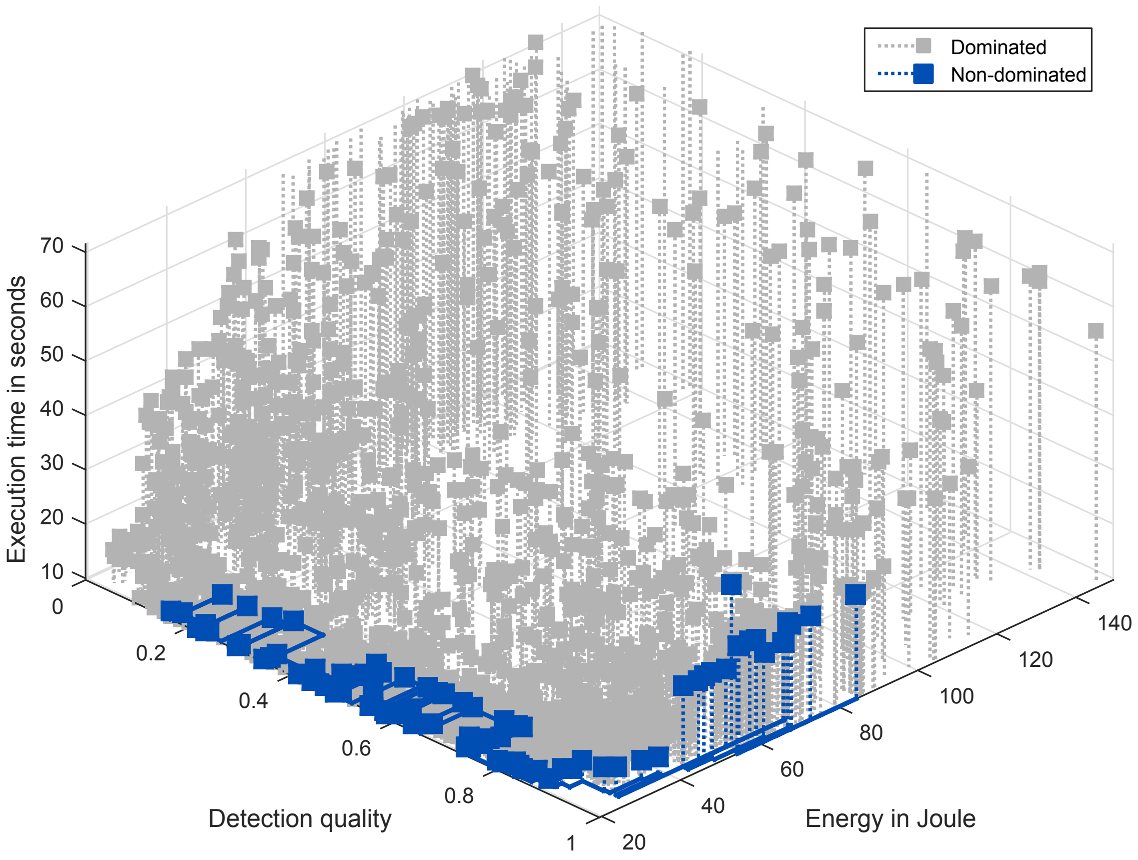


Figure 8.14: Pareto front of non-dominated points and most of the dominated points for Exp12$_{\text{HW\&SW}}$. Both, hardware and software parameters were optimized for this experiment. For reasons of clarity, the Pareto front is plotted in front of the dominated points. [NLE+15]

Compared to Exp11$_{\text{SW}}$, for all similar detection qualities a solution with a better energy consumption could be found. Also, for several points of the front of Exp11$_{\text{SW}}$, dominating solutions have been found in Exp12$_{\text{HW\&SW}}$. With the additional optimization of the hardware parameters, several solutions could be found for the same detection qualities. While in Exp11$_{\text{SW}}$ only up to two solutions for the same detection quality could be identified, Exp12$_{\text{HW\&SW}}$ shows up to seven solutions. For example, the solutions for the best detection quality vary in the range of 54.3 J to 84.5 J and 28.9 s to 29.7 s, which is quite a lot for the energy consumption.

The execution time objective shows a strong fall-off for individuals on the front with an energy consumption of less than 43 J. The execution time drops from 23.8 s to 14.8 s. This effect can not be explained by a single parameter. Instead, it is a combination of several changes of different pipeline elements: the length of the inspected time series is reduced, the temporal noise reduction uses fewer frames, and the spatial noise reduction is also less accurate. Because of the synergistic effects downward the pipeline, usually several parameters change if individuals on the front are traversed.

Some points on the front look like outliers. Some points with a detection quality between 0.2 and 0.75 show an energy consumption of about 30 J instead of 20 J. For example, one solution in Table 8.10, with an F_1 score of 0.742, uses 23.5 J and another solution with the same F_1 score uses 33.6 J. Although the second point has a 43 % higher energy consumption than the first one, they are both valid solutions on the Pareto front: the execution time for the solutions with 33.6 J is 10.4 s and therefore slightly lower than the solutions with 23.5 J and 10.7 s.

Further investigations show that solutions with a higher energy consumption and slightly faster execution time use the PERFORMANCE governors, which sets the clock frequency to the highest value. However, as the VirusDetectionCL program primarily runs on the GPU, a higher CPU frequency only has a minor influence on the execution time but a high influence on the energy consumption. As a result, the PERFORMANCE governor improves the execution time of the example solution by three percent but at the same time increases the energy consumption by 43 %. The solutions with lower energy consumption use the USERSPACE governor with a fixed frequency of 1113 MHz in average and a standard deviation of 381 This low frequency setting is enough for a fast decoding of the images and other parts of the program that run on the CPU but results in less energy consumption. Overall, the governor USERSPACE was chosen 69 times, PERFORMANCE 17 times, and all other governors combined four times. The INTERACTIVE governor was never chosen.

An overall trend can be seen in Exp11_{SW} and $\text{Exp12}_{\text{HW\&SW}}$ for the correlation between energy consumption and execution time. If the execution time is longer, more energy is consumed. However, the oversimplified statement that low execution times result in low energy consumptions is not always true. Several results can be found where the opposite is true. For example, a slightly faster solution with 86.5 fps and 33.6 J consumes 42 % more energy than the slightly slower one with 84.1 fps and 23.5 J.

Comparison with Measurements of the Odroid-XU4

Based on the results from this chapter, Popovic performed additional measurements for the baseline measurement and $\text{Exp12}_{\text{HW\&SW}}$ in her bachelor thesis [Pop16]. An Odroid-XU4, the successor of the Odroid-XU3, was chosen as the target platform. The configuration of the Odroid was slightly different from that of the Odroid-XU3, a SD card was used instead of an eMMC and UBUNTU™14.04 was used as OS.

The VirusDetectionCL software was run and measured on the Odroid-XU4 with optimized and unoptimized parameters. For the unoptimized run the parameters from the baseline measurement in Table 8.10 have been used and for the optimized run parameters from the first individual of $\text{Exp12}_{\text{HW\&SW}}$ in Table 8.10. As the Odroid-XU4 does not include INA231 sensors to measure the power or energy consumption of CPU GPU and RAM, the whole system was measured instead using a Smart Power device from HARDKERNEL®.

For the unoptimized setup with the baseline configuration, an energy consumption of 786.6 J has been measured for the whole system as an average over 20 runs. The execution time was 117.0 s. For the optimized setup an energy consumption of 226.8 J has been measured also averaged over 20 runs. The execution time was 32.5 s. [Pop16]

The saved energy, compared to the baseline measurement, was with 70 % similar to the saving measured for the Odroid-XU3. For the execution time also a similar speedup of 3.6 could be achieved. The measurements by Popovic and those in this work show consistent results and the optimized parameters for the Odroid-XU3 show a good performance on the Odroid-XU4 with regard to execution time and energy consumption.

Linear Regression-Based Modeling of the Objectives

In order to obtain further insight into the results, three linear regression models [RPC84] have been constructed for $\textsc{Exp}12_{\text{HW\&SW}}$ to predict the execution time, energy consumption, and detection quality of the individuals on the Pareto front. The WEKA framework [HFH+09] has been used to build and measure the quality of the models.

The linear regression models provide a mapping from the parameter space to the objective space. The response variable of each model is explained by one or more predictor variables. For each predictor variable a regression coefficient is given that can either be positive or negative, which indicates a positive or negative correlation of this variable to the response variable. Only the few individuals of the Pareto front are inspected as here new findings about the Pareto front are of interest. A linear model is chosen as the model is relatively easy to interpret. Non linear models are inspected in the following subsection.

First, a linear regression has been constructed for the execution time as response variable. The execution time t can be predicted with the following model:

$$\begin{aligned} t = {} & 0.0307 \cdot \text{backgroundRefs} \\ & + 19.7875 \cdot \text{gaussImage} \\ & + 2.2796 \cdot \text{brightnessCorrection} \\ & + 0.4526 \cdot \text{detectOverexposedSpots} \\ & + 6.8718 \cdot \text{simpleThresholdBasedCounting} \\ & + 0.6359 \cdot \text{timeSeriesDistancePatternSize} \\ & + 0.8188 \cdot \text{timeSeriesDistancePatternSlopeRelaxation} \\ & - 19.1131 \cdot \text{timeSeriesDistanceMinStep} \\ & - 9.8909 \cdot \text{timeSeriesDistanceMaxStep} \\ & - 2.1125 \cdot \text{timeSeriesDistanceMinDownStep} \\ & + 1.4994 \cdot \text{timeSeriesDistanceMaxDownStep} \\ & + 12.3599 \cdot \text{powersaveGovernor} \\ & + 2.0148 \cdot \text{userspaceGovernor} \\ & - 1.1843 \cdot \text{a15frequInGHz} \\ & - 0.0638 \cdot \text{MARCHING\_SQUARES\_WORK\_GROUP\_HEIGHT} \\ & - 6.7275. \end{aligned} \tag{8.20}$$

The model has been validated with a stratified 10-fold cross-validation [RPC84]. The Pearson's product-moment correlation coefficient [RN88], Mean Absolute Error (MAE), and Relative Absolute Error (RAE) were calculated to measure the quality of the model. MAE and RAE are defined as follows:

Definition 52 (MAE)**.** The MAE over $n \in \mathbb{N}$ true values $x_1, \dots, x_n \in \mathbb{R}$ and n estimated values $\hat{x}_1, \dots, \hat{x}_n \in \mathbb{R}$, as defined in WEKA [HFH+09], is given as

$$\mathrm{MAE} := \frac{\sum_{i=1}^{n} |\hat{x}_i - x_i|}{n}. \tag{8.21}$$

Definition 53 (RAE)**.** The RAE, as defined in WEKA [HFH+09], is given as

$$\mathrm{RAE} := \frac{\sum_{i=1}^{n} |\hat{x}_i - x_i|}{\sum_{i=1}^{n} |\mu - x_i|}, \tag{8.22}$$

with μ as the average over all x_i according to Definition 19.

As a result, the Pearson's correlation coefficient for this model is 0.96, MAE is 1.04, and RAE 20.86 %. The correlation coefficient 0.96 indicates a high correlation. However, the regression model has to be interpreted carefully: Predictor variables with a large absolute regression coefficient may not have large influence on the response variable, because the predictor variables are of different ranges. Also, correlation does not imply a causation.

With the interpretation of the model, several insights can be gained. The predictor variable *gaussImage* has the largest absolute regression coefficient. With the *gaussImage* parameter the Gauss filter for spatial noise reduction is enabled, as explained in Section 5.3.3. Here, *gaussImage* has in fact the largest influence on the response variable execution time. In a fitted simple linear regression model, this single predictor variable shows a correlation coefficient of 40.3 %. Therefore, this parameter has a high influence on the response variable. Also, the causation of *gaussImage* influencing the execution time is given: the Gauss filter uses a large convolution kernel and the discretized Gaussian is calculated for every work item. Both has a strong influence on the execution time.

By inspecting the points on the front, it could be found that the *gaussImage* filter is chosen only for points with a detection quality of more than 0.985. Every point with a detection quality of 1.0 uses the *gaussImage* filter. This shows clearly that the *gaussImage* filter is beneficial for a high detection quality. However, these points are the only ones with an execution time above 26 s and also show a higher energy consumption.

The parameters *brightnessCorrection* and *detectOverexposedSpots* are of minor importance. Only one point on the front makes use of the brightness correction, as described in Section 5.2.2. Although the parameter *detectOverexposedSpots* is used by about half of the points on the front, it is a Boolean type parameter with zero or one as value, it has a low regression coefficient and therefore it is of less influence.

Two more important predictor variables are *simpleThresholdBasedCounting* and *timeSeriesPatternSize*. With the *simpleThresholdBasedCounting* parameter, the threshold-based counting algorithm, as explained in Section 5.4.2, is used for temporal feature extraction. With the *timeSeriesPatternSize* parameter, the number of previous frames that are inspected is influenced. In all cases where *simpleThresholdBasedCounting* is not selected *temporalTemplateMatching* is used, as explained in Section 5.4.2. Because only two different algorithms of the four possible temporal feature extraction algorithms are selected for the individuals on the front, *temporalTemplateMatching* does not show up in the model.

The parameters *timeSeriesDistanceMinStep* and *timeSeriesDistanceMaxStep*, in a range of 0.013 to 0.192, show a negative correlation. Lower values produce longer execution times. This can be explained as lower values are needed for the larger scores, and to achieve a better F_1 score, usually more time needs to be spend on the processing.

The governor related parameters show a positive influence of the parameters *powersaveGovernor* and *userspaceGovernor* on the execution time. As already shown, the POWERSAVE governor causes an increase of execution time. With USERSPACE the increase is also positive but is dependent on the frequency, which is given in *a15frequInGHz*. As expected, the frequency shows a negative correlation coefficient. With higher frequencies the execution time is reduced.

Finally, for the OpenCL work group sizes the parameter *MARCHING_SQUARES_-WORK_GROUP_HEIGHT* is the only parameter that shows up in the model. If the work group height is increased, the execution time drops. For the other work group sizes, two different influences on the execution time can be observed: for some OpenCL kernels it is beneficial to increase the work groups and for others it is beneficial to decrease the work group sizes.

However, the work group sizes have only a minor impact on the execution time because all individuals on the front use work group sizes in a reasonable range. For example, to sort out double detections of the same virus, polygons at nearly the same location and at nearly the same point in time are combined. This OpenCL kernel is difficult to parallelize. Therefore, small work group sizes are beneficial and all individuals on the front use small work group sizes. This would be different if also dominated individuals were used to build the model. Too small or too large work group sizes or a wrong shape can heavily influence the execution time as already observed in other experiments.

The next linear regression model has been set up to predict the energy consumption with the individuals of the Pareto front. The energy consumption e can be predicted as

$$
\begin{aligned}
e = {} & 0.0723 \cdot \text{totalRefs} \\
& + 33.615 \cdot \text{gaussImage} \\
& + 8.282 \cdot \text{simpleThresholdBasedCounting} \\
& + 1.4776 \cdot \text{timeSeriesDistancePatternSize} \\
& + 10.9179 \cdot \text{performanceGovernor} \\
& - 9.237 \cdot \text{userspaceGovernor} \\
& + 8.1482 \cdot \text{conservativeGovernor} \\
& - 6.5783 \cdot \text{powersaveGovernor} \\
& + 8.3393 \cdot \text{a15frequInGHz} \\
& + 0.1206 \cdot \text{REMOVE\_BG\_WORK\_GROUP\_HEIGHT} \\
& - 0.0899 \cdot \text{TIME\_SERIES\_WORK\_GROUP\_WIDTH} \\
& - 0.2801 \cdot \text{MARCHING\_SQUARES\_WORK\_GROUP\_HEIGHT} \\
& - 0.2699 \cdot \text{VIRUS\_LIST\_COMBINE\_WORK\_GROUP\_WIDTH} \\
& - 0.3099 \cdot \text{VIRUS\_LIST\_COMBINE\_WORK\_GROUP\_HEIGHT} \\
& - 0.1474 \cdot \text{FOREGROUND\_BACKGROUND\_WORK\_GROUP\_HEIGHT} \\
& + 1.5091.
\end{aligned}
\tag{8.23}
$$

The stratified 10-fold cross-validation for this model shows a correlation coefficient of 0.96, an MAE of 2.79, and an RAE of 24.76 %.

By inspecting these parameters in detail, it could be found that a wide variety of values was used throughout the front. The correlation of which parameter influenced the energy consumption was not obvious by only inspecting the individuals and their parameters. With help of the linear regression model, further insights could be found.

A hypothesis was that the energy consumption is strongly influenced by the amount of data processed in every frame. The amount of processed data in every frame is influenced by the parameters *totalRefs*, *timeSeriesDistancePatternSize*, and *POLYGON_MAX_-SIZE*. As can be seen in the model, three of these parameters play also an important role to predict the energy consumption. *totalRefs* is equal to the sum of *currentImageRefs* and *backgroundRefs*. It influences how many frames are used to reduce the temporal noise. With *timeSeriesDistancePatternSize* the number of frames for the time series-based feature calculation is influenced and *POLYGON_MAX_SIZE* influences how much memory is reserved for the polygons.

An interesting observation can be made for the parameter *totalRefs*: the correlation of *totalRefs* is much less than for *timeSeriesDistancePatternSize* although both parameters influence the number of processed frames. This can be explained as for all individuals the average calculation over time has been chosen over the median calculation. As shown in Section 5.3.2, the median calculation needs to access all images, whereas the average calculation accesses only the newest and the oldest image to update a sum and calculate a new average. As a result, the implementation with the average is much more energy efficient, which explains the low regression coefficients for *totalRefs*.

For the parameter *gaussImage*, the increased execution time combined with the increased number of memory accesses causes the higher energy consumption. This can also be observed in the data as nine out of twelve individuals with an energy consumption of more than 50 J make use of the Gauss filter and none of the individuals with lesser energy consumption.

With the parameter *simpleThresholdBasedCounting*, the simple temporal feature extraction as explained in Section 5.4.2 is selected. This also influences the segmentation. For the individuals on the front, only the simple threshold-based counting and a highly optimized version of the temporal template matching, cf. Section 5.4.2, is selected. With the latter selected for the high detection qualities. Dynamic time warping was not used for the individuals on the front. The positive correlation coefficient of the parameter *simpleThresholdBasedCounting* has to be interpreted carefully, as the highest energy consumption is achieved if the parameter is set to zero and the lowest for the parameter set to one. Hence, this parameter cannot be considered in isolation.

For the best detection qualities, the parameter *timeSeriesDistancePatternSize* was chosen as 12 or 24, which is in the middle of the possible range of 8 to 32. For all 69 individuals with an energy consumption below 13 J, the minimum amount of frames has been chosen. However, this amount is enough to reach detection qualities up to 0.891, which is somehow surprising.

The last linear regression model has been set up to predict the detection quality F_1 score:

$$F_1 = 0.0115 \cdot \text{backgroundRefs} \tag{8.24}$$

$$
\begin{aligned}
&- 0.0086 \cdot \text{totalRefs} \\
&+ 0.3248 \cdot \text{gaussImage} \\
&+ 0.289 \cdot \text{brightnessCorrection} \\
&+ 0.184 \cdot \text{simpleThresholdBasedCounting} \\
&- 0.7934 \cdot \text{segmentationThreshold} \\
&+ 0.0348 \cdot \text{timeSeriesDistancePatternSize} \\
&+ 0.0955 \cdot \text{timeSeriesDistancePatternSlopeRelaxation} \\
&- 3.4357 \cdot \text{timeSeriesDistanceMinStep} \\
&+ 0.1379 \cdot \text{timeSeriesDistanceMaxDownStep} \\
&- 0.179.
\end{aligned}
$$

As expected, this model has the lowest correlation coefficient and highest error values. The stratified 10-fold cross-validation shows a correlation coefficient of 0.36, an MAE of 0.17, and an RAE of 76.14 %. This is because of the highly non-linear dependencies of the parameters. Because of heavy dependencies to previous pipeline stages, the prediction of the detection quality is a difficult task.

Despite the low correlation coefficient and high error values, a few insights can be gained from this model, too. For *totalRefs* a slightly negative correlation is given. By inspecting the individuals in detail, this can be explained. While for the low detection qualities high values from 29 to 73 are used, all individuals with a detection quality of 1.0 use 38 frames to calculate the approximated virus signal as explained in Section 5.3.2. As previously stated, the Gauss filter is useful for the very high detection values and the sigma shows only slightly variations. Although, *brightnessCorrection* shows a positive correlation, only one individual uses the brightness correction. For all other individuals this value is zero and therefore this regression coefficient has low explanatory power.

Similar as for the energy consumption model, the *simpleThresholdBasedCounting* cannot be considered in isolation. Although the correlation coefficient is positive, all except two of the largest F_1 scores are achieved with the highly optimized version of the temporal template matching, cf. Section 5.4.2, and not with the simple threshold-based method as explained in Section 5.4.2.

A surprising and unexpected result is that the segmentation threshold shows a negative correlation in the model. During manually performed parameter optimization, the threshold was usually chosen higher to sort out unwanted signals and increase the detection quality. In contrast, with MOGEA-DSE low threshold values have shown to be better for a high detection quality. However, the low detection values can only result in good detection quality if the noise reduction in the earlier pipeline stages removes as many unwanted signals as possible. With MOGEA-DSE combinations of noise reduction parameters and low segmentation thresholds have been identified that produce very good results. In manual optimization these combinations have not been considered before. That this insight could be obtained, is truly a highlight of MOGEA-DSE.

For the parameter *timeSeriesDistancePatternSize*, only a low positive correlation coefficient has been identified. All individuals with an F_1 score below 0.9 use only 8 frames to calculate the temporal features, whereas the other individuals use 12 to 30 frames.

Table 8.11: Model-based prediction of the objectives execution time, energy consumption, and F_1 score on 2688 individuals. The correlation coefficient (Corr. coeff.) indicates the quality of the model, higher values are better. For MAE, RAE, and execution time, lower values are better. Best values are highlighted with bold font.

Method	Objective	Corr. coeff.	MAE	RAE	Exec.
Linear Regression	Execution time	84.6 %	9.8	39.4 %	**0.2 s**
	Energy consumption	89.9 %	19.0	47.6 %	**0.42 s**
	F_1 score	53.4 %	0.2	80.2 %	**0.44 s**
Multilayer Perceptron	Execution time	84.8 %	8.4	34.0 %	42.2 s
	Energy consumption	**95.8 %**	**11.1**	**22.0 %**	58.0 s
	F_1 score	44.8 %	0.3	100.2 %	65.0 s
SVM (SMOreg)	Execution time	83.6 %	7.8	31.6 %	346.1 s
	Energy consumption	89.0 %	15.2	30.0 %	562.5 s
	F_1 score	51.2 %	22.2	75.9 %	37.3 s
Random Forest	Execution time	**89.4 %**	8.6	34.8 %	2.1 s
	Energy consumption	94.3 %	17.0	33.7 %	2.1 s
	F_1 score	**77.8 %**	0.2	58.6 %	2.6 s
M5P	Execution time	88.2 %	**7.6**	**30.5 %**	1.5 s
	Energy consumption	95.5 %	13.1	26.0 %	1.5 s
	F_1 score	75.9 %	**0.1**	**47.8 %**	2.4 s

Finally, the parameter *timeSeriesDistanceMinStep* is of interest. For this a negative correlation is given, which is also reasonable. With lower step sizes smaller increases in intensity can be detected. With this, more of the area of individual virus signals is detected. Similar to the segmentation threshold, a good noise reduction is needed in order to make smaller *timeSeriesDistanceMinStep* values usable.

To sum up, the combined optimization of hardware and software parameters shows the best results of all three experiments. Including the hardware parameters could especially improve the energy consumption. The execution time could only be improved in a limited range, as the ONDEMAND governor, which can adapt the CPU frequency to different loads, was used in the baseline measurement and the GPU parameters have only a minor influence.

Model-Based Prediction of the Objectives

As shown, a linear regression model is suitable to explain the objectives execution time and energy consumption of the individuals on the Pareto front. However, a linear regression model is not suited to explain the objective F_1 score. Therefore, in addition to the regression models, also other models are inspected in the following. The aim is to find out if a model-based prediction of the objectives is possible. This could also lead to a model-based optimization. For this reason, all individuals are considered, including the dominated ones. The methods linear regression, multilayer perceptron, SVM for regression [SKB+00], Random Forest [Bre01], and M5P [Qui92; WW97] from WEKA [HFH+09] are used.

For every method and each of the three objectives, one model is set up and trained. Again a 10-fold cross-validation is used for validation of the models.

Table 8.11 shows the results that could be achieved. For each model and each objective that should be predicted, the correlation coefficient, MAE, and RAE of the 10-fold cross-validation, and the time to set up and train the model are given. The best values are highlighted in bold font.

Regarding the correlation coefficient, the multilayer perceptron and the Random Forest methods achieve the best results. For the objective execution time, the correlation coefficient is 89.4 % with the Random Forest, for the objective energy consumption it is 95.8 % with the multilayer perceptron, and for the objective F_1 score it is 77.8 % with the Random Forest. Although the multilayer perceptron achieved the best correlation coefficient for the objective energy consumption, it achieved the worst correlation coefficient for the F_1 score. The SVM for regression achieved the worst correlation coefficient for the objectives execution time and energy consumption.

Regarding the MAE and the RAE, the multilayer perceptron and the M5P methods achieved the best result. The multilayer perceptron achieved an MAE of 11.1 and an RAE of 22.0 % for predicting the energy consumption. M5P achieved the best values for the objectives execution time and F_1 score.

Regarding the time to set up and train the model, the linear regression was the fastest method for all three objectives with less than half a second. The Random Forest and M5P methods perform very well. The multilayer perceptron and the SVM for regression are the two slowest methods in this experiment.

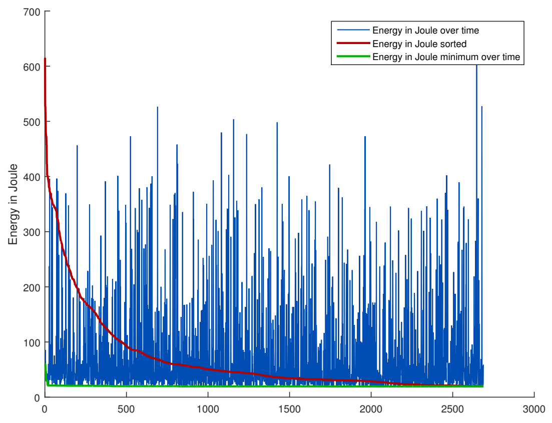
Overall, the tree-based models Random Forest and M5P achieved good results on all three objectives. They also provide a much faster execution time than the multilayer perceptron and the SVM for regression, which can be important if these are used withing a model-based optimization.

Convergence of the Objectives

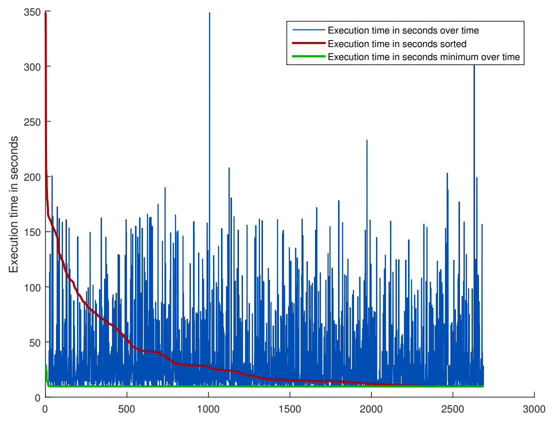
The convergence plots in Figure 8.15 show the objectives energy consumption (a), execution time (b), and F_1 score detection quality (c) over all individuals. The plot for each objective shows the objective values, plotted as a function over time, the sorted objective values, and the best objective values over time.

The plotted values of the objectives over time show how the variation of the objectives looks like. And the sorted values of each objective show if all important values have been reached. Furthermore, the convergence plots shows how fast MOGEA-DSE has found good solutions by plotting the best values for energy consumption, execution time and detection quality over time.

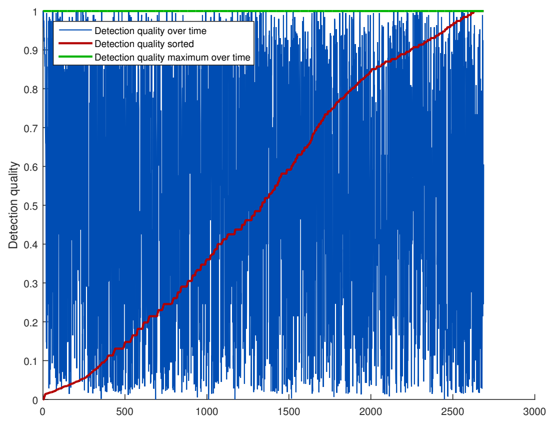
As a result, it can be seen that in less than 100 evaluations very good solutions could be identified, that throughout the evaluation a wide span of solutions have been found, and that the target space has been sampled very well and to a sufficient level.



(a) Convergence plot for the objective energy consumption in Joule, lower is better.



(b) Convergence plot for the objective execution time in seconds, lower is better.



(c) Convergence plot for the objective F_1 score, higher is better.

Figure 8.15: Convergence line plots for the objectives energy consumption (a), execution time (b), and the F_1 score detection quality (c) over all individuals.

8.5.5 Discussion

Several interesting insights and unexpected results could be gained during the evaluation of the MOGEA-DSE approach on actual ES hardware. By integrating the ENERGYMETER and allowing a reduced QoR, a wide variety of different Pareto-optimal solutions has been found. With the identified solutions, hardware and software parameters can be switched to adapt the CPS to the current demands depending on the field of application.

As has been shown, an optimization of the hardware parameters in experiment EXP10$_{\text{HW}}$ provides good results on the energy savings. But an optimization of only the software parameters in experiment EXP11$_{\text{SW}}$ outperforms the optimization of only the hardware parameters. While for experiment EXP10$_{\text{HW}}$ the best energy saving in Table 8.10 is only 37 % and the best frame rate only 7.7 fps, the worst energy saving of EXP11$_{\text{SW}}$ is 76 % and the worst frame rate is 30.3 fps. The results from EXP10$_{\text{HW}}$ are dominated by the results from EXP11$_{\text{SW}}$. This shows clearly that the optimization of the software parameters of VirusDetectionCL has a huge impact on the objectives and should not be neglected. Only optimizing the hardware parameters to save energy or gain a speedup can waste a lot of potential.

As expected, simultaneously optimizing software and hardware parameters in a hardware/software codesign manner as in experiment Exp12$_{\text{HW\&SW}}$ achieves the best results. Energy savings of 77 % to 94 % and frame rates of 30.8 fps to 90.2 fps could be achieved with MOGEA-DSE for F_1 scores of 0.5 to 1.0. With these results a mobile use of the PAMONO sensor with live processing of the data is possible without any problems. Achieving this good results was entirely unexpected. It has been expected that the optimization of the hardware and software parameters lead to a solution with good detection quality and a frame rate of perhaps 10 fps to 15 fps and that for further speedup the QoR has to be drastically reduced.

A very surprising result and a highlight for the MOGEA-DSE method is that two solutions with an F_1 score of 1.0 on the training data, 0.995 on the testing data, and a frame rate of 30.8 fps and 31.2 fps could be found. This results in a speedup 4.1 and enables an almost perfect detection quality while meeting the soft real-time constraint of an on the fly virus detection. Although the energy consumption is 2.8 and 4.1 times as high as that from the most energy efficient solution, still 84 % an 77 % energy is saved compared to the baseline experiment. This result alone exceeded the expectations of what is possible with VirusDetectionCL on an ES and with MOGEA-DSE.

To compare this result with others, Backes, Rico, and Franke [BRF15] also inspected the performance of computer vision applications on ESs. They used the Odroid-XU3 and an Exynos 5250 Arndale®board as target platform. For their *KinectFusion* computer vision application running on a Mali GPU, they achieved, by several optimizations, a speedup of up to four while, at the same time, the energy consumption could be reduced. The speedup and energy savings are calculated with regard to their CPU implementation. Despite the speedup they only achieved a frame rate of 2.2 fps to 3.1 fps, which is still far from their target frame rate of 30 fps. They conclude, that the used MPSoCs are not useful to run demanding computer vision tasks like *KinectFusion* with a sufficient frame rate. In contrast to their results, here the speedup and savings in energy consumption could be achieved compared to an existing GPU program as baseline and not simply by porting a CPU program to the GPU and using the CPU program as baseline. By inspecting their results in more detail, it can be seen that almost all speedup in [BRF15] has only been achieved due to the porting to the GPU and not by their other optimizations.

For experiments Exp11$_{\text{SW}}$ and Exp12$_{\text{HW\&SW}}$, also the use of approximate computing has been explored. This led to a surprising result. With a decline in QoR of 3.1 % in the F_1 score, a further reduction of 52 % in energy consumption could be achieved compared to the most energy efficient solution with 1.0 detection quality. An F_1 score of 0.969 on the training data set and 0.878 on the testing data set is still a very reasonable quality for many use cases of the PAMONO sensor. Missing a few single viruses is usually acceptable.

To save more energy the decrease in QoR has to be quite high. If such solutions are still useful, highly depends on the task. For example, an F_1 score of 0.5 can still be feasible for virus test that only detects if a sample is positive or negative. For this test the actual virus load in the sample is of no interest, this can be examined later in an additional test. Of importance is only if the sample contains viruses or not. For this, the number of detected individual viruses can be relatively low but the number of FP detections should be zero in the best case. If several hundred viruses attach to the PAMONO sensor it does not matter if 50 % are missed, the test result would still be positive. However, for such a test the objective F_1 score can be changed to a more appropriate fitness value taking the specific demands into account. By optimizing toward other objectives, MOGEA-DSE can identify solutions for several scenarios.

By further inspecting interesting solutions on the front, it has been found that the governors CONSERVATIVE and PERFORMANCE show the best execution times. The PERFORMANCE governor lets the Cortex-A15 run at the highest clock rate and the CONSERVATIVE governor smoothly increases or decreases the frequency depending on the current load on the CPU. As the pre-processing has been excluded from the measurement, the CONSERVATIVE governor is able to adapt to the load of the pre-processing before the measurement starts. With the governor USERSPACE a fixed frequency can be chosen. It could be shown that higher frequencies result in better execution times but the frequency is not as important as expected. Also, higher frequencies cause a higher energy consumption. It could be shown that a frequency of 1.1 GHz is a good compromise between performance and energy consumption.

A surprising result is that the simple threshold-based counting temporal feature extraction performs well enough for F_1 scores up to 1.0. Only for a slightly faster processing combined with an F_1 scores greater than 0.99 the more sophisticated temporal template matching is needed. This is counter intuitive as the simple method by itself runs faster. But the simple method is more demanding to the noise reduction. Consequently, using the faster simple method is slower in total.

By inspecting the models of the linear regression, several insights could be gained. For example, the Gauss filter, cf. Section 5.3.3, has been identified to have a crucial impact on the very high detection quality values and also on the energy consumption and execution time. Here, a potential for further optimization is given. It has been shown that for this data set, the Gauss filter sigma should be set to 1.6 to enable detection qualities above 0.985. If such knowledge is available the Gaussian can be pre-calculated for typical sigmas, which can be used to speed up the calculation and save energy.

The most surprising and unexpected result for the detection quality is that with more effort on the early pipeline stages the segmentation threshold for the time series features can be chosen very low. MOGEA-DSE was able to find solutions with high quality that, even with expert knowledge, were previously not considered as beneficial. Manually with expert knowledge generated solutions spend medium effort on the noise reduction and more effort in an accurate segmentation or a more accurate classification of false positives. The solutions automatically generated with MOGEA-DSE were able to achieve a very good detection quality by combining a very good noise reduction with a more inaccurate pattern detection. These solutions incorporate a completely different use of the existing virus detection methods. That these extremely useful solutions could be found shows clearly the advantage of MOGEA-DSE.

With linear regression, three models could be learned to predict the energy consumption, execution time, and F_1 score of the Pareto front. Some additional insights could be gained from these models. They facilitate the interpretation of the individuals on the Pareto front and show which parameters might have an important influence on the objectives.

To examine if a model-based optimization could be feasible as an extension of MOGEA-DSE, additional machine learning approaches have been inspected on all data and not only on the front. The methods linear regression, multilayer perceptron, SVM for regression, Random Forest, and M5P have been evaluated for a prediction of the three objectives. Somewhat surprising is the bad performance of the SVM for regression, which is, in these experiments, worse than that for linear regression on almost all quality measures and is also the slowest method. Especially the tree-based methods show good performance on the prediction of execution time and energy consumption. A model-based optimization could be feasible and could lead to a faster evaluation with MOGEA-DSE. Individuals that show a good performance, based on the prediction with the models, can be preferred. Overall, this is a promising approach for future work.

To sum up, MOGEA-DSE has been able to decrease the execution time and energy consumption of VirusDetectionCL drastically. By keeping the detection quality of 1.0, a speedup of 4.1 with energy savings of 77 % to 84 % could be achieved. This exceeded all expectations. With a slightly reduced QoR of 0.969, a speedup of 8.1 with energy savings of 93 % could be achieved. By allowing a larger drop in QoR, e.g., to an F_1 score down to 0.5, which is still feasible for some use cases, speedups up to 12 and energy savings of up to 94 % are possible.

MOGEA-DSE is also able to achieve a large diversity of solutions in the parameter and design space. Both, the Pareto front and the overall search space are densely sampled. The resulting Pareto fronts for Exp11$_{\text{SW}}$ and Exp12$_{\text{HW\&SW}}$ are composed of several solutions with very different performance characteristics. As a consequence, MOGEA-DSE is able to explore the parameter and design space very well.

For the model-based prediction of the three objectives with five machine learning approaches, it should be emphasized that the prediction of the F_1 score only achieved a correlation coefficient of 77.8 % in the best case. This indicates that the overall optimization task is not easy. For a mere model-based optimization, it could be very difficult to optimize the detection quality. Execution time and energy consumption could be predicted quite well. For these objectives a model-based optimization is feasible.

Based on the optimized hardware and software configurations, the use of the Odroid-XU3 can be recommended without restrictions. This enables the construction of a fast, energy efficient, hand-held CPS for the detection of viruses. The system can be adapted to different scenarios by using a suitable solution from the found Pareto front. With approximated computing larger energy savings are possible, which can be useful if the system is running low on battery.

8.5.6 Summary and Conclusion

In this section the MOGEA-DSE method has been extended to a multi-objective energy-aware hybrid PSE and DSE on ESs. Hardware and software configuration have been jointly considered for the Odroid ES and actual measurements have been performed. Additionally, approximated computing and the use of model-based prediction has been explored.

With the Energymeter, measurements of the Odroid system and dynamic frequency scaling for the Arm®big.LITTLE CPUs has been made possible. The Energymeter could be synced with VirusDetectionCL to only measure the main processing pipeline and not the pre-processing by using signals.

This resulted in a highlight for the MOGEA-DSE method. A small hand-held device for the processing of PAMONO sensor data could be automatically explored in a hardware/software codesign manner. The virus detection software and the Odroid ES could be optimized to make a fast, mobile, and energy efficient use of a PAMONO CPS possible. Without the use of approximate computing the soft real-time limit of 30 fps to meet the speed of the camera could be achieved. A speedup of 4.1 and an energy saving of 84 % could be achieved compared to the baseline measurement. These results alone exceeded all expectations and are clearly a highlight. A time-consuming manual tuning of the hardware and software parameters was not needed.

The use of approximate computing in MOGEA-DSE for the evaluated programs has shown to be very beneficial. By additionally making use of approximate computing, higher frame rates and larger energy savings were possible. With only a slight reduction in QoR, the speedup could be nearly doubled and 93 % energy could be saved. By allowing a further reduction in QoR, speedups up to 12 were possible. These results are both very useful in practice. A slight reduction in detection quality can be approved to increase the number of samples that can be analyzed if the device is running on battery. The large reduction in QoR is useful if the actual number of viruses is of no importance and only a virus test for positive or negative detection result is needed. However, it has to be carefully verified if precision and recall, cf. Definition 35 and Definition 36, are both reasonable for such a test.

For the research issue if model-based prediction of the objectives can be used to speed up the evaluation, it could be shown that not all objectives can be predicted accurately. A prediction of execution time and energy consumption is feasible for the inspected use case. For these objectives a model-based prediction, for example with a Random Forest, might be an opportunity to speed up the optimization by skipping individuals with low predicted values or even guide the optimization by creating promising individuals. However, for the detection quality only a relatively low correlation could be achieved.

8.6 Summary and Conclusion of the MOGEA-DSE Method

In this chapter the MOGEA-DSE method has been presented as the most advanced PSE and DSE method in this thesis. MOGEA-DSE is the further development of SOG-PSE and SOG-DSE from Chapter 7. It is a hybrid PSE and DSE method that can be used for a multi-objective exploration of hardware and software, e.g., for CPSs in different PAMONO scenarios. An exploration of the target hardware can either be performed simulation-based or measurement-based.

Figure 8.16 shows the complete MOGEA-DSE approach with all parts of MOGEA-DSE from this chapter in one schematic. Ten inputs are possible: the parameter and design space, the objectives, one or more fitness evaluators, program dependencies, file dependencies, program parameters, an EA configuration, a GPU configuration, setup information for the remote evaluation, and an LTE configuration.

These inputs are managed in six databases: for parameter space and design space definition, dependencies, parameters, GPU configurations, remote access data, and the LTE parameters with the LTE environment. One additional database is used to manage the evaluated results. Within the main evaluation the evaluation manager drives the evaluation and runs the EA cycle, the client manager connects the available clients, and the GPU code compiler can be used to set up the PTX code for the simulated GPUs if a compiler is not available on the remote evaluation device. The remote evaluation consists of the fitness evaluation, a GPU simulator, actual GPU hardware, the Energymeter

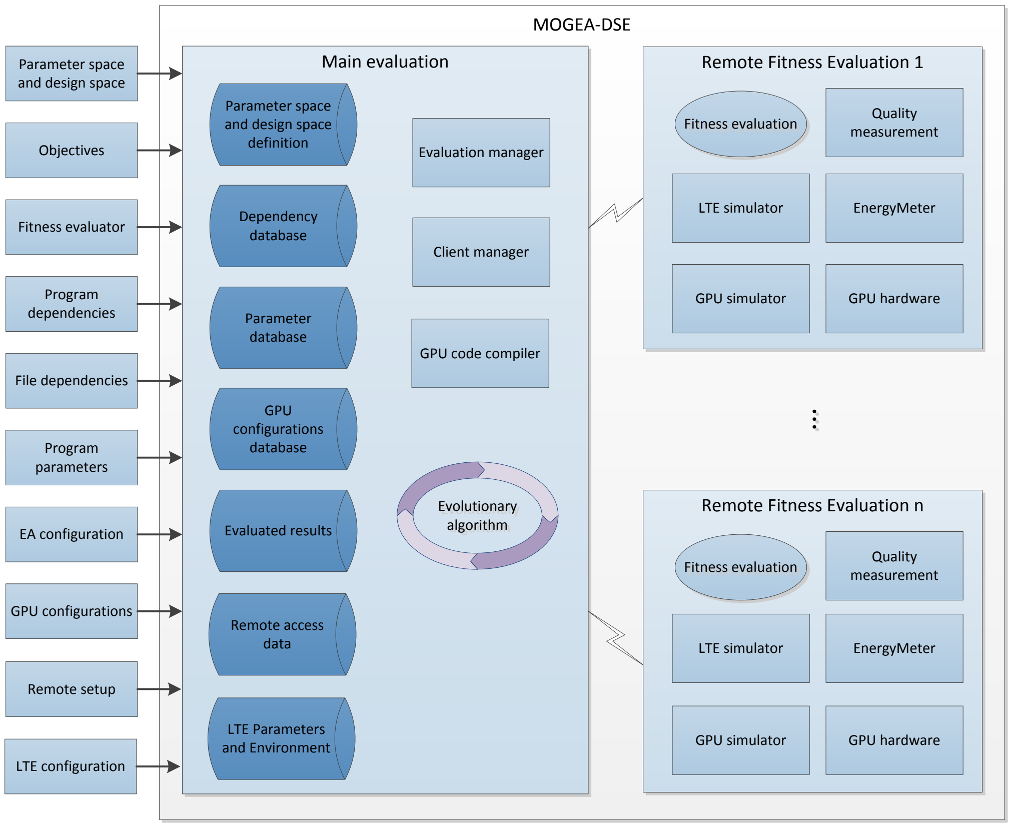


Figure 8.16: The complete MOGEA-DSE approach for a hybrid PSE and DSE of software and actual GPU architectures, simulation of GPU architectures, and the exploration of offloading capabilities. Additionally, approximated computing can be integrated by using several quality measurements.

for the measurement of actual GPUs, a quality measurement for the QoR output, and an LTE simulator to simulate offloading.

By combining PSE and DSE in MOGEA-DSE, a simultaneous exploration of hardware and software has been made possible. The conducted experiments in a hardware/software codesign manner showed that such a hybrid PSE and DSE is very useful for the explored CPS. Considering hardware and software jointly this led to better results for energy consumption and execution time.

Two multi-objective EAs, SPEA-2 and NSGA-II, have been integrated to enable multi-objective optimization. This is especially important for the hybrid PSE and DSE as for the software other objectives can be optimized as for the hardware. By integrating two power-aware GPU simulators into MOGEA-DSE, GPGPU-Sim in conjunction GPU-Wattch and the GPUSimPow simulator it could be shown that MOGEA-DSE is not limited to a specific simulator.

A proper handling of the parameter and design space has been introduced with MOGEA-DSE. This includes that the search space can be kept as small as possibly by take advantage of dependencies. These dependencies could be handled by deriving the dependent software and hardware parameters from other parameters and genes. To handle also very difficult dependencies, JavaScript®could be used and references to other parameters and genes could be automatically resolved. This has shown to be very useful for modeling the design space of simulated GPUs.

The *checkpointing* and *quick start* methods can be used to speed up the evaluation without losing accuracy in the result. With the *checkpointing* method buffers can be initialized faster by using pre-calculated results. And with the *quick start* method queues can be initialized faster. Both can shorten the initialization phase of GPGPU programs.

Additionally, an exploration of different offloading strategies has been introduced for MOGEA-DSE. VirusDetectionCL has been extended to process the sensor data up to a certain extent. The remaining work can be offloaded to other devices in a local network or to the cloud where it is proceed by a compute server. To simulate the data transfer, the CoPoMo LTE model has been integrated in MOGEA-DSE. This enabled a VirusDetectionCL has also been extended to enable an actually offloading the work instead of using the simulator.

The Energymeter has been developed for the Odroid. The Energymeter enables the measurement of the CPUs, the GPU, and the RAM of the Odroid. With a tight coupling with VirusDetectionCL, the initialization phase of VirusDetectionCL could be excluded from the measurements. As a result, less frames need to be processed to achieve reliable results. An exploration of dynamic frequency scaling has also been enabled by the Energymeter. The Odroid CPUs could be run with different governors or at a fixed frequency.

This enabled a combined optimization of a hand-held device and the parametrization of VirusDetectionCL. For VirusDetectionCL also several hardware related parameters and the pipeline configuration with algorithmic alternatives have been optimized. As a result, a fast, mobile, energy efficient use of a hand-held device has been made possible for the PAMONO use case.

The model-based prediction of the objectives with different models showed promising results. The prediction of energy consumption and execution time is feasible and can be useful to either identify individuals that do not have to be evaluated or to even guide the optimization process. However, for the detection quality only a relatively low correlation could be achieved. This strongly indicates the difficulty of such a PSE and DSE and it motivates why the optimization with EAs is useful in MOGEA-DSE.

The goal for the MOGEA-DSE method could be achieved: to explore CPSs with HPC, desktop, laptop, and hand-held devices as target architectures and to achieve a rapid and reliable detection and counting of viruses in PAMONO sensor data on the target architectures. Very well suited software and hardware configurations for CPSs of all considered PAMONO scenarios could be found. As MOGEA-DSE is the last method in this thesis, this leads to the overall conclusion and future work in the next chapter.

Chapter 9

Conclusion and Future Work

Contents

9.1 Conclusion **231**
9.1.1 Conclusion for VirusDetectionCL 232
9.1.2 Conclusion for RFC-Gen 232
9.1.3 Conclusion for SOG-PSE and SOG-DSE 233
9.1.4 Conclusion for MOGEA-DSE 233
9.1.5 Broader Context 234
9.1.6 Limitations 235
9.1.7 Further Use 235
9.2 Future Work **236**
9.2.1 Proposal of the MP-MSP430-Sim Approach 238
9.2.2 Proposal of the GOOM Approach 239

In this chapter the global conclusions and future work of this thesis is given. The conclusion in Section 9.1 includes the contributions and conclusions for the methods from this work, a broader context of what can be done with these methods, some limitations, and further use of the methods by other researchers. Because more detailed conclusions have been provided throughout this thesis, the conclusion focuses on the most important contributions. Finally, the future work in Section 9.2 includes preliminary work regarding the modeling of larger sensor networks and regarding the integration of offloading capabilities into several types of GPGPU programs.

9.1 Conclusion

The following five methods have been presented in this thesis:

- VirusDetectionCL, a computer vision-based method for detecting and counting viruses in PAMONO sensor data,
- RFC-Gen, a method for automatic code generation for the application of Random Forest models making them executable on GPUs,
- SOG-PSE, a method for a single-objective exploration of parameters toward one objective,
- SOG-DSE, a method for a single-objective, simulation-based DSE of GPU architectures, and
- MOGEA-DSE, a hybrid PSE and DSE method for a multi-objective, simulation-based or measurement-based exploration of CPSs of all kind of sizes.

All these methods have been motivated by practical problems of the PAMONO sensor use case. The common thread running through this thesis and linking all methods are different scenarios of how the PAMONO sensor can be used. Using the PAMONO sensor in laboratories, hospitals, airports, and in mobile scenarios has been considered. For these scenarios a wide range of target architectures has been explored. Each target architecture had to fulfill different requirements.

With MOGEA-DSE, VirusDetectionCL, and RFC-Gen the overall goal could be achieved: to develop methods that explore and configure CPSs for a rapid, reliable detection and counting of viruses in PAMONO sensor data with HPC, desktop, laptop, and hand-held devices as target architectures for all considered PAMONO scenarios. This includes various methods for a computer vision-based virus detection and methods for the exploration of CPSs regarding suitable target hardware with suitable software and hardware configurations.

9.1.1 Conclusion for VirusDetectionCL

For detecting viruses in PAMONO sensor images with the VirusDetectionCL method, three major contributions could be achieved.

The first and most important contribution of VirusDetectionCL is a fast and accurate detection and counting of individual viruses with hand-held devices. This contribution opens up completely new possibilities for a mobile virus detection indoors and outdoors. As Mairhofer, Roppert, and Ertl stated, there is a huge demand for fast, portable, accurate diagnostic methods [MRE09]. This has been made possible for biological virus detection with VirusDetectionCL and the PAMONO sensor.

The second contribution is that with VirusDetectionCL, sensor data with low SNR can be handled. In processing methods for similar data, in the context of particle tracking in fluorescence microscopy data, an SNR of two to four [CWG01; SLN+09] is difficult to handle. For VirusDetectionCL sensor signals with a median SNR slightly below two can be handled with the noise reduction, feature extraction, and classification methods. As a result, the detection of individual viruses down to 100 nm with a reasonable detection quality is possible.

The third contribution is that the diagnostic gap could be further reduced with VirusDetectionCL by evaluating the data set of a sample in less than three minutes. An expert needs a few hours or even days for a manual evaluation of the data set of one experiment.

9.1.2 Conclusion for RFC-Gen

The RFC-Gen methods enables an automatic generation of optimized code for the application of Random Forest models that can be run on the GPU. This method was used for VirusDetectionCL. With a given output of VirusDetectionCL, consisting of features and detected virus candidates, a Random Forest model can be trained and translated into GPGPU code if a correct classification is known. This Random Forest model can be looped back into VirusDetectionCL.

RFC-Gen generates GPGPU code for the application of the model, host code to call this model application from within VirusDetectionCL, and an automatic configuration for VirusDetectionCL to activate only the calculation of the actually required features. As a result, the Random Forest model can be used to classify virus candidates. And because of the automatic code optimization, the application of the model is fast.

9.1.3 Conclusion for SOG-PSE and SOG-DSE

The SOG-PSE and SOG-DSE methods and the experiments conducted with these are the basis for the MOGEA-DSE method. Several insights could be obtained.

The most important contribution from SOG-PSE is that it can provide, on a regular basis, feedback of the QoR that is achievable with VirusDetectionCL and the current PAMONO sensor setup. This strongly influenced and guided the direction of research and developments both for VirusDetectionCL and the PAMONO sensor. Bottlenecks and hidden potential could be uncovered and insights could be gained early and frequently throughout the whole process. With every major change, e.g., in the PAMONO sensor, the quality or characteristic of the sensor data, size of the viruses, or in VirusDetectionCL, SOG-PSE can tune the parameters and reevaluate the QoR. No manual tuning is necessary to adapt the parameters to the changed conditions.

Another major contribution is that better detection results could be achieved than with manual tuning of the parameters with expert knowledge. For example, for three experiments the detection quality could be improved from 45 %, 20 %, and 74 % accuracy for the manually tuned parameters to 88 %, 80 %, and 86 % accuracy for the automatically tuned parameters. For all experiments where the detection quality was optimized, the automatic exploration of parameters with SOG-PSE reached or exceeded the quality of manually tuned parameters.

With SOG-DSE suitable GPU hardware can be explored. The most important contribution from SOG-DSE is that an automatic, simulation-based exploration of GPU architectures has been made possible. To achieve this, a GPU simulator and a GPU power model have been integrated into SOG-DSE. A design space of GPU architectures is defined by the user and SOG-DSE can automatically explore it.

As a result, GPUs for desktop and HPC systems could be explored for 14 benchmark programs. Energy and power consumption have been used as objectives. With the experiments it could be shown that the fastest GPU was not always the most energy efficient and that the slowest GPU was not always the most power efficient.

However, SOG-PSE and SOG-DSE showed some limitations, e.g., that parameters and hardware can only be explored in succession, that the simulation of GPU architectures is very time-consuming, and no actual measurements of hardware are supported. These limitations led to the MOGEA-DSE approach.

9.1.4 Conclusion for MOGEA-DSE

With MOGEA-DSE various CPSs for the PAMONO sensor use case could be explored and several contributions could be achieved. This includes, but is not limited to, the exploration of CPSs for the PAMONO sensor use case. All parts of the system profited from the exploration. For VirusDetectionCL a high detection quality could be achieved, well suited processing hardware could be found, and feedback could be provided for further improvements of the PAMONO sensor.

MOGEA-DSE can explore GPU architectures for HPC servers, desktop systems, mobile systems, down to ESs. The most important contribution of MOGEA-DSE is that the soft real-time processing of VirusDetectionCL on ESs has been made possible. This enables a fast, mobile virus detection with small, hand-held devices. On the Odroid system a speedup of 4.1 and an energy saving of 84 % could be achieved without reducing the detection quality.

Other important contributions are that the MOGEA-DSE supports a hardware/software codesign development process, a multiple-objective exploration, integrates two GPU simulators for the simulation of GPUs, can simulate offloading via the LTE network, integrates approximate computing, and can perform measurements of actual ES hardware.

MOGEA-DSE is well suited to guide and accelerate the development of CPSs toward different target architectures and to adapt them to changed conditions. This has been shown for the PAMONO use case. With regular improvements of the PAMONO sensor, the VirusDetectionCL method and changed target architectures, the system has to adapted quite often. The target architectures were extended from desktop, to mobile, to hand-held devices, and to HPC systems with offloading from mobile devices. MOGEA-DSE enabled to automatically explore suitable hardware and to adapt the whole system to the changed conditions.

Overall, MOGEA-DSE has shown to be very useful for the exploration of CPSs for all considered PAMONO scenarios. Along with the exploration of 20 benchmark programs and several GPU architectures from high performance down to an ES GPU it could be shown that MOGEA-DSE is useful for a wide range of scenarios.

9.1.5 Broader Context

Several new possibilities for a virus detection with the PAMONO sensor opened up with the research on the VirusDetectionCL, SOG-PSE, SOG-DSE, and MOGEA-DSE methods.

For an early version of VirusDetectionCL and without the PSE and DSE methods, only a laboratory use on desktop hardware was feasible [SWL+11]. Then, with further advances in VirusDetectionCL, SOG-PSE, SOG-DSE, and MOGEA-DSE, a mobile use in laboratories, hospitals, and airports on laptop hardware became feasible [LST+13a], followed by a mobile use with offloading capabilities [LKD+14] in laboratories, hospitals, airports, and outdoors on laptop hardware and additional processing on HPC hardware [LSW14; LMS+14]. Finally, a mobile, stand-alone use on small ES hardware has been made possible [NLE+15]. This enables a use of the PAMONO sensor virtually everywhere as no data connection is needed. With further miniaturization of the PAMONO sensor a small, hand-held detector is possible.

It is very difficult to foresee every last implication of these developments. With a rapid detection and hand-held outdoor use, it is possible to identify the source of a virus outbreak. With several PAMONO devices hundreds of samples can be evaluated in minutes and a virus detection at places with a high passenger volume such as airports is possible. An automatic evaluation of many samples is also useful in pharmaceutical research for developing new antiviral drugs. Of additional interest might be, that the binding process of individual viruses is visible even during the processing. Because of the very fast evaluation of samples, prophylactic or acute virus tests in hospitals or doctor's offices are possible.

Additionally, the fast processing methods from VirusDetectionCL might also be useful in the context of cell tracking and fluorescence microscopy. The involved processing tasks and the sensor images share several characteristics.

The methods SOG-PSE, SOG-DSE, and MOGEA-DSE are also useful in a much broader context. Exploration of hardware architectures and software parameters is important for many other use cases. Hardware architectures from HPC systems, desktop, laptop, down to ESs have been explored for a wide range of biological, physical, and industrial applications. This opens up a wide range of application fields for the methods.

In addition, MOGEA-DSE is able to generate more useful information that has not been fully utilized. For example, with the calculation of the utilization of the GPU in the GPU simulator or a detailed per kernel statistic, application designers can easily identify and remedy bottlenecks on a system level. MOGEA-DSE can also be used for the design and development of new GPU architectures. With the exploration of devised GPUs, this large potential for improvements has only been slightly indicated.

9.1.6 Limitations

During the research on the methods in this thesis, some limitations and boundaries have been revealed. For the automatic virus detection with the PAMONO sensor combined with the VirusDetectionCL method, the SNR of the sensor images seems to be the most limiting factor. Currently, a median SNR of two and slightly below can be handled. As a result, viruses down to 100 nm can be detected with a reasonable detection quality. For data sets with a good quality with low noise and few artifacts, a detection of viruses down to 50 nm is feasible. To make this detection applicable in practice, the physical PAMONO sensor setup and VirusDetectionCL need to be improved.

A major limitation for the simulation-based approaches in SOG-DSE and MOGEA-DSE is the accuracy of the used simulators. Simulators might contain modeling errors and might be overfitted to certain architectures [NMH+14]. This can be particularly adverse if devised architectures are simulated. But also for existing architectures, some assumptions have to be made in the simulator because the architecture design is often not publicly available. Important details might be kept secret by the manufacturer. The used simulators get around this limitation by calibrating the simulation with measurements on existing hardware, which on the other hand might overfit the simulator to the evaluated architectures. For devised architectures this kind of calibration is not possible.

Another limitation is the still time-consuming simulation of GPUs. A model-based prediction of objectives has been inspected. This possibly enhance the speed of the DSE. It could be shown that for the objectives execution time and energy consumption, a model-based prediction is feasible. However, the detection quality could not be predicted very well because of the highly non-linear influences in the parameter space. Very different configurations can lead to the same detection quality and parameters of early pipeline stages can affect later pipeline stages. This clearly motivates the use of EAs.

9.1.7 Further Use

The methods from this work have also found use in other works and have opened up new directions in research for other researchers.

The VirusDetectionCL method was used by Timm as a use case in his Ph.D. thesis [Tim12], by Shpacovitch et al. to perform a specificity measurement of the PAMONO sensor [STM+15], and by Popovic for research on offloading in her bachelor thesis [Pop16].

For the latter, optimized hardware and software parameters, that have been generated with MOGEA-DSE, were used.

The VirusDetectionCL software is also used for further research by Lenssen [Len17]. He integrated Convolutional Neural Networks (CNNs) into the VirusDetectionCL approach. MOGEA-DSE can be useful to bring his CNNs to the Odroid ES as they currently only meet the soft real-time constraint on a fast desktop GPU when using appropriate image sizes and network parameters.

The Energymeter developed and used for MOGEA-DSE and the VirusDetectionCL method will be further used by Neugebauer [Neu17] for his research. Approximate computing for parallelization strategies on resource constricted ESs will be considered. VirusDetectionCL will serve as an important use case and the Energymeter will be used for further measurements.

9.2 Future Work

The research for this thesis opened up some new questions, challenges, and opportunities for future work. These are listed according to the structure of this thesis. The future work also includes to proposals of two approaches that fit in the context of this work.

As future work for the virus detection, it is planned that the PAMONO sensor is extended to detect more than one virus stem at a time. For this approach the PAMONO sensor will be equipped with a larger sensor surface that is divided into different areas, each covered with different antibodies to detect one type of virus or one type of virus stem. The new challenge with this approach is that the image sizes must be adapted to the larger sensor area, which is more demanding for the target platform and that different parameters must be explored for the areas. One opportunity for the detection software and a PSE and DSE method like MOGEA-DSE is an automatic adaptation of the detection quality, energy consumption and execution time, independently to each single area of the sensor. If viruses appear only in one part of the sensor images, this area can be inspected with an increased detection quality. While in the other areas a low quality of service might be enough, to not miss the moment where particles start to appear and the detection quality needs to be adapted.

While collecting and perusing the related work of the PAMONO sensor, a promising opportunity for further improvements of the sensor could be found. As Naimushin et al. have indicated [NSB+03], there might be a temperature range for antibody and antigen binding within a SPR sensor for which a higher binding rate can be obtained. In their study they could double the binding rate. Even if the inspected particles differ, the optimal detection temperature for the PAMONO sensor should be further investigated. For the PAMONO sensor and the VirusDetectionCL application, there might be a temperature range at which the sensor and the detection software produce better results in less amount of time. There might also exist a trade-off between binding rate and detection quality such that a faster binding rate results in faster detection results with less energy consumption but also less accuracy and vice versa. Therefore, the sensor temperature should be included in the optimization and should be chosen automatically by the software or manually by the user depending on the desired objective.

For an accurate detection of viruses of 50 nm and below, further research is needed on VirusDetectionCL method and the physical PAMONO sensor setup. The SNR seems to be the most limiting factor to make smaller sizes detectable with sufficient detection quality. For the PAMONO sensor setup the camera and lenses offer most opportunities for increasing the SNR. For the camera a CCD chip with larger pixel capacities can be used. Because the light source in the sensor setup can be further increased, which results in larger signal, only the pixel full well capacity seems to limit a better SNR. For the lens setup the region of perfectly in focus viruses should be increased. Only these viruses offer the largest SNR. The more out of focus the particles are the lesser the SNR is because the signal is distributed to a larger area. Smaller apertures or a different orientation of lens, gold layer, and camera might improve the overall sharpness. For VirusDetectionCL it is supposed that several methods need to be improved to detect smaller viruses. This includes the methods for noise reduction, signal restoration, and feature extraction. As preliminary examined by Lenssen [Len17], the use of CNNs might be beneficial.

For the methods SOG-PSE, SOG-DSE, and MOGEA-DSE, the optimization with EAs can be replaced or extended with other optimization methods. As the experiments in Section 8.5.4 have shown, a model-based optimization may be suitable to replace or extend the EAs. For the objectives energy consumption and execution time, this is possible now. But for the objective detection quality further research has to be done as a prediction of this objective is difficult.

For SOG-DSE and MOGEA-DSE it might be useful to also simulate the CPU and not only the GPU of a system. For the measurement-based experiments in this work, the CPU has been part of the optimization but not for the simulation-based. To achieve this, for example, the GEM5 simulator [BBB+11] can be integrated into SOG-DSE and MOGEA-DSE. With this simulator, ARM®and *x86* CPUs can be simulated. If this simulator is combined with a power model, e.g., from McPAT [SAS+09], also energy and power consumption can be integrated. This is especially useful for mobile systems but can also be useful for HPC systems.

An exciting development for HPC systems is that recently small MPSoCs are used for processing in these large systems. Rajovic et al. have shown how several hundred SAMSUNG®Exynos MPSoCs with an integrated Mali GPU can be used to build an energy efficient HPC cluster [RCG+13]. MOGEA-DSE can be used to explore and optimize these systems and even further reduce the energy consumption. However, new challenges have to be handled by MOGEA-DSE: For an HPC cluster, a large amount of sub-systems and the communication between these has to be considered.

To explore cooperative homogeneous and heterogeneous networks of PAMONO devices, is also part of future research. This includes the modeling of larger sensor networks and offloading to other ESs or other mobile devices in the area via a BLUETOOTH®or W-LAN connection. These connection channels have not been evaluated in this work. Some research in this direction has already been done in [Pop16].

Some preliminary work has already been done toward modeling of larger sensor networks and toward offloading, which may open up new opportunities for further research. In the following the MASSIVE PARALLEL MSP430 SIMULATOR (MP-MSP430-Sim) approach and the GPGPU OPENCL OFFLOADING MIDDLEWARE (GOOM) approach are presented. The MP-MSP430-Sim approach has been developed for modeling of larger sensor networks, and the GOOM approach is proposed for an easy integration of powerful offloading techniques to all kind of existing GPGPU programs without the need to change them.

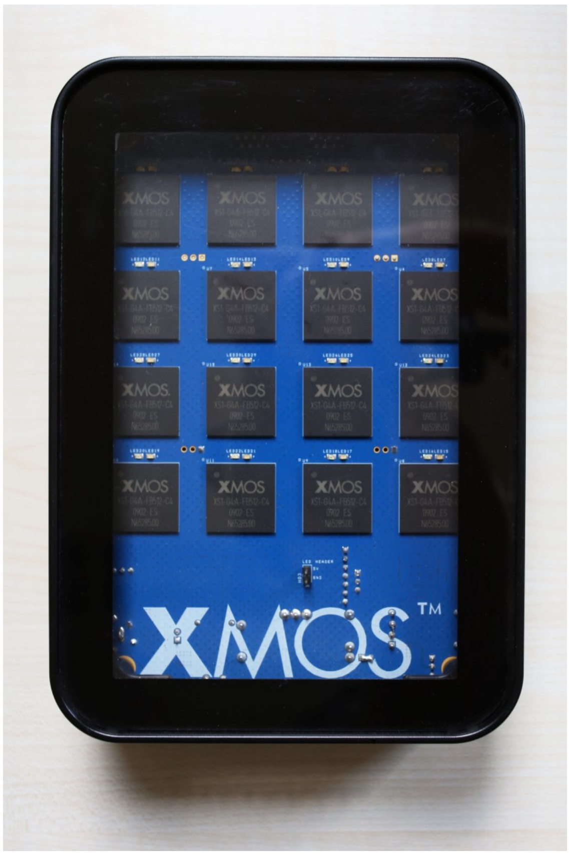


Figure 9.1: The XMOS®XK-XMP-64 platform with an array of 4 × 4 XS1-G4 quad core CPUs. The size of the housing is 192 mm × 132 mm × 46 mm.

9.2.1 Proposal of the MP-MSP430-Sim Approach

In this section a proposal of an approach for the parallel simulation of sensor nodes is drafted. A parallel simulation of sensor nodes is useful to simulate large sensor networks. The approach is called Massive Parallel MSP430 Simulator (MP-MSP430-Sim). It can be used to simulate several hundred MSP430 sensor nodes in parallel on XMOS®hardware. The MSP430 is a low power ES by Texas Instruments®. This micro-controller is a popular device for various sensor applications.

The main idea of the MP-MSP430-Sim approach is to simulate a whole network of sensors on hardware that enables a fast communication between the simulated sensor nodes. In the future, this approach can be extended to simulate a network of PAMONO sensors that distribute work to each other and share resources as depicted in Figure 1.1.

To simulate an MSP430 sensor node, MP-MSP430-Sim uses naken430asm [Koh11] a MSP430 assembler, disassembler, and simulator. For each sensor node a source code is loaded, which is first assembled, then loaded onto a core on the hardware, and finally executed with the simulator.

To run the simulations MP-MSP430-Sim can use the XMOS®devices as processing hardware. Both, the XMOS®XK-XMP-64 and the XMOS®XC-1 hardware can be used. These devices use Reduced Instruction Set Computing (RISC) microprocessors and can run several simulations in parallel and enable a fast communication.

To simulate MSP430 applications, the XMOS®XK-XMP-64 boards [Han11] can be used. It is shown in Figure 9.1 and has a housing of 192 mm × 132 mm × 46 mm. The XK-XMP-64 has a 16 quad core XS1-G4 CPU with 64 cores in total. The cores are arranged in a 4 × 4 array and are connected in a 4-dimensional hypercube topology with a round trip time of 200 ns to 1 µs [XMO15]. This enables fast communication between the cores.

To fully utilize the 64 cores, at least 256 threads are needed because the pipeline stages decode, read, execute, and write need four active threads to be utilized. Therefore, the XK-XMP-64 can simulate up to 256 MSP430 applications at full speed.

To simulate more than 256 MSP430 applications, several XMOS®XK-XMP-64 boards can be combined. The XK-XMP-64 board offers connectors to simply plug boards together in a two-dimensional array.

To simulate up to 16 MSP430 applications at full speed, the XMOS®XC-1 board can be used. The XC-1 has a quad core XS1-G4 CPU [May09]. The board is 85 mm × 54 mm in size, about the size of a credit card.

To conclude, MP-MSP430-Sim is an approach for simulating several hundred sensor nodes. As proof of concept, it has been implemented to simulate MSP430 sensor nodes on XMOS®hardware. Theoretical, the communication between the sensor nodes can be easily integrated due to the hardware properties of the XMOS®hardware, which offers a fast communication between the cores, but there are still several challenges to be solved to make this possible. To be useful for the PAMONO sensor use case, GPGPU hardware would need to be simulated. However, this would need much simulation time and might even not be possible on the XMOS®hardware as the MSP430 simulation is already challenging.

9.2.2 Proposal of the GOOM Approach

For future research on offloading of OpenCL programs, the proposal of the GPGPU OpenCL Offloading Middleware (GOOM) approach is drafted. This approach includes an offloading interface to offload GPGPU tasks to other devices. Existing GPGPU programs can extended to make use of offloading simply by linking an OpenCL interface of the presented approach. As the OpenCL *C/C++* API [Khr11b] offers a tight interface, it is well suited to integrate offloading approaches.

For the GOOM approach the main idea of the architectural design is that all OpenCL commands that are usually directed to the GPU driver on the local device are received by the GOOM approach. The GOOM approach redirects the commands and needed memory buffers to one or more remote devices. The OpenCL program is executed on the remote device and afterward results are transferred back to the local device. All memory allocations on the GPU are made on the remote devices. Only the OpenCL memory pointers need to be available on the local device and can be used there without knowing that the memory resides on the GPU of the remote device. Only if the memory is accessed by the CPU on the local device, the memory needs to be transferred. Because every CPU access to the GPU memory requires that the memory is transferred from the GPU with an OpenCL command, this can be detected and handled appropriately with GOOM.

Figure 9.2 shows a schematic of the GOOM approach. In the first layer a GPGPU program simply links to the OpenCL interface. Any GPGPU program can use this interface as if a local OpenCL device is used. The remaining layers are hidden. The GOOM layer

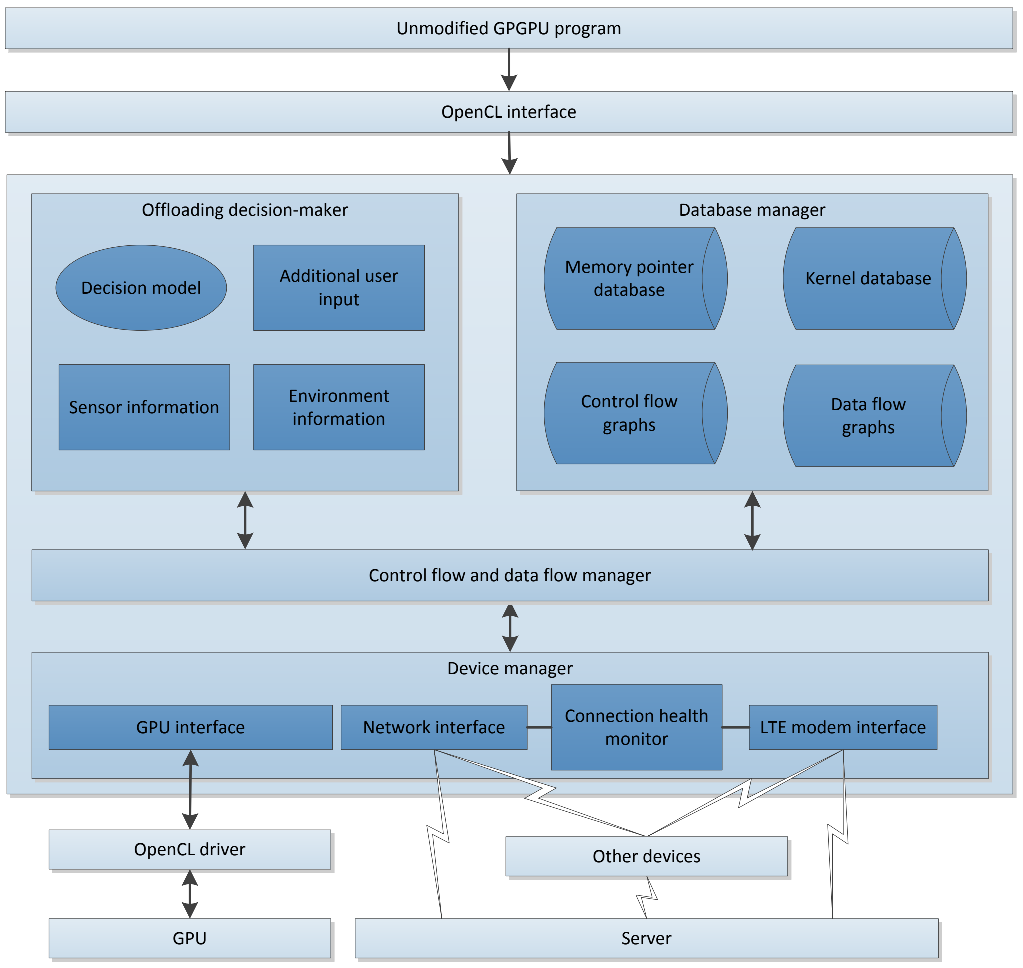


Figure 9.2: The GOOM offloading approach for offloading GPGPU tasks to several devices, e.g., from an embedded system to a server or other embedded systems available in the area. An unmodified OpenCL GPGPU program can simply be linked to GOOM to incorporate offloading.

consists of an offloading decision-maker, a database manager, a control flow and data flow manager, and a device manager. The offloading decision-maker uses a trained decision model. With this model, a decision can be made whether and what parts of the task should be offloaded. The decision can be based on sensor information, environment information, and additional user input. The database manager keeps track of all allocated memory pointers, the available OpenCL kernels including their source code, control flow graphs, and data flow graphs. The control flow and data flow manager guides the direction of control and data throughout the modules in GOOM. It also builds and adapts the control flow and data flow graphs in the database manager. The device manager organizes the available devices. This includes local GPU devices that are available through a local GPU interface and the remote devices that are available through a network connection, e.g., Local Area Network (LAN), wireless LAN, or an LTE connection. The connection health might also be observed by the device manager. It also provides the interface to the device layer to all local and remote GPUs.

With the GOOM approach virtually any existing GPU program that interfaces with OpenCL can be offloaded without the need to modify the program. If no remote devices are available or if the result is needed fast, OpenCL commands are forwarded to a local OpenCL driver and the work is processed on the local GPU or CPU. If, on the other hand, the work should be offloaded, OpenCL commands and needed local memory buffers are transferred to the remote device. The remote device processes the OpenCL commands on available OpenCL devices. Allocated GPU buffers can reside on the remote device and can be reused. If results are needed on the local device the buffers are downloaded from the remote GPU to the remote device and transferred back to the local device.

This approach opens up several possibilities for dynamic offloading. Dynamically changing the offloading conditions is possible because at all times during processing, the location of any buffer on the GPU is known and full control over the control flow and data flow is given. For example, if the battery on a mobile device runs low, the OpenCL pipeline can be stalled and all data and OpenCL kernels can be offloaded to a server. Then, an OpenCL pipeline is set up and partially calculated results are loaded to the GPU. Subsequently, all calculations can be offloaded from the mobile device to the server.

A dynamic change from offloading of work to local processing is also possible. If a calculation is offloaded to a server and should be continued locally, all buffers can be downloaded from the GPU of the server, transferred to the local device and used there on the local GPU. This can be useful if a device without sufficient wireless connection to the server should be disconnected from the wired connection or if a wireless connection is expected to be reduced in quality, e.g., by leaving a certain area. After all data is transferred back to the local device, it can be disconnected from the network and the calculation can be continued locally without losing any results. The same is useful if a power supply becomes available and the local processing is faster.

Additionally, this approach opens up opportunities for partial offloading. With the control flow graph and the data flow graph, an offloading point can be determined automatically. For example, it can be determined that after some pre-processing the least amount of data needs to be transferred and that all subsequent calculations can be done on the server. The dynamic and partial offloading can be combined. Dynamic changes in the control flow or data flow graph can change the partial offloading.

To sum up, the GOOM approach provides offloading capabilities for GPGPU tasks with a narrow interface. Complexity can be hidden from the user and appropriate offloading decisions can be made on a wide information basis. Dynamic offloading decisions can be combined with full or partial offloading.

Acronyms

ACO	Ant Colony Optimization 31
ALU	Arithmetic Logic Unit 31
API	Application Programming Interface 139, 140, 241
CACTI	Cache Access and Cycle TIme model 137
CCD	Charge Coupled Device 17, 18, 74, 238
CMA-ES	Evolutionary Strategy with Covariance Matrix Adaptation 29
CMOS	Complementary Metal-Oxide-Semiconductor 17, 18
CNN	Convolutional Neural Network 237, 238
CNSGA-II	Controlled Elitist Non-dominated Sorting Genetic Algorithm-II 31
CoPoMo	COntext-aware POwer consumption MOdel 190–196, 199, 230
CPS	Cyber-Physical System 2–5, 7, 9, 30, 33, 60, 117, 156, 158, 189, 201, 208, 225, 228–231, 233–235
CPU	Central Processing Unit 1, 27, 31–34, 37, 39, 45, 77, 106, 122, 128, 129, 131–137, 144, 148, 152, 181, 193, 196, 200, 202, 204–206, 208, 217, 222, 226, 228, 230, 238–240, 242
CUDA	Compute Unified Device Architecture 27, 28, 135, 136, 138, 139, 143, 144, 178, 179
DCPS	Distributed Cyber-Physical System 189, 190
DFS	Dynamic Frequency Scaling 205
DNA	Deoxyribonucleic acid 24, 25, 143, 144, 178, 186
DRAM	Dynamic Random Access Memory 40, 135, 136, 141, 142, 144–146, 149, 150, 152, 163, 165, 166, 179–181, 185, 187, 188, 195, 274, 275
DSE	Design Space Exploration 2–11, 13, 23, 28, 30–35, 53, 67, 106, 116, 117, 122, 129, 133–135, 138, 145–148, 150–153, 156–158, 160–162, 172, 173, 180, 181, 189, 199, 200, 228–231, 233, 238

DSP	Digital Signal Processor 39
DTW	Dynamic Time Warping 91, 92
DVFS	Dynamic Voltage and Frequency Scaling 33, 205
DVS	Dynamic Voltage Scaling 205
EA	Evolutionary Algorithm 9, 28–31, 34, 37, 59–61, 121, 123, 138, 161, 162, 167, 174, 230, 231, 237, 238
EACOF	Energy Aware COmputing Framework 31
EC	Evolutionary Computation 59
ECJ	A Java-based Evolutionary Computation research system 160, 162, 166
ELISA	Enzyme-Linked Immuno-Sorbent Assay 24
EMOA	Evolutionary Multi-objective Optimization Algorithm 29, 249
ES	Embedded System 5, 9, 25, 26, 31, 33–35, 39, 69, 91, 107, 151, 156–159, 193, 196, 200–203, 208, 225, 226, 228, 235–237, 239
FADSE	Framework for Automatic Design Space Exploration 31
FALIS	Feedback-based and memory-Aware gLobal Instruction Scheduling 8, 33
FFT	Fast Fourier Transform 77
FN	False Negatives 57–59, 124, 125
FP	False Positives 57–59, 66, 124, 125, 128, 226
FPGA	Field Programmable Gate Array 31, 33, 34, 39
GA	Genetic Algorithm 28–30, 33, 34, 59, 60, 63, 121, 160, 203
GDDR	Graphics Double Data Rate 136, 141–143, 150, 179, 180, 187, 188, 195
GOOM	GPGPU OpenCL Offloading Middleware 239, 241–243
GPGPU	General-Purpose computing on Graphics Processing Units 1, 5, 8, 9, 25, 26, 28, 32, 33, 37–39, 41, 45, 56, 60, 67, 77, 116, 117, 120, 121, 134, 135, 139, 140, 145, 146, 151, 152, 156, 157, 162, 174, 181, 234, 239–243, 246, 249

GPU	Graphics Processing Unit 1, 4–11, 25–27, 30–35, 37–39, 41, 42, 44–46, 49, 50, 52, 53, 56, 60, 67–69, 72, 75–77, 79, 81, 82, 91, 98, 100, 102, 106, 110, 112–114, 116–120, 122, 128, 129, 131–152, 156–160, 162, 163, 165–168, 172–183, 185–190, 193–202, 204, 207–209, 217, 222, 226, 229, 230, 233, 235–239, 241, 242, 274, 275
GUI	Graphical User Interface 103, 104, 210
HIL	Hardware-In-the-Loop 3, 63, 122
HIV	Human Immunodeficiency Virus 19–21, 24, 125
HMP	Heterogeneous Multi-Processing 201
HPC	High-Performance Computing 4, 8, 9, 25, 69, 146–148, 150–152, 156–158, 160, 172, 173, 177, 179, 180, 186, 188, 189, 200, 231, 234–236, 239, 274
IPP	Integrated Power and Performance 32
ISAS	Institute for Analytical Science 13
JGAP	Java Genetic Algorithms Package 121
KVM	Kernel-based Virtual Machine 144, 181
LAN	Local Area Network 242
LoG	Laplacian Of Gaussian 84
LTE	Long Term Evolution 3, 190–193, 199, 200, 230, 235, 242
LTS	Long Time Support 210
MAE	Mean Absolute Error 218–221, 223, 224
McPAT	MultiCore Power, Area, and Timing 137, 138, 147, 168, 238
MOBLIS	Multi-OBjective Local Instruction Scheduling 33
MOCHC	Multi-Objective Cross generational elitist selection, Heterogeneous recombination and Cataclysmic mutation 31
MOEA	Multi-Objective Evolutionary Algorithms 28, 29

MOGEA-DSE	Multi-Objective GPGPU Energy-Aware Design Space Exploration 7–11, 112, 116, 129, 156–158, 160–168, 171–175, 179, 181, 186, 189–192, 194, 195, 199, 200, 202, 203, 205–210, 222, 225–231, 233–239
MP-MSP430-Sim	Massive Parallel MSP430 Simulator 239, 240
MPSoC	MultiProcessor System-On-Chip 5, 31, 34, 156, 200–203, 226, 239
NOP	NO Operation 100
NSGA-II	Non-dominated Sorting Genetic Algorithm-II 28, 29, 31, 33, 34, 166, 167, 209, 230
NSGA-III	Non-dominated Sorting Genetic Algorithm-III 28
OCL	Object Constraint Language 34
OpenCL	Open Computing Language 26, 28, 39–43, 45, 49–53, 67, 69, 72, 76, 79, 88, 96, 99, 102, 111–113, 135, 138–140, 142–144, 148, 150, 176–179, 186, 188, 196, 202, 203, 206, 219, 220, 239, 241, 242, 246
OpenCV	Open Source Computer Vision 88
OpenGL	Open Graphics Library 103
OpenMP	Open Multi-Processing 144
OS	Operating System 148, 205, 210, 217
PAMONO	Plasmon Assisted Microscopy Of Nano-sized Objects 1–3, 7–11, 13–27, 37, 53–55, 57, 58, 61–63, 66–70, 72–74, 77–82, 84, 86, 87, 91–96, 98, 103, 104, 106, 109, 110, 117, 118, 122, 124, 125, 128–134, 146, 147, 153, 156–160, 172, 173, 177, 180, 189–191, 195, 200, 201, 208, 225, 226, 228, 229, 231, 233–240
PCR	Polymerase Chain Reaction 24, 25
PSE	Parameter Space Exploration 2–10, 13, 23, 28–30, 53, 67, 106, 116, 117, 122, 128, 129, 146, 151–153, 156–158, 160, 161, 189, 200, 228–231, 233, 238
PTX	Parallel Thread eXecution 135, 136, 139, 140, 148, 168, 230
QoR	Quality Of Result 28, 29, 60, 116, 122, 123, 158, 200, 206, 215, 225–228, 230, 234

RAE	Relative Absolute Error 218–221, 223, 224
REVAC	Relevance Estimation and VAlue Calibration on parameters 29
RFC-Gen	Random Forest Code GENeration 7, 8, 10, 27, 102, 107, 111–114, 233, 234
RISC	Reduced Instruction Set Computing 239
RNA	RiboNucleic Acid 24, 25
ROC	Receiver Operating Characteristic 106, 107
ROP	Raster Operation Processor 163, 165, 179, 180, 195, 274, 275
RTP	Real-time Transport Protocol 193
SASS	Source and ASSembly 135, 136
SDK	Software Development Kit 143, 144, 178, 179
SIL	Software-In-the-Loop 3
SIMD	Single Instruction Multiple Data 35, 38, 39, 41, 69, 135, 136
SLD	Super-Luminescent Diode 14–17, 70, 72–74
SMP	Streaming MultiProcessor 27, 32, 38, 40, 45, 46, 49, 50, 56, 113, 135–137, 141–145, 152, 163, 165, 166, 174, 175, 179–181, 184, 185, 187, 188, 195, 208, 274, 275
SMS-EMOA	S-Metric-Selection EMOA 29
SNR	Signal-to-Noise Ratio 7, 18, 27, 57, 66, 68, 73, 74, 82, 88–90, 103, 109, 110, 127, 128, 191, 234, 236, 238
SoC	System-On-a-Chip 33, 34
SOG-DSE	Single-Objective GPGPU Design Space Exploration 7–10, 116, 117, 119–122, 129, 134, 135, 138–140, 143, 145–148, 150–153, 157, 158, 160–163, 166, 171, 173, 229, 233–238
SOG-PSE	Single-Objective GPGPU Parameter Space Exploration 7, 8, 10, 116, 117, 119–125, 127–129, 134, 138, 146, 151–153, 157, 158, 160–162, 166, 229, 233–238
SP	Streaming Processor 37, 38, 42, 50, 131, 132, 134–137, 141–143, 146, 149–151, 163, 165, 168, 179–181, 185, 187, 188, 195, 274, 275

SPEA-2	Strength Pareto Evolutionary Algorithm 2 28, 29, 31, 33, 166, 167, 181, 230
SPMD	Single Program Multiple Data 39
SPO	Sequential Parameter Optimization 29
SPR	Surface Plasmon Resonance 7, 13, 15, 18, 24, 238
SPRi	Surface Plasmon Resonance Imaging 24
SURF	Speeded Up Robust Features 83
SVM	Support Vector Machine 29, 30, 223, 224, 227
SynOpSis	SYNthesis/OPtimization/analySIS 10, 20, 30, 37, 61–63, 107–109, 125, 208
TDP	Thermal Design Power 9, 146–148
TMU	Texture Mapping Unit 163, 165, 179, 180, 195, 274, 275
TN	True Negatives 57–59, 124
TP	True Positives 57–59, 124
UDP	User Datagram Protocol 193
VirusDetectionCL	VIRUS DETECTION with openCL 7–11, 30, 60–62, 66–69, 72, 78, 82, 98, 101–111, 113, 117, 119, 122–124, 127–132, 134, 138, 142, 152, 156–158, 168, 169, 172, 175–182, 185–187, 189, 191, 193, 195, 196, 199–202, 204–208, 212, 216, 217, 225, 227, 228, 230, 231, 233–238, 271, 275
VLP	Virus Like Particle 19–21, 107, 125
VRM	Voltage Regulator Module 137
W-LAN	Wireless Local Access Network 35, 239
WEKA	Waikato Environment for Knowledge Analysis 102, 107, 111, 217, 218, 223
XML	eXtensible Markup Language 161

List of Mathematical Symbols

A	Artifacts within the sensor images, cf. Definition 15.
B	A background signal within the sensor images, cf. Definition 16, which consists of a constant background signal combined with some artifact signal.
B_{constant}	The constant part of the signal within the sensor images.
$\widetilde{B}$	An approximation of the background signal, cf. Equation (5.6).
$*$	The convolution operation.
$\widetilde{C}$	An approximation of the current signal, cf. Equation (5.7).
$\widetilde{C}$	The estimated number of cycles.
E	The (actual) energy consumption.
$\widetilde{E}$	An approximation of the energy consumption.
e	Euler's number.
F_1	The F_1 score is used to quantify the detection quality, cf. Definition 37.
G	A representation of the global GPU memory, cf. Definition 4.
h	The height of the input images, cf. Notation 1.
h_{g}	The padded height of the input images, cf. Definition 2.
h_{l}	The height of the work group, cf. Notation 2.
h_{pl}	The padded height of a local memory, cf. Notation 4.
w_{tile}	The height of an image tile or local memory, cf. Definition 27.
I	A gray-scale image or a queue of gray-scale images.
i	A representation of a GPU work-item, cf. Definition 3.
K	The degradation kernel used to model the distortion of the particles, depending of the spot on the sensor where they appear.
S	Set of all work items within a two dimensional global index space, cf. Definition 2.
L	A representation of the local GPU memory, cf. Definition 6.
median	The median of a set, cf. Definition 26.

μ	The average of a distribution, cf. Definition 19.
$\hat{\mu}$	An approximation of the average of a distribution, cf. Definition 19.
$\mathbb{N}$	The natural numbers without zero.
N	A multiset of neighborhood coordinates, cf. Definition 18.
$\mathbb{N}_0$	The natural numbers included zero.
N	An additive noise term, cf. Definition 17.
$\mathcal{O}$	Big O of the Bachmann-Landau notation.
PA	The positive agreement, cf. Definition 38.
P	The (actual) Power consumption.
$\widetilde{P}$	An approximation of the power consumption.
$\mathbb{R}$	The real numbers.
σ	The standard deviation of a distribution, cf. Definition 20.
$\hat{\sigma}$	An approximation of the standard deviation of a distribution.
σ^2	The variance of a distribution (the squared standard deviation).
$\hat{\sigma}^2$	An approximation of the variance of a distribution (the squared standard deviation approximation).
T	A tile of an image. Is also used for the template image.
T	Time in the meaning of run time, execution time or communication time.
$\widetilde{T}$	An approximation of the (execution) time.
t	The frame number (time), with $t = 0$ referencing the first image in the image sequence.
V	The virus signal, caused by attaching viruses, cf. Definition 14. Also, used synonymously for signal of artificial particles and virus like particles.
$\widetilde{V}$	An approximation of the virus signal, cf. Equation (5.8).
$\hat{V}$	The actual virus signal from the sensor, cf. Equation (5.5).
w	The width of the input images, cf. Notation 1.
w_{g}	The padded width of the input images, cf. Definition 2.
w_{l}	The width of the work group, cf. Notation 2.
w_{pl}	The padded width of a local memory, cf. Notation 4.

w_{tile}	The width of an image tile or local memory, cf. Definition 27.
W	A representation of a GPU work group, cf. Definition 3.
$\mathbb{Z}$	The integer numbers.

Bibliography

[ACD+07] G. Ascia, V. Catania, A. G. Di Nuovo, M. Palesi, and D. Patti. "Efficient Design Space Exploration for Application Specific Systems-on-a-Chip". In: *Journal of Systems Architecture* 53.10 (2007), pp. 733–750. DOI: 10.1016/j.sysarc.2007.01.004 (Cited on page 35).

[ACL+06] F. Angiolini, J. Ceng, R. Leupers, F. Ferrari, C. Ferri, and L. Benini. "An Integrated Open Framework for Heterogeneous MPSoC Design Space Exploration". In: *Design, Automation and Test in Europe. Proceedings of the Conference on.* DATE '06. Leuven, Belgium: European Design and Automation Association, 2006, pp. 1145–1150 (Cited on page 35).

[AJB+13] Z. Abbasi, M. Jonas, A. Banerjee, S. K. S. Gupta, and G. Varsamopoulos. "Evolutionary Green Computing Solutions for Distributed Cyber Physical Systems". In: *Evolutionary Based Solutions for Green Computing.* Ed. by S. U. Khan, J. Kołodziej, J. Li, and A. Y. Zomaya. Vol. 432. Studies in Computational Intelligence. Berlin, Germany and Heidelberg, Germany: Springer, 2013, pp. 1–28. DOI: 10.1007/978-3-642-30659-4_1 (Cited on page 34).

[AK12] B. Akay and D. Karaboga. "A Modified Artificial Bee Colony Algorithm for Real-Parameter Optimization". In: *Information Sciences* 192 (2012), pp. 120–142. DOI: 10.1016/j.ins.2010.07.015 (Cited on page 30).

[All15a] Allied Vision Technologies. *Guppy PRO F-503.* 2015. URL: http://www.covistech.com/uploads/AVT_Datasheets_Product_Images_App_Notes/Guppy%20PRO_F-503.pdf (Cited on page 17).

[All15b] Allied Vision Technologies. *Prosilica GC 2450.* 2015. URL: https://www.alliedvision.com/en/products/cameras/detail/2450/action/pdf.html (Cited on page 17).

[APS04] G. Agosta, G. Palermo, and C. Silvano. "Multi-Objective Co-Exploration of Source Code Transformations and Design Space Architectures for Low-Power Embedded Systems". In: *Applied Computing. Proceedings of the 2004 ACM Symposium on.* 2004, pp. 891–896 (Cited on page 35).

[AR12] I. Ahmad and S. Ranka. *Handbook of Energy-Aware and Green Computing.* Chapman and Hall/CRC Computer and Information Science Series. Boca Raton, FL, USA: CRC Press, 2012 (Cited on page 30).

[Ash10] M. Ash. *OpenCL Basics.* 2010. URL: http://www.mikeash.com/pyblog/friday-qa-2010-04-02-opencl-basics.html (Cited on page 51).

[AVS+14] H. Abdeen, D. Varró, H. Sahraoui, A. S. Nagy, C. Debreceni, Á. Hegedüs, and Á. Horváth. "Multi-Objective Optimization in Rule-Based Design Space Exploration". In: *Automated Software Engineering (ASE). Proceedings of the 29th ACM/IEEE International Conference on.* ASE '14. New York, NY, USA: ACM, 2014, pp. 289–300. DOI: 10.1145/2642937.2643005 (Cited on page 34).

[AW81] B. D. Acland and N. H. Weste. "The Edge Flag Algorithm - A Fill Method for Raster Scan Displays". In: *Computers, IEEE Transactions on* C-30.1 (1981), pp. 41–48. DOI: 10.1109/TC.1981.6312155 (Cited on page 100).

[Ban14] I. N. Bankman, ed. *Handbook of Medical Image Processing and Analysis.* 2nd ed. Academic Press Series in Biomedical Engineering. Amsterdam, Netherlands: Elsevier / Academic Press, 2014 (Cited on page 25).

[BB12] J. Bergstra and Y. Bengio. “Random Search for Hyper-Parameter Optimization”. In: *The Journal of Machine Learning Research* 13.1 (2012), pp. 281–305 (Cited on page 29).

[BBB+11] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. “The Gem5 Simulator”. In: *SIGARCH Computer Architecture News* 39.2 (2011), pp. 1–7. DOI: 10.1145/2024716.2024718 (Cited on page 237).

[BBG+10] T. Basten, E. Benthum, M. Geilen, M. Hendriks, F. Houben, G. Igna, F. Reckers, S. Smet, L. Somers, E. Teeselink, N. Trčka, F. Vaandrager, J. Verriet, M. Voorhoeve, and Y. Yang. “Model-Driven Design-Space Exploration for Embedded Systems: The Octopus Toolset”. In: *Leveraging Applications (ISoLA). 4th International Symposium on.* Ed. by T. Margaria and B. Steffen. Berlin, Germany and Heidelberg, Germany: Springer, 2010, pp. 90–105. DOI: 10.1007/978-3-642-16558-0_10 (Cited on page 35).

[BC10] W. Baek and T. M. Chilimbi. “Green: A Framework for Supporting Energy-Conscious Programming Using Controlled A pproximation”. In: *SIGPLAN Notices* 45.6 (2010), pp. 198–209. DOI: 10.1145/1809028.1806620 (Cited on page 36).

[BC94] D. J. Berndt and J. Clifford. “Using Dynamic Time Warping to Find Patterns in Time Series”. In: *KDD workshop* 10 (1994) (Cited on pages 91 sq.).

[BF14] W. Birkfellner and M. Figl. *Applied Medical Image Processing: A Basic Course*. 2nd ed. Boca Raton, FL, USA: CRC Press, 2014 (Cited on page 25).

[BFK+14] M. Brettel, N. Friederichsen, M. Keller, and M. Rosenberg. “How Virtualization, Decentralization and Network Building Change the Manufacturing Landscape: An Industry 4.0 Perspective”. In: *International Journal of Science, Engineering and Technology 8 (1), 37* 44 (2014) (Cited on page 4).

[BL15] Z. Banković and P. López-García. “Stochastic vs. Deterministic Evolutionary Algorithm-Based Allocation and Scheduling for XMOS Chips”. In: *Neurocomputing* 150 (2015), pp. 82–89. DOI: 10.1016/j.neucom.2014.06.077 (Cited on page 35).

[BLB+08] J. B. Beusink, A. M. C. Lokate, Besselink, Geert A. J., G. J. M. Pruijn, and Schasfoort, Richard B. M. “Angle-Scanning SPR Imaging for Detection of Biomolecular Interactions on Microarrays”. In: *Biosensors and Bioelectronics* 23.6 (2008), pp. 839–844. DOI: 10.1016/j.bios.2007.08.025 (Cited on page 24).

[BLP10] T. Bartz-Beielstein, C. Lasarczyk, and M. Preuss. “The Sequential Parameter Optimization Toolbox”. In: *Experimental Methods for the Analysis of Optimization Algorithms.* Ed. by T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss. Berlin, Germany and Heidelberg, Germany: Springer, 2010, pp. 337–362. DOI: 10.1007/978-3-642-02538-9_14 (Cited on page 30).

[BNE07] N. Beume, B. Naujoks, and M. Emmerich. “SMS-EMOA: Multiobjective Selection Based on Dominated Hypervolume”. In: *European Journal of Operational Research* 181.3 (2007), pp. 1653–1669. DOI: 10.1016/j.ejor.2006.08.008 (Cited on page 29).

[Bou97] T. Boutell. *PNG (Portable Network Graphics) Specification Version 1.0.* RFC Editor, 1997. DOI: 10.17487/rfc2083 (Cited on page 193).

[Bra00] G. Bradski. “OpenCV Library”. In: *Dr. Dobb’s Journal of Software Tools* (2000) (Cited on page 88).

[Bre01] L. Breiman. “Random Forests”. In: *Machine Learning* 45.1 (2001), pp. 5–32 (Cited on pages 102, 111, 222).

[BRF15] L. Backes, A. Rico, and B. Franke. “Experiences in Speeding Up Computer Vision Applications on Mobile Computing Platforms”. In: *Systems, Architectures, Modeling, and Simulation (SAMOS XV). Proceedings of the International Symposium on.* 2015 (Cited on pages 26, 35, 202, 224).

[Bro15] D. Brodowski. *CPU Frequency and Voltage Scaling Code in the Linux Kernel: Linux CPUFreq - CPUFreq Governors.* 2015. URL: `https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt` (Cited on pages 203 sq.).

[BS03] J. M. S. Bartlett and D. Stirling. “A Short History of the Polymerase Chain Reaction”. In: *PCR Protocols.* Ed. by J. M. S. Bartlett and D. Stirling. Vol. 226. Methods in Molecular Biology. Totowa, NJ, USA: Humana Press, 2003, pp. 3–6. DOI: `10.1385/1-59259-384-4:3` (Cited on page 24).

[BSM+08] S. Bandyopadhyay, S. Saha, U. Maulik, and K. Deb. “A Simulated Annealing-Based Multiobjective Optimization Algorithm: AMOSA”. In: *Foundations and Trends in Computer Graphics and Vision* 12.3 (2008), pp. 269–283. DOI: `10.1109/TEVC.2007.900837` (Cited on page 30).

[BTG06] H. Bay, T. Tuytelaars, and L. van Gool. “SURF: Speeded Up Robust Features”. In: *Computer Vision (ECCV). International Conference on.* Ed. by A. Leonardis, H. Bischof, and A. Pinz. Vol. 3951. Lecture Notes in Computer Science. Berlin, Germany and Heidelberg, Germany: Springer, 2006, pp. 404–417. DOI: `10.1007/11744023_32` (Cited on page 83).

[Bub12] W. Bublitz. *Hoch parallele multiskalierte Tiefpassfilterung zur Rauschreduktion in Bilddatenfolgen: M.Sc. thesis.* Dortmund, Germany: Department of Computer Science, 2012 (Cited on page 77).

[BYF+09] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt. “Analyzing CUDA Workloads Using a Detailed GPU Simulator”. In: *Performance Analysis of Systems and Software (ISPASS). IEEE International Symposium on.* 2009, pp. 163–174. DOI: `10.1109/ISPASS.2009.4919648` (Cited on pages 32, 135 sq., 143, 166, 176).

[BZH+07] D. Boecker, A. Zybin, V. Horvatic, C. Grunwald, and K. Niemax. “Differential Surface Plasmon Resonance Imaging for High-Throughput Bioanalyses”. In: *Analytical Chemistry* 79.2 (2007), pp. 702–709. DOI: `10.1021/ac061623j` (Cited on pages 23, 74).

[Can15] Canon. *EF 50mm f/2.5 Compact Macro.* 2015. URL: `http://gdlp01.c-wss.com/gds/5/0300003495/02/ef50f25compactmacro-im2-eng.pdf` (Cited on page 18).

[CBM+09] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron. “Rodinia: A Benchmark Suite for Heterogeneous Computing”. In: *Workload Characterization (IISWC). IEEE International Symposium on.* 2009, pp. 44–54. DOI: `10.1109/IISWC.2009.5306797` (Cited on pages 143, 176).

[CDF+11] E. Cannella, L. Di Gregorio, L. Fiorin, M. Lindwer, P. Melonr, O. Neugebauer, and A. Pimentel. “Towards an ESL Design Framework for Adaptive and Fault-Tolerant MPSoCs: MADNESS or not?” In: *Embedded Systems for Real-Time Multimedia. 9th IEEE Symposium on.* 2011, pp. 120–129. DOI: `10.1109/ESTIMedia.2011.6088518` (Cited on page 35).

[CGG12] J. M. Cebrián, G. D. Guerrero, and J. M. García. “Energy Efficiency Analysis of GPUs”. In: *Parallel and Distributed Processing (IPDPSW). IEEE 26th International Symposium Workshops & PhD Forum on.* 2012, pp. 1014–1022. DOI: `10.1109/IPDPSW.2012.124` (Cited on page 33).

[Cha15] T. Chaira. *Medical Image Processing: Advanced Fuzzy Set Theoretic Techniques.* Boca Raton, FL, USA: CRC Press, 2015 (Cited on page 26).

[Che14] C. H. Chen, ed. *Computer Vision in Medical Imaging.* Vol. 2. Series in Computer Vision. New Jersey, USA: World Scientific, 2014. DOI: 10.1142/SCV (Cited on pages 25 sq.).

[CMF+04] T. M. Chinowsky, T. Mactutis, E. Fu, and P. Yager. "Optical and Electronic Design for a High-Performance Surface Plasmon Resonance Imager". In: *Optical Technologies for Industrial, Environmental, and Biological Sensing.* 2004, pp. 173–182 (Cited on page 24).

[Coe06] C. A. C. Coello. "Evolutionary Multi-Objective Optimization: A Historical View of the Field". In: *IEEE Computational Intelligence Magazine* 1.1 (2006), pp. 28–36. DOI: 10.1109/MCI.2006.1597059 (Cited on page 28).

[Coe99] C. A. C. Coello. "A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques". In: *Knowledge and Information Systems* 1.3 (1999), pp. 269–308. DOI: 10.1007/BF03325101 (Cited on page 28).

[CP07] A. Chinchuluun and P. M. Pardalos. "A Survey of Recent Developments in Multiobjective Optimization". In: *Annals of Operations Research* 154.1 (2007), pp. 29–50. DOI: 10.1007/s10479-007-0186-0 (Cited on page 28).

[CR14] A. Canedo and J. H. Richter. "Architectural Design Space Exploration of Cyber-Physical Systems Using the Functional Modeling Compiler". In: *Procedia CIRP* 21 (2014), pp. 46–51. DOI: 10.1016/j.procir.2014.03.183 (Cited on page 34).

[CS93] N. Chinchor and B. Sundheim. "MUC-5 Evaluation Metrics". In: *Message Understanding. Proceedings of the 5th Conference on.* MUC5 '93. Stroudsburg, PA, USA: Association for Computational Linguistics, 1993, pp. 69–78. DOI: 10.3115/1072017.1072026 (Cited on page 58).

[CSC+14] N. Chenouard, I. Smal, F. de Chaumont, M. Maška, I. F. Sbalzarini, Y. Gong, J. Cardinale, C. Carthel, S. Coraluppi, M. Winter, A. R. Cohen, W. J. Godinez, K. Rohr, Y. Kalaidzidis, L. Liang, J. Duncan, H. Shen, Y. Xu, K. E. G. Magnusson, J. Jaldén, H. M. Blau, P. Paul-Gilloteaux, P. Roudot, C. Kervrann, F. Waharte, J.-Y. Tinevez, S. L. Shorte, J. Willemse, K. Celler, G. P. van Wezel, H.-W. Dan, Y.-S. Tsai, C. O. de Solórzano, J.-C. Olivo-Marin, and E. Meijering. "Objective Comparison of Particle Tracking Methods". In: *Nature Methods* 11.3 (2014), pp. 281–289. DOI: 10.1038/nmeth.2808 (Cited on page 26).

[CSG05] B. C. Carter, G. T. Shubeita, and S. P. Gross. "Tracking Single Particles: A User-Friendly Quantitative Evaluation". In: *Physical Biology* 2.1 (2005), p. 60 (Cited on pages 66, 110).

[CTB+14] D. P. Chau, M. Thonnat, F. Brémond, and E. Corvée. "Online Parameter Tuning for Object Tracking Algorithms". In: *Image and Vision Computing* 32.4 (2014), pp. 287–302. DOI: 10.1016/j.imavis.2014.02.003 (Cited on page 30).

[CTL+12] J. Castrillón, A. Tretter, R. Leupers, and G. Ascheid. "Communication-Aware Mapping of KPN Applications onto Heterogeneous MPSoCs". In: *Design Automation Conference (DAC). Proceedings of the 49th Annual.* 2012, pp. 1266–1271 (Cited on page 35).

[CV10] H. Calborean and L. Vintan. "An Automatic Design Space Exploration Framework for Multicore Architecture Optimizations". In: *Networking in Education and Research (RoEduNet).* 2010, pp. 202–207 (Cited on page 30).

[CVV13] R. Chis, M. Vintan, and L. Vintan. "Multi-Objective DSE Algorithms' Evaluations on Processor Optimization". In: *Intelligent Computer Communication and Processing (ICCP). IEEE International Conference on.* 2013, pp. 27–33. DOI: 10.1109/ICCP.2013.6646076 (Cited on page 31).

[CWG01] M. K. Cheezum, W. F. Walker, and W. H. Guilford. "Quantitative Comparison of Algorithms for Tracking Single Fluorescent Particles". In: *Biophysical Journal* 81.4 (2001), pp. 2378–2388. DOI: 10.1016/S0006-3495(01)75884-5 (Cited on pages 7, 26, 57, 232).

[DAP+00] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. "A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II". In: *Lecture Notes in Computer Science* (2000), pp. 849–858. DOI: 10.1007/3-540-45356-3_83 (Cited on pages 28, 164 sq.).

[DBD+14] S. S. Dosanjh, R. F. Barrett, D. W. Doerfler, S. D. Hammond, K. S. Hemmert, M. A. Heroux, P. T. Lin, K. T. Pedretti, A. F. Rodrigues, T. G. Trucano, and J. P. Luitjens. "Exascale Design Space Exploration and Co-Design". In: *Future Generation Computer Systems* 30 (2014), pp. 46–58. DOI: 10.1016/j.future.2013.04.018 (Cited on page 31).

[Deb02] K. Deb. *Multi-Objective Optimization Using Evolutionary Algorithms.* Wiley-Interscience Series in Systems and Optimization. Chichester, UK: Wiley, 2002 (Cited on page 28).

[DG06] J. Davis and M. Goadrich. "The Relationship Between Precision-Recall and ROC Curves". In: *Machine Learning. Proceedings of the 23rd International Conference on.* ICML '06. New York, NY, USA: ACM, 2006, pp. 233–240. DOI: 10.1145/1143844.1143874 (Cited on page 58).

[DIC+13] B. Dusza, C. Ide, L. Cheng, and C. Wietfeld. "CoPoMo: A Context-Aware Power Consumption Model for LTE User Equipment". In: *Transactions on Emerging Telecommunications Technologies* 24.6 (2013), pp. 615–632. DOI: 10.1002/ett.2702 (Cited on pages 11, 188 sqq.).

[DJ14] K. Deb and H. Jain. "An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach: Part I: Solving Problems with Box Constraints". In: *IEEE Transactions on Evolutionary Computation* 18.4 (2014), pp. 577–601. DOI: 10.1109/TEVC.2013.2281535 (Cited on page 29).

[DK11] C. F. Dormann and I. Kühn. "Angewandte Statistik für die biologischen Wissenschaften". In: *Helmholtz Zentrum für Umweltforschung-UFZ* 2 (2011) (Cited on pages 55 sq.).

[DMM03] K. Deb, M. Mohan, and S. Mishra. "Towards a Quick Computation of Well-Spread Pareto-Optimal Solutions". In: *Evolutionary Multi-Criterion Optimization: Second International Conference, EMO 2003, Faro, Portugal, April 8–11, 2003. Proceedings.* Ed. by C. M. Fonseca, P. J. Fleming, E. Zitzler, L. Thiele, and K. Deb. Berlin, Heidelberg: Springer, 2003, pp. 222–236. DOI: 10.1007/3-540-36970-8_16 (Cited on page 29).

[DPA+02] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II". In: *Evolutionary Computation, IEEE Transactions on* 6.2 (2002), pp. 182–197. DOI: 10.1109/4235.996017 (Cited on page 164).

[DSJ06] Y. Dong, Z. Sun, and H. Jia. "A Cosine Similarity-Based Negative Selection Algorithm for Time Series Novelty Detection". In: *Mechanical Systems and Signal Processing* 20.6 (2006), pp. 1461–1472. DOI: 10.1016/j.ymssp.2004.12.006 (Cited on page 90).

[Dul54] R. Dulbecco. “Plaque Formation and Isolation of Pure Lines with Poliomylitis Viruses”. In: *Journal of Experimental Medicine* 99.2 (1954), pp. 167–182. DOI: 10.1084/jem.99.2.167 (Cited on page 24).

[DZJ03] K. Deb, P. Zope, and A. Jain. “Distributed Computing of Pareto-Optimal Solutions with Evolutionary Algorithms”. In: *Evolutionary Multi-Criterion Optimization*. Ed. by G. Goos, J. Hartmanis, J. van Leeuwen, C. M. Fonseca, P. J. Fleming, E. Zitzler, L. Thiele, and K. Deb. Vol. 2632. Lecture Notes in Computer Science. Berlin, Germany and Heidelberg, Germany: Springer, 2003, pp. 534–549. DOI: 10.1007/3-540-36970-8_38 (Cited on page 31).

[ECP06] C. Erbas, S. Cerav-Erbas, and A. D. Pimentel. “Multiobjective Optimization and Evolutionary Algorithms for the Application Mapping Problem in Multiprocessor System-on-Chip Design”. In: *Evolutionary Computation, IEEE Transactions on* 10.3 (2006), pp. 358–374. DOI: 10.1109/TEVC.2005.860766 (Cited on page 35).

[EDF+13] A. Eklund, P. Dufort, D. Forsberg, and S. M. LaConte. “Medical Image Processing on the GPU - Past, Present and Future”. In: *Medical Image Analysis* 17.8 (2013), pp. 1073–1094. DOI: 10.1016/j.media.2013.05.008 (Cited on page 25).

[ES15] A. Eiben and J. Smith. *Introduction to Evolutionary Computing*. 2nd ed. Natural Computing Series. Berlin, Germany and Heidelberg, Germany: Springer, 2015 (Cited on pages 29 sq.).

[FAE14] H. Field, G. Anderson, and K. Eder. *EACOF: A Framework for Providing Energy Transparency to Enable Energy-Aware Software Development*. ACM, 2014. DOI: 10.1145/2554850.2554920 (Cited on page 31).

[FLP+10] F. Ferrandi, P. L. Lanzi, C. Pilato, D. Sciuto, and A. Tumeo. “Ant Colony Heuristic for Mapping and Scheduling Tasks and Communications on Heterogeneous Embedded Systems”. In: *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 29.6 (2010), pp. 911–924. DOI: 10.1109/TCAD.2010.2048354 (Cited on page 31).

[FLY+16] Z. Fei, B. Li, S. Yang, C. Xing, H. Chen, and L. Hanzo. “A Survey of Multi-Objective Optimization in Wireless Sensor Networks: Metrics, Algorithms, and Open Problems”. In: *IEEE Communications Surveys Tutorials* 19.1 (2016), pp. 550–586. DOI: 10.1109/COMST.2016.2610578 (Cited on page 32).

[FM08] J. Fung and S. Mann. “Using Graphics Devices in Reverse: GPU-Based Image Processing and Computer Vision”. In: *Multimedia and Expo. IEEE International Conference on*. 2008, pp. 9–12. DOI: 10.1109/ICME.2008.4607358 (Cited on page 26).

[FP03] D. A. Forsyth and J. Ponce. *Computer Vision: A Modern Approach*. Prentice Hall Series in Artificial Intelligence. Upper Saddle River, NJ, USA: Prentice Hall, 2003 (Cited on page 26).

[GAR+15] R. Gensh, A. Aalsaud, A. Rafiev, A. Xia, A. Iliasov, A. Romanovsky, and A. Xakovlev. *Experiments with Odroid-XU3 Board*. Newcastle, UK, 2015 (Cited on page 35).

[GBH+99] K.-F. Giebel, C. S. Bechinger, S. Herminghaus, M. Riedel, P. Leiderer, U. M. Weiland, and M. Bastmeyer. *Imaging of Cell/Substrate Contacts of Living Cells with Surface Plasmon Resonance Microscopy*. Konstanz, Germany: Bibliothek der Universität Konstanz, 1999 (Cited on page 24).

[GLL+11] H. Grahn, N. Lavesson, M. H. Lapajne, and D. Slat. “CudaRF: A CUDA-Based Implementation of Random Forests”. In: *Computer Systems and Applications (AICCSA). 9th IEEE/ACS International Conference on*. 2011, pp. 95–101. DOI: 10.1109/AICCSA.2011.6126612 (Cited on page 27).

[GP13] S. D. Gan and K. R. Patel. “Enzyme Immunoassay and Enzyme-Linked Immunosorbent Assay”. In: *The Journal of Investigative Dermatology* 133.9 (2013), pp. 1–3. DOI: 10.1038/jid.2013.287 (Cited on pages 24 sq.).

[GSt16] GStreamer. 2016. URL: https://gstreamer.freedesktop.org (Cited on pages 68, 192).

[GTÜ+11] E. Gurevich, V. Temchura, K. Überla, and A. Zybin. “Analytical Features of Particle Counting Sensor Based on Plasmon Assisted Microscopy of Nano Objects”. In: *Sensors and Actuators B: Chemical* 160.1 (2011), pp. 1210–1215. DOI: 10.1016/j.snb.2011.09.050 (Cited on page 23).

[Gun99] S. R. Gunn. “On the Discrete Representation of the Laplacian of Gaussian”. In: *Pattern Recognition* 32.8 (1999), pp. 1463–1472. DOI: 10.1016/S0031-3203(98)00163-0 (Cited on page 83).

[Han11] J. Hanlon. “The XMOS XK-XMP-64 Development Board”. In: *Networks on Chip (NoCS). Fifth IEEE/ACM International Symposium on.* 2011, pp. 255–256 (Cited on page 239).

[Har15] Hardkernel. *Odroid-XU3.* 2015. URL: http://www.hardkernel.com/main/products/prdt_info.php?g_code=G140448267127 (Cited on page 199).

[HFH+09] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. “The WEKA Data Mining Software: An Update”. In: *ACM SIGKDD Explorations Newsletter* 11.1 (2009), pp. 10–18. DOI: 10.1145/1656274.1656278 (Cited on pages 102, 106, 111, 216 sq., 222).

[HK10] S. Hong and H. Kim. “An Integrated GPU Power and Performance Model”. In: *ACM SIGARCH Computer Architecture News.* Vol. 38. 2010, pp. 280–289 (Cited on page 32).

[HTF13] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* 2nd ed. Springer Series in Statistics. New York, NY, USA: Springer, 2013 (Cited on page 107).

[HTW+10] H. Hacker, C. Trinitis, J. Weidendorfer, and M. Brehm. “Considering GPGPU for HPC Centers: Is it Worth the Effort?” In: *Facing the Multicore-Challenge.* Ed. by R. Keller, D. Kramer, and J.-P. Weiss. Springer, 2010, pp. 118–130 (Cited on page 146).

[Int16] Intel. *Intel Core i7-6950X Processor Extreme Edition.* 2016. URL: http://ark.intel.com/products/94456/Intel-Core-i7-6950X-Processor-Extreme-Edition-25M-Cache-up-to-3_50-GHz (Cited on page 37).

[JBD13] A. Jooya, A. Baniasadi, and N. J. Dimopoulos. “Efficient Design Space Exploration of GPGPU Architectures”. In: *Euro-Par 2012: Parallel Processing Workshops.* Ed. by I. Caragiannis, M. Alexander, R. Badia, M. Cannataro, A. Costan, M. Danelutto, F. Desprez, B. Krammer, J. Sahuquillo, S. L. Scott, and J. Weidendorfer. Vol. 7640. Lecture Notes in Computer Science. Berlin, Germany and Heidelberg, Germany: Springer, 2013, pp. 518–527. DOI: 10.1007/978-3-642-36949-0_60 (Cited on page 32).

[JCV+12] R. Jahr, H. Calborean, L. Vintan, and T. Ungerer. “Boosting Design Space Explorations with Existing or Automatically Learned Knowledge”. In: *Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance: 16th International GI/ITG Conference (MMB & DFT).* Ed. by J. B. Schmitt. Berlin, Germany and Heidelberg, Germany: Springer, 2012, pp. 221–235. DOI: 10.1007/978-3-642-28540-0_16 (Cited on page 31).

[JRZ14] M. Janidarmian, K. Radecka, and Z. Zilic. “Automated Diagnosis of Knee Pathology Using Sensory Data”. In: *Wireless Mobile Communication and Healthcare (Mobihealth). EAI 4th International Conference on.* Ed. by IEEE. 2014, pp. 95–98. DOI: 10.1109/MOBIHEALTH.2014.7015918 (Cited on page 93).

[JSM12] W. Jia, K. A. Shaw, and M. Martonosi. “Stargazer: Automated Regression-Based GPU Design Space Exploration”. In: *Performance Analysis of Systems and Software (ISPASS). IEEE International Symposium on.* 2012, pp. 2–13 (Cited on page 32).

[JSW98] D. R. Jones, M. Schonlau, and W. J. Welch. “Efficient Global Optimization of Expensive Black-Box Functions”. In: *Journal of Global Optimization* 13.4 (1998), pp. 455–492. DOI: 10.1023/A:1008306431147 (Cited on page 30).

[JT81] J. F. Jarvis and J. A. Tyson. “FOCAS - Faint Object Classification and Analysis System”. In: *The Astronomical Journal* 86 (1981), pp. 476–495. DOI: 10.1086/112907 (Cited on pages 98 sq.).

[JZK+07] S. Jiang, X. Zhou, T. Kirchhausen, and S. T. C. Wong. “Detection of Molecular Particles in Live Cells via Machine Learning”. In: *Cytometry Part A* 71A.8 (2007), pp. 563–575. DOI: 10.1002/cyto.a.20404 (Cited on page 27).

[Kap15] Kappa. *DX 40 - 1020 FW.* 2015. URL: http://www.iberoptics.com/documentos/DX+40C+-+285+FW.pdf (Cited on page 17).

[KCS06] A. Konak, D. W. Coit, and A. E. Smith. “Multi-Objective Optimization Using Genetic Algorithms: A Tutorial”. In: *Reliability Engineering & System Safety* 91.9 (2006), pp. 992–1007. DOI: 10.1016/j.ress.2005.11.018 (Cited on page 28).

[Khr09] Khronos OpenCL Working Group. *The OpenCL Specification: Version 1.0.* 2009. URL: https://www.khronos.org/registry/cl/specs/opencl-1.0.pdf (Cited on page 39).

[Khr11a] Khronos OpenCL Working Group. *The OpenCL Specification: Version 1.1.* 2011. URL: https://www.khronos.org/registry/cl/specs/opencl-1.1.pdf (Cited on page 39).

[Khr11b] Khronos OpenCL Working Group. *The OpenCL Wrapper C++ API: Version 1.1.* 2011. URL: https://www.khronos.org/registry/cl/specs/opencl-cplusplus-1.1.pdf (Cited on page 239).

[Khr12] Khronos OpenCL Working Group. *The OpenCL Specification: Version 1.2.* 2012. URL: https://www.khronos.org/registry/cl/specs/opencl-1.2.pdf (Cited on pages 39, 41, 44, 51).

[Khr14] Khronos OpenCL Working Group. *The OpenCL Specification: Version 2.0.* 2014. URL: https://www.khronos.org/registry/cl/specs/opencl-2.0.pdf (Cited on page 39).

[KL10] K. Kumar and Y.-H. Lu. “Cloud Computing for Mobile Users: Can Offloading Computation Save Energy?” In: *Computer* 43.4 (2010), pp. 51–56. DOI: 10.1109/MC.2010.98 (Cited on pages 36, 194).

[Kle14] R. Klette. *Concise Computer Vision: An Introduction into Theory and Algorithms.* Undergraduate Topics in Computer Science. London, UK: Springer, 2014. DOI: 10.1007/978-1-4471-6320-6 (Cited on page 26).

[Koh11] M. Kohn. *naken430asm.* 2011. URL: http://www.mikekohn.net/micro/naken430asm_msp430_assembler.php (visited on 06/30/2015) (Cited on page 238).

[KP03] H. L. Kundel and M. Polansky. “Measurement of Observer Agreement”. In: *Radiology* 228.2 (2003), pp. 303–308. DOI: 10.1148/radiol.2282011860 (Cited on page 59).

[KPL+12] A. Klöckner, N. Pinto, Y. Lee, B. Catanzaro, P. Ivanov, and A. Fasih. “PyCUDA and PyOpenCL: A Scripting-Based Approach to GPU Run-Time Code Generation”. In: *Parallel Computing* 38.3 (2012), pp. 157–174. DOI: 10.1016/j.parco.2011.09.001 (Cited on page 27).

[Kro07] G. Kroah-Hartman. *Linux Kernel in a Nutshell*. O’Reilly, 2007 (Cited on pages 203 sq.).

[KSR+16] A. Kumar, F. Shaik, B. A. Rahim, and D. Kumar. *Signal and Image Processing in Medical Applications*. SpringerBriefs in Applied Sciences and Technology. Singapore: Springer, 2016. DOI: 10.1007/978-981-10-0690-6 (Cited on page 25).

[KT11] J. Keinert and J. Teich. *Design of Image Processing Embedded Systems Using Multidimensional Data Flow*. Embedded Systems. New York, NY, USA: Springer, 2011 (Cited on page 35).

[KTL+12] K. Kasichayanula, D. Terpstra, P. Luszczek, S. Tomov, S. Moore, and G. D. Peterson. “Power Aware Computing on GPUs”. In: *Application Accelerators in High Performance Computing (SAAHPC). Symposium on.* 2012, pp. 64–73. DOI: 10.1109/SAAHPC.2012.26 (Cited on page 32).

[Lan06] G. Landini. “Quantitative Analysis of the Epithelial Lining Architecture in Radicular Cysts and Odontogenic Keratocysts”. In: *Head & Face Medicine* 2.1 (2006), p. 4. DOI: 10.1186/1746-160X-2-4 (Cited on page 98).

[LC87] W. E. Lorensen and H. E. Cline. “Marching Cubes: A High Resolution 3D Surface Construction Algorithm”. In: *Computer Graphics and Interactive Techniques. Proceedings of the 14th Annual Conference on.* SIGGRAPH ’87. New York, NY, USA: ACM, 1987, pp. 163–169. DOI: 10.1145/37401.37422 (Cited on page 97).

[LDX+11] W. Liu, Z. Du, Y. Xiao, D. A. Bader, and C. Xu. “A Waterfall Model to Achieve Energy Efficient Tasks Mapping for Large Scale GPU Clusters”. In: *Parallel and Distributed Processing Workshops and Phd Forum. IEEE International Symposium on.* 2011, pp. 82–92. DOI: 10.1109/IPDPS.2011.129 (Cited on page 33).

[Lee07] E. A. Lee. *Computing Foundations and Practice for Cyber-Physical Systems: A Preliminary Report*. Ed. by EECS Department. Berkeley, CA, USA, 2007 (Cited on page 2).

[Len17] J. E. Lenssen. *Personal communication*. 2017 (Cited on pages 236 sq.).

[LHE+13] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi. “GPUWattch: Enabling energy optimizations in GPGPUs”. In: *International Symposium on Computer Architecture* 41.3 (2013), pp. 487–498. DOI: 10.1145/2508148.2485964 (Cited on pages 6 sq., 32, 134, 136 sqq.).

[Lib11] P. Libuschewski. *Massiv parallele Verarbeitung von Bildsequenzen zur Erkennung von Nanopartikeln: Diploma thesis*. Dortmund, Germany, 2011 (Cited on pages 10, 13 sq., 77, 89, 97 sq., 101, 103).

[LKD+14] P. Libuschewski, D. Kaulbars, B. Dusza, D. Siedhoff, F. Weichert, H. Müller, C. Wietfeld, and P. Marwedel. “Multi-Objective Computation Offloading for Mobile Biosensors via LTE”. In: *Wireless Mobile Communication and Healthcare (Mobihealth). EAI 4th International Conference on.* Ed. by IEEE. 2014, pp. 226–229. DOI: 10.4108/icst.mobihealth.2014.257374 (Cited on pages 11, 66, 155, 188 sq., 191, 194, 196 sq., 234).

[LL08] H. Li and D. Landa-Silva. “Evolutionary Multi-Objective Simulated Annealing with Adaptive and Competitive Search Direction”. In: *Evolutionary Computation (CEC). IEEE Congress on.* 2008, pp. 3311–3318. DOI: 10.1109/CEC.2008.4631246 (Cited on page 30).

[LLA+13] J. Lucas, S. Lal, M. Andersch, M. Alvarez-Mesa, and B. Juurlink. “How a Single Chip Causes Massive Power Bills: GPUSimPow: A GPGPU Power Simulator”. In: *Performance Analysis of Systems and Software (ISPASS). IEEE International Symposium on.* 2013, pp. 97–106. DOI: 10.1109/ISPASS.2013.6557150 (Cited on pages 32, 166).

[LMS+14] P. Libuschewski, P. Marwedel, D. Siedhoff, and H. Müller. “Multi-Objective, Energy-Aware GPGPU Design Space Exploration for Medical or Industrial Applications”. In: *Signal-Image Technology and Internet-Based Systems (SITIS). Tenth International Conference on.* Los Alamitos, CA, USA: IEEE Computer Society, 2014, pp. 637–644. DOI: 10.1109/SITIS.2014.11 (Cited on pages 11, 13 sq., 116, 135, 141, 143 sq., 155, 171, 177 sq., 181 sqq., 234, 276 sq.).

[LPB+14] S. Luke, L. Panait, G. Balan, S. Paus, Z. Skolicki, E. Popovici, K. Sullivan, J. Harrison, J. Bassett, and R. Hubley. *ECJ: A Java-Based Evolutionary Computation Research System.* 2014. URL: http://cs.gmu.edu/~eclab/projects/ecj/ (Cited on pages 158, 164).

[LST+13a] P. Libuschewski, D. Siedhoff, C. Timm, A. Gelenberg, and F. Weichert. “Fuzzy-Enhanced, Real-Time Capable Detection of Biological Viruses Using a Portable Biosensor”. In: *Biomedical Engineering Systems and Technologies (BIOSIGNALS). Proceedings of the International Joint Conference on.* 2013, pp. 169–174 (Cited on pages 10 sq., 20, 55 sq., 66, 94, 116, 122, 127, 129, 131, 133, 234).

[LST+13b] P. Libuschewski, D. Siedhoff, C. Timm, and F. Weichert. “Mobile Detektion viraler Pathogene durch echtzeitfähige GPGPU-Fuzzy-Segmentierung”. In: *Bildverarbeitung für die Medizin (BVM).* Ed. by H.-P. Meinzer, T. M. Deserno, H. Handels, and T. Tolxdorff. Informatik Aktuell. With permission of Springer. Berlin, Germany and Heidelberg, Germany: Springer, 2013, pp. 326–331. DOI: 10.1007/978-3-642-36480-8_57 (Cited on pages 11, 66).

[LSW14] P. Libuschewski, D. Siedhoff, and F. Weichert. “Energy-Aware Design Space Exploration for GPGPUs”. In: *Computer Science - Research and Development* 29.3-4 (2014), pp. 171–176. DOI: 10.1007/s00450-013-0237-5 (Cited on pages 11, 116, 135, 141, 146, 234).

[Luk13] S. Luke. *Essentials of Metaheuristics.* 2nd ed. Lulu, 2013 (Cited on pages 28, 59 sqq.).

[LW02] A. Liaw and M. Wiener. “Classification and Regression by RandomForest”. In: *R News* 2.3 (2002), pp. 18–22 (Cited on page 102).

[LWS10] J. Liu, J. M. White, and R. M. Summers. “Automated Detection of Blob Structures by Hessian Analysis and Object Scale”. In: *Image Processing (ICIP). 17th IEEE International Conference on.* 2010, pp. 841–844. DOI: 10.1109/ICIP.2010.5653499 (Cited on page 26).

[LWT12] P. Libuschewski, F. Weichert, and C. Timm. “Parameteroptimierte und GPGPU-basierte Detektion viraler Strukturen innerhalb Plasmonen-unterstützter Mikroskopiedaten”. In: *Bildverarbeitung für die Medizin (BVM).* Ed. by T. Tolxdorff, T. M. Deserno, H. Handels, and H.-P. Meinzer. Informatik Aktuell. With permission of Springer. Berlin, Germany and Heidelberg, Germany: Springer, 2012, pp. 237–242. DOI: 10.1007/978-3-642-28502-8_42 (Cited on pages 10 sq., 66, 116, 122, 126).

[LWX01] Z. Li, C. Wang, and R. Xu. “Computation Offloading to Save Energy on Handheld Devices: A Partition Scheme”. In: *Compilers, Architecture, and Synthesis for Embedded Systems. Proceedings of the 2001 International Conference on.* ACM, 2001, pp. 238–246. DOI: 10.1145/502217.502257 (Cited on pages 36, 194).

[MA04] R. T. Marler and J. S. Arora. “Survey of Multi-Objective Optimization Methods for Engineering”. In: *Structural and Multidisciplinary Optimization* 26.6 (2004), pp. 369–395. DOI: 10.1007/s00158-003-0368-6 (Cited on page 28).

[Mar11] P. Marwedel. *Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems*. 2nd ed. Embedded Systems. Dordrecht, Netherlands: Springer Science & Business Media, 2011 (Cited on pages 3, 5).

[Mat75] B. W. Matthews. “Comparison of the Predicted and Observed Secondary Structure of T4 Phage Lysozyme”. In: *Biochimica et Biophysica Acta (BBA) - Protein Structure* 405.2 (1975), pp. 442–451. DOI: 10.1016/0005-2795(75)90109-9 (Cited on page 59).

[May09] D. May. *The XMOS XS1 Architecture*. 2009. URL: https://www.xmos.com/en/download/public/The-XMOS-XS1-Architecture%281.0%29.pdf (Cited on page 239).

[MBH+14] A. Maghazeh, U. D. Bordoloi, A. Horga, P. Eles, and Z. Peng. “Saving Energy Without Defying Deadlines on Mobile GPU-Based Heterogeneous Systems”. In: *Hardware/Software Codesign and System Synthesis. Proceedings of the International Conference on.* CODES ’14. New York, NY, USA: ACM, 2014, 8:1–8:10. DOI: 10.1145/2656075.2656097 (Cited on page 33).

[MBP+10] G. Mariani, A. Brankovic, G. Palermo, J. Jovic, V. Zaccaria, and C. Silvano. “A Correlation-Based Design Space Exploration Methodology for Multi-Processor Systems-on-Chip”. In: *Design Automation Conference (DAC). Proceedings of the 47th.* DAC ’10. New York, NY, USA: ACM, 2010, pp. 120–125. DOI: 10.1145/1837274.1837307 (Cited on page 35).

[MDS12] E. Meijering, O. Dzyubachyk, and I. Smal. “Methods for Cell and Particle Tracking”. In: *Methods Enzymol* 504.9 (2012), pp. 183–200 (Cited on page 26).

[Meh05] J. Mehnen. *Mehrkriterielle Optimierverfahren für produktionstechnische Prozesse*. Schriftenreihe des ISF. Essen, Germany: Vulkan-Verlag, 2005 (Cited on page 28).

[Mei12] E. Meijering. “Cell Segmentation: 50 Years down the Road”. In: *Signal Processing Magazine, IEEE* 29.5 (2012), pp. 140–145. DOI: 10.1109/MSP.2012.2204190 (Cited on page 26).

[Mit16] S. Mittal. “A Survey of Techniques for Approximate Computing”. In: *ACM Computing Surveys* 48.4 (2016), 62:1–62:33. DOI: 10.1145/2893356 (Cited on pages 36, 204).

[MKK14] S. A. Mirsoleimani, A. Karami, and F. Khunjush. “A Two-Tier Design Space Exploration Algorithm to Construct a GPU Performance Predictor”. In: *Architecture of Computing Systems (ARCS).* Ed. by E. Maehle, K. Römer, W. Karl, and E. Tovar. Vol. 8350. Lecture Notes in Computer Science. Springer International Publishing, 2014, pp. 135–146. DOI: 10.1007/978-3-319-04891-8_12 (Cited on page 32).

[MNK10a] T. Mélange, M. Nachtegael, and E. E. Kerre. “A Fuzzy Filter for Random Impulse Noise Removal from Video”. In: *Fuzzy Information Processing Society (NAFIPS). Annual Meeting of the North American.* 2010, pp. 1–6. DOI: 10.1109/NAFIPS.2010.5548214 (Cited on page 79).

[MNK10b] T. Mélange, M. Nachtegael, and E. E. Kerre. “A Fuzzy Filter for the Removal of Random Impulse Noise in Colour Video”. In: *Fuzzy Systems (FUZZ-IEEE). The 19th IEEE International Conference on.* 2010, pp. 1–8. DOI: 10.1109/FUZZY.2010.5583983 (Cited on page 79).

[MNK11] T. Mélange, M. Nachtegael, and E. E. Kerre. “Fuzzy Random Impulse Noise Removal From Color Image Sequences”. In: *Image Processing. IEEE Transactions on* 20.4 (2011), pp. 959–970. DOI: 10.1109/TIP.2010.2077305 (Cited on pages 79, 94).

[MNS+11] T. Mélange, M. Nachtegael, S. Schulte, and E. E. Kerre. “A Fuzzy Filter for the Removal of Random Impulse Noise in Image Sequences”. In: *Image and Vision Computing* 29.6 (2011), pp. 407–419. DOI: 10.1016/j.imavis.2011.01.005 (Cited on pages 79 sqq.).

[Moo98] G. E. Moore. “Cramming More Components Onto Integrated Circuits”. In: *Proceedings of the IEEE* 86.1 (1998), pp. 82–85. DOI: 10.1109/JPROC.1998.658762 (Cited on pages 6, 134).

[Mor04] T. Morris. *Computer Vision and Image Processing*. Cornerstones of Computing. Basingstoke, UK: Palgrave Macmillan, 2004 (Cited on page 26).

[MPN+02] S. Mohanty, V. K. Prasanna, S. Neema, and J. Davis. “Rapid Design Space Exploration of Heterogeneous Embedded Systems Using Symbolic Search and Multi-Granular Simulation”. In: *Languages, Compilers and Tools for Embedded Systems: Software and Compilers for Embedded Systems. Proceedings of the Joint Conference on.* LCTES/SCOPES ’02. New York, NY, USA: ACM, 2002, pp. 18–27. DOI: 10.1145/513829.513835 (Cited on page 35).

[MPS+09] G. Mariani, G. Palermo, C. Silvano, and V. Zaccaria. “Meta-model Assisted Optimization for Design Space Exploration of Multi-Processor Systems-on-Chip”. In: *Digital System Design, Architectures, Methods and Tools (DSD). 12th Euromicro Conference on.* 2009, pp. 383–389. DOI: 10.1109/DSD.2009.154 (Cited on page 35).

[MR11] K. Meffert and N. Rotstan. *JGAP: Java Genetic Algorithms Package*. 2011. URL: http://jgap.%20sourceforge.%20net (Cited on page 121).

[MRE09] J. Mairhofer, K. Roppert, and P. Ertl. “Microfluidic Systems for Pathogen Sensing: A Review”. In: *Sensors* 9.6 (2009), pp. 4804–4823. DOI: 10.3390/s90604804 (Cited on pages 1, 232).

[MRV+15] R. G. Mantovani, A. L. D. Rossi, J. Vanschoren, B. Bischl, and A. C. P. L. F. de Carvalho. “Effectiveness of Random Search in SVM Hyper-parameter Tuning”. In: *Neural Networks (IJCNN). International Joint Conference on.* 2015, pp. 1–8. DOI: 10.1109/IJCNN.2015.7280664 (Cited on page 29).

[MSB+13] W. K. Moon, Y.-W. Shen, M. S. Bae, C.-S. Huang, J.-H. Chen, and R.-F. Chang. “Computer-Aided Tumor Detection Based on Multi-Scale Blob Detection Algorithm in Automated Breast Ultrasound Images”. In: *Medical Imaging. IEEE Transactions on* 32.7 (2013), pp. 1191–1200. DOI: 10.1109/TMI.2012.2230403 (Cited on pages 26, 82–86).

[MV14] S. Mittal and J. S. Vetter. “A Survey of Methods for Analyzing and Improving GPU Energy Efficiency”. In: *ACM Computing Surveys* 47.2 (2014), 19:1–19:23. DOI: 10.1145/2636342 (Cited on page 33).

[MVR+12] M. M. Molina, S. Ventura, C. Romero, and J. M. Luna. “Meta-Learning Approach for Automatic Parameter Tuning: A Case Study with Educational Datasets”. In: *Educational Data Mining. Proceedings of the 5th International Conference on (EDM).* 2012 (Cited on page 30).

[MWI+12] S. McIntosh-Smith, T. Wilson, A. Á. Ibarra, J. Crisp, and R. B. Sessions. “Benchmarking Energy Efficiency, Power Costs and Carbon Emissions on Heterogeneous Systems”. In: *The Computer Journal* 55.2 (2012), pp. 192–205 (Cited on pages 31 sq., 134).

[Nas08] D. B. Nash. “Health 3.0”. In: *Pharmacy and Therapeutics* 33.2 (2008), pp. 69–75 (Cited on page 4).

[NBG+08] J. Nickolls, I. Buck, M. Garland, and K. Skadron. “Scalable Parallel Programming with CUDA”. In: *Queue* 6.2 (2008), pp. 40–53. DOI: 10.1145/1365490.1365500 (Cited on page 70).

[Neu17] O. Neugebauer. *Personal communication.* 2017 (Cited on pages 11, 202, 236).

[NL15] O. Neugebauer and P. Libuschewski. *Odroid Energy Measurement Software.* 2015. URL: http://sfb876.tu-dortmund.de/auto?self=$egw1pio6bk (Cited on page 202).

[NLE+15] O. Neugebauer, P. Libuschewski, M. Engel, H. Müller, and P. Marwedel. “Plasmon-Based Virus Detection on Heterogeneous Embedded Systems”. In: *Software & Compilers for Embedded Systems (SCOPES). Proceedings of the Workshop on.* 2015. DOI: 10.1145/2764967.2764976 (Cited on pages 11, 66, 155, 199, 202 sq., 208, 210, 212, 214, 234).

[NMH+14] T. Nowatzki, J. Menon, C.-h. Ho, and K. Sankaralingam. *Gem5, GPGPUSim, McPAT, GPUWattch, "Your Favorite Simulator Here" Considered Harmful.* 2014 (Cited on page 235).

[NMK11] M. Nachtegael, T. Mélange, and E. E. Kerre. “Powerful Video Noise Removal Using Fuzzy Logic”. In: *GCC Conference and Exhibition (GCC). IEEE.* 2011, pp. 291–294. DOI: 10.1109/IEEEGCC.2011.5752536 (Cited on page 79).

[NP00] D. Nam and C. H. Park. “Multiobjective Simulated Annealing: A Comparative Study to Evolutionary Algorithms”. In: *International Journal of Fuzzy Systems* 2.2 (2000), pp. 87–97 (Cited on page 29).

[NSB+03] A. N. Naimushin, S. D. Soelberg, D. U. Bartholomew, J. L. Elkind, and C. E. Furlong. “A Portable Surface Plasmon Resonance (SPR) Sensor System with Temperature Regulation”. In: *Sensors and Actuators B: Chemical* 96.1-2 (2003), pp. 253–260. DOI: 10.1016/S0925-4005(03)00533-1 (Cited on pages 24, 236).

[NVI09a] NVIDIA Corporation. *Fermi Architecture Whitepaper: Version 1.1.* 2009. URL: http://www.nvidia.com/content/pdf/fermi_white_papers/nvidia_fermi_compute_architecture_whitepaper.pdf (Cited on page 141).

[NVI09b] NVIDIA Corporation. *NVIDIA OpenCL Best Practices Guide: Version 1.0.* 2009. URL: http://www.nvidia.com/content/cudazone/CUDABrowser/downloads/papers/NVIDIA_OpenCL_BestPracticesGuide.pdf (Cited on pages 38 sq., 45–48, 100, 112, 136).

[NVI12a] NVIDIA Corporation. *Kepler GK110 Whitepaper: Version 1.0.* 2012. URL: http://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf (Cited on page 38).

[NVI12b] NVIDIA Corporation. *Optimizing Parallel Reduction in CUDA.* 2012. URL: http://vuduc.org/teaching/cse6230-hpcta-fa12/slides/cse6230-fa12--05b-reduction-notes.pdf (Cited on pages 70, 72).

[NVI15a] NVIDIA Corporation. *CUDA Architecture.* 2015. URL: http://nvidia.com/object/cuda_home_new.html (Cited on page 176).

[NVI15b] NVIDIA Corporation. *CUDA C Best Practices Guide: Version 7.5.* 2015. URL: http://docs.nvidia.com/cuda/pdf/CUDA_C_Best_Practices_Guide.pdf (Cited on pages 38 sq.).

[NVI15c] NVIDIA Corporation. *Cuda Toolkit.* 2015. URL: https://developer.nvidia.com/cuda-toolkit (Cited on page 143).

[NVI15d] NVIDIA Corporation. *GeForce GTX Titan.* 2015. URL: http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-titan/specifications (Cited on pages 6, 134).

[NVI15e] NVIDIA Corporation. *GeForce GTX Titan Black*. 2015. URL: http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-titan-black/specifications (Cited on page 38).

[NVI16a] NVIDIA Corporation. *NVIDIA Pascal*. 2016. URL: https://www.nvidia.de/graphics-cards/geforce/pascal/ (Cited on pages 185 sq.).

[NVI16b] NVIDIA Corporation. *NVIDIA Tesla P100: GP100 Pascal Whitepaper*. 2016. URL: http://www.nvidia.de/object/pascal-architecture-whitepaper-de.html (Cited on page 186).

[NWV+01] M. Nachtegael, D. van der Weken, D. van de Ville, E. E. Kerre, W. Philips, and I. Lemahieu. "An Overview of Fuzzy Filters for Noise Reduction". In: *Fuzzy Systems (FUZZ-IEEE). The 10th IEEE International Conference on*. Vol. 1. 2001, pp. 7–10. DOI: 10.1109/FUZZ.2001.1007231 (Cited on page 79).

[Oli02] J.-C. Olivo-Marin. "Extraction of Spots in Biological Images Using Multiscale Products". In: *Pattern Recognition* 35.9 (2002), pp. 1989–1996. DOI: 10.1016/S0031-3203(01)00127-3 (Cited on page 27).

[OMC08] B. Ozisikyilmaz, G. Memik, and A. Choudhary. "Efficient System Design Space Exploration Using Machine Learning Techniques". In: *Design Automation Conference (DAC). Proceedings of the 45th Annual*. DAC '08. New York, NY, USA: ACM, 2008, pp. 966–969. DOI: 10.1145/1391469.1391712 (Cited on page 31).

[Pat05] P. Pattnaik. "Surface Plasmon Resonance: Applications in Understanding Receptor-Ligand Interaction". In: *Applied Biochemistry and Biotechnology* 126.2 (2005), pp. 79–92. DOI: 10.1385/ABAB:126:2:079 (Cited on pages 15, 23 sq.).

[PG02] M. Palesi and T. Givargis. "Multi-Objective Design Space Exploration Using Genetic Algorithms". In: *Hardware/Software Codesign (CODES). Proceedings of the Tenth International Symposium on*. 2002, pp. 67–72 (Cited on page 34).

[PKS+13] H. Park, Y. W. Ko, J. So, and J.-G. Lee. "Performance/Power Design Space Exploration and Analysis for GPU Based Software". In: *International Journal of Control and Automation* 6.6 (2013), pp. 371–380. DOI: 10.14257/ijca.2013.6.6.35 (Cited on page 33).

[Pop16] D. Popovic. *Saving Energy for Mobile Biosensors by Offloading: B.Sc. thesis*. Dortmund, Germany: Department of Computer Science, 2016 (Cited on pages 215, 235, 237).

[PSK+13] N. K. Pham, A. K. Singh, A. Kumar, and Aung, Khin Mi Mi. "Incorporating Energy and Throughput Awareness in Design Space Exploration and Run-Time Mapping for Heterogeneous MPSoCs". In: *Digital System Design (DSD). Euromicro Conference on*. 2013, pp. 513–521. DOI: 10.1109/DSD.2013.61 (Cited on page 35).

[QPh15] QPhotonics. *QSDM-680-9 Super-Luminescent Diode*. 2015. URL: http://www.qphotonics.com/get_attachment.php?id=882 (Cited on page 16).

[Qui92] R. J. Quinlan. "Learning with Continuous Classes". In: *Artificial Intelligence. 5th Australian Joint Conference on*. Singapore: World Scientific, 1992, pp. 343–348 (Cited on page 222).

[RCG+13] N. Rajovic, P. M. Carpenter, I. Gelado, N. Puzovic, A. Ramirez, and M. Valero. "Supercomputing with Commodity CPUs: Are Mobile SoCs Ready for HPC?" In: *High Performance Computing, Networking, Storage and Analysis (SC). International Conference for*. 2013, pp. 1–12. DOI: 10.1145/2503210.2503281 (Cited on page 237).

[RKB+16] J. Richter, H. Kotthaus, B. Bischl, P. Marwedel, J. Rahnenführer, and M. Lang. *Faster Model-Based Optimization through Resource-Aware Scheduling Strategies*. 2016 (Cited on page 30).

[RN88] J. L. Rodgers and W. A. Nicewander. “Thirteen Ways to Look at the Correlation Coefficient”. In: *The American Statistician* 42.1 (1988), pp. 59–66. DOI: 10.2307/2685263 (Cited on page 216).

[RPC84] R. Richard, R. Picard, and D. Cook. “Cross-Validation of Regression Models”. In: *Journal of the American Statistical Association* 79.387 (1984), pp. 575–583. DOI: 10.1080/01621459.1984.10478083 (Cited on pages 107, 216).

[RRP+99] A. Rudenko, P. Reiher, G. J. Popek, and G. H. Kuenning. “The Remote Processing Framework for Portable Computer Power Saving”. In: *Applied Computing. Proceedings of the 1999 ACM Symposium on.* SAC ’99. New York, NY, USA: ACM, 1999, pp. 365–372. DOI: 10.1145/298151.298385 (Cited on page 36).

[RSR+08] M. Rofouei, T. Stathopoulos, S. Ryffel, W. Kaiser, and M. Sarrafzadeh. “Energy-Aware High Performance Computing with Graphic Processing Units”. In: *Power Aware Computing and Systems. Proceedings of ACM SOSP Workshop on.* 2008 (Cited on pages 7, 32, 134, 171).

[Sam15] Samsung. *Specification of the Exynos 5422.* 2015. URL: http://www.samsung.com/global/business/semiconductor/product/application/detail?productId=7978 (Cited on page 200).

[San02] M. A. A. Sanvido. “Hardware-in-the-Loop Simulation Framework”. Ph.D. thesis. Swiss Federal Institute of Thechnology (ETH) Zurich, 2002. DOI: 10.3929/ethz-a-004317368 (Cited on page 3).

[SAS+09] L. Sheng, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. “McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures”. In: *Microarchitecture (MICRO-42). 42nd Annual IEEE/ACM International Symposium on.* 2009, pp. 469–480 (Cited on pages 137, 166, 237).

[SCR+13] S. Shuaiwen, S. Chunyi, B. Rountree, and K. W. Cameron. “A Simplified and Accurate Model of Power-Performance Efficiency on Emergent GPU Architectures”. In: *Parallel and Distributed Processing (IPDPS). IEEE 27th International Symposium on.* 2013, pp. 673–686. DOI: 10.1109/IPDPS.2013.73 (Cited on page 147).

[SEM+11] S. Scarano, M. L. Ermini, M. Mascini, and M. Minunni. “Surface Plasmon Resonance Imaging for Affinity-Based Sensing: An Analytical Approach”. In: *BioPhotonics. International Workshop on.* 2011, pp. 957–966. DOI: 10.1109/IWBP.2011.5954820 (Cited on pages 15, 24).

[SFL+14] D. Siedhoff, H. Fichtenberger, P. Libuschewski, F. Weichert, C. Sohler, and H. Müller. “Signal/Background Classification of Time Series for Biological Virus Detection”. In: *Pattern Recognition (GCPR). Proceedings of the 36th German Conference on.* Münster, Germany: Springer, 2014, pp. 2–5 (Cited on page 11).

[SFP+11] S. N. Sinha, J.-M. Frahm, M. Pollefeys, and Y. Genc. “Feature Tracking and Matching in Video Using Programmable Graphics Hardware: Machine Vision and Applications”. In: *Machine Vision and Applications* 22.1 (2011), pp. 207–217. DOI: 10.1007/s00138-007-0105-z (Cited on page 26).

[SG14] J. L. Semmlow and B. Griffel. *Biosignal and Medical Image Processing.* 3rd ed. Boca Raton, FL, USA: CRC Press, 2014 (Cited on page 25).

[Sha08] T. Sharp. “Implementing Decision Trees and Forests on a GPU”. In: *Computer Vision (ECCV). 10th European Conference on.* Ed. by D. Forsyth, P. Torr, and A. Zisserman. Berlin, Germany: Springer, 2008, pp. 595–608. DOI: 10.1007/978-3-540-88693-8_44 (Cited on page 27).

[SHB08] M. Sonka, V. Hlaváč, and R. Boyle. *Image Processing, Analysis, and Machine Vision.* 3rd ed. Toronto, Canada: Thomson Learning, 2008 (Cited on page 26).

[Sia09] P. Siarry. *Optimization in Signal and Image Processing.* Digital Signal and Image Processing Series. London, UK and Hoboken, NJ, USA: ISTE and Wiley, 2009 (Cited on page 29).

[Sie14] D. Siedhoff. *PAMONO Dataset Documentation.* 2014. URL: `http://sfb876.tu-dortmund.de/auto?self=$e45xe3ghkw` (Cited on page 21).

[Sie16] D. Siedhoff. "A Parameter-Optimizing Model-Based Approach to the Analysis of Low-SNR Image Sequences for Biological Virus Detection". Ph.D. thesis. Dortmund, Germany: TU Dortmund, 2016. DOI: `10.17877/DE290R-17272` (Cited on pages 10 sq., 20, 23, 30, 61, 66 sq., 102 sq., 106–110).

[Sin14] G. R. Sinha. *Medical Image Processing.* Prentice-Hall Of India, 2014 (Cited on page 25).

[SKB+00] S. K. Shevade, S. S. Keerthi, C. Bhattacharyya, and K. R. K. Murthy. "Improvements to the SMO Algorithm for SVM Regression". In: *IEEE Transactions on Neural Networks* 11.5 (2000), pp. 1188–1193. DOI: `10.1109/72.870050` (Cited on page 222).

[SLJ+14] M. Samadi, J. Lee, D. A. Jamshidi, S. Mahlke, and A. Hormati. "Scaling Performance via Self-Tuning Approximation for Graphics Engines". In: *ACM Transactions on Computer Systems* 32.3 (2014), 7:1–7:29. DOI: `10.1145/2631913` (Cited on page 28).

[SLN+09] I. Smal, M. Loog, W. Niessen, and E. Meijering. "Quantitative Comparison of Spot Detection Methods in Live-Cell Fluorescence Microscopy Imaging". In: *Biomedical Imaging: From Nano to Macro. IEEE International Symposium on.* 2009, pp. 1178–1181. DOI: `10.1109/ISBI.2009.5193268` (Cited on pages 7, 27, 66, 110, 232).

[SLW+14] D. Siedhoff, P. Libuschewski, F. Weichert, A. Zybin, P. Marwedel, and H. Müller. "Modellierung und Optimierung eines Biosensors zur Detektion viraler Strukturen". In: *Bildverarbeitung für die Medizin (BVM).* Ed. by T. M. Deserno, H. Handels, H.-P. Meinzer, and T. Tolxdorff. Informatik Aktuell. With permission of Springer. Berlin, Germany and Heidelberg, Germany: Springer, 2014, pp. 108–113. DOI: `10.1007/978-3-642-54111-7_24` (Cited on pages 10 sq., 20, 30, 55, 61 sqq., 66, 73 sq., 107 sq., 206).

[Smi06] K. I. Smith. "A Study of Simulated Annealing Techniques for Multi-Objective Optimisation". Ph.D. thesis. Exeter, UK: University of Exeter, 2006 (Cited on pages 29 sq.).

[SNM16] V. Scherbahn, S. Nizamov, and V. M. Mirsky. "Plasmonic Detection and Visualization of Directed Adsorption of Charged Single Nanoparticles to Patterned Surfaces". In: *Microchimica Acta* 183.11 (2016), pp. 2837–2845. DOI: `10.1007/s00604-016-1956-7` (Cited on page 23).

[SRW+14] Y. S. Shao, B. Reagen, G. Y. Wei, and B. D. "Aladdin: A Pre-RTL, Power-Performance Accelerator Simulator Enabling Large Design Space Exploration of Customized Architectures". In: *Computer Architecture (ISCA). ACM/IEEE 41st International Symposium on.* 2014, pp. 97–108. DOI: `10.1109/ISCA.2014.6853196` (Cited on page 35).

[SS01] G. Steiner and R. Salzer. "Biosensors Based on SPR Imaging". In: *2. BioSensor Symposium.* Ed. by Universität Tübingen. Tübingen, Germany: Universität Tübingen, 2001 (Cited on page 24).

[STM+15] V. Shpacovitch, V. Temchura, M. Matrosovich, J. Hamacher, J. Skolnik, P. Libuschewski, D. Siedhoff, F. Weichert, P. Marwedel, H. Müller, K. Überla, R. Hergenröder, and A. Zybin. "Application of Surface Plasmon Resonance Imaging Technique for the Detection of Single Spherical Biological Submicrometer Particles". In: *Analytical Biochemistry: Methods in the Biological Sciences* 486 (2015), pp. 62–69. DOI: 10.1016/j.ab.2015.06.022 (Cited on pages 11, 13, 23, 66, 89, 101, 235).

[Sut05] H. Sutter. "The Free Lunch is Over: A Fundamental Turn Toward Concurrency in Software". In: *Dr. Dobb's Journal* 30.3 (2005), pp. 202–210 (Cited on page 5).

[SWL+11] D. Siedhoff, F. Weichert, P. Libuschewski, and C. Timm. "Detection and Classification of Nano-Objects in Biosensor Data". In: *Microscopic Image Analysis with Applications in Biology* (2011) (Cited on pages 10 sq., 66, 234).

[SZS+14a] D. Siedhoff, A. Zybin, V. Shpacovitch, and P. Libuschewski. *PAMONO Sensor Data: 100nm 27Sep13 exp2*. 2014. DOI: 10.15467/e9ofalaebk (Cited on pages 20 sq.).

[SZS+14b] D. Siedhoff, A. Zybin, V. Shpacovitch, and P. Libuschewski. *PAMONO Sensor Data: 100nm 27Sep13 exp3*. 2014. DOI: 10.15467/e9ofomedxc (Cited on pages 20 sq.).

[SZS+14c] D. Siedhoff, A. Zybin, V. Shpacovitch, and P. Libuschewski. *PAMONO Sensor Data: 200nm 10Apr13*. 2014. DOI: 10.15467/e9ofqnvl6o (Cited on pages 20 sq.).

[SZS+14d] D. Siedhoff, A. Zybin, V. Shpacovitch, and P. Libuschewski. *PAMONO Sensor Data: 200nm 11Apr13 1*. 2014. DOI: 10.15467/e9ofylrdvk (Cited on pages 20 sq., 206).

[SZS+14e] D. Siedhoff, A. Zybin, V. Shpacovitch, and P. Libuschewski. *PAMONO Sensor Data: 200nm 11Apr13 2*. 2014. DOI: 10.15467/e9ofxjfh8g (Cited on pages 20 sq.).

[TAM+08] S. Thoziyoor, J. H. Ahn, M. Monchiero, J. B. Brockman, and N. P. Jouppi. "A Comprehensive Memory Modeling Tool and its Application to the Design and Analysis of Future Memory Hierarchies". In: *Computer Architecture (ISCA). 35th International Symposium on*. 2008, pp. 51–62. DOI: 10.1109/ISCA.2008.16 (Cited on page 137).

[TC13] A. Toma and J.-J. Chen. "Computation Offloading for Frame-Based Real-Time Tasks with Resource Reservation Servers". In: *Real-Time Systems (ECRTS). 25th Euromicro Conference on*. 2013, pp. 103–112. DOI: 10.1109/ECRTS.2013.21 (Cited on pages 36, 194).

[Tex13] Texas Instruments. *High- or Low-Side Measurement, Bidirectional CURRENT/POWER MONITOR with 1.8-V^{2}C Interface*. 2013 (Cited on pages 200, 202, 208).

[The08] A. J. P. Theuwissen. "CMOS Image Sensors: State-of-the-Art". In: *Solid-State Electronics* 52.9 (2008), pp. 1401–1406. DOI: 10.1016/j.sse.2008.04.012 (Cited on page 17).

[The15] The Imaging Source. *DMK 23UP031*. 2015. URL: http://dl.theimagingsource.com/4dd0f1c014/ (Cited on page 17).

[Tho15a] Thorlabs. *LDC205C Manual*. 2015. URL: https://www.thorlabs.com/thorcat/15900/LTC100-A-LDC205CManual.pdf (Cited on page 16).

[Tho15b] Thorlabs. *LTC100-B*. 2015. URL: https://www.thorlabs.com/newgrouppage9.cfm?objectgroup_id=2437&pn=LTC100-B (Cited on pages 16 sq.).

[Tho15c] Thorlabs. *TCLDM9 Manual*. 2015. URL: https://www.thorlabs.com/thorcat/1900/LTC100-A-TCLDM9Manual.pdf (Cited on page 16).

[Tho15d] Thorlabs. *TED200C Manual*. 2015. URL: https://www.thorlabs.com/thorcat/15900/LTC100-A-TED200CManual.pdf (Cited on page 17).

[Tho15e] Thorlabs. *WPH10M-670*. 2015. URL: www.thorlabs.de/newgrouppage9.cfm?objectgroup_id=711&pn=WPH10M-670 (Cited on page 17).

[Tim12] C. Timm. "Resource Efficient Processing and Communication in Sensor/Actuator Environments". Ph.D. thesis. Dortmund, Germany: TU Dortmund, 2012 (Cited on pages 10, 34, 60, 235).

[TJ15] J. M. R. S. Tavares and R. N. Jorge, eds. *Developments in Medical Image Processing and Computational Vision*. Vol. 19. Lecture Notes in Computational Vision and Biomechanics. Cham, Switzerland: Springer, 2015. DOI: 10.1007/978-3-319-13407-9 (Cited on page 25).

[TRS+02] D. Thomann, D. R. Rines, P. K. Sorger, and G. Danuser. "Automatic Fluorescent Tag Detection in 3D with Super-Resolution: Application to the Analysis of Chromosome Movement". In: *Journal of Microscopy* 208.1 (2002), pp. 49–64 (Cited on pages 86 sq.).

[WGK+08] R. Wolski, S. Gurun, C. Krintz, and D. Nurmi. "Using Bandwidth Data to Make Computation Offloading Decisions". In: *Parallel and Distributed Processing (IPDPS). IEEE International Symposium on*. 2008, pp. 1–8. DOI: 10.1109/IPDPS.2008.4536215 (Cited on page 36).

[WW97] Y. Wang and I. H. Witten. "Induction of Model Trees for Predicting Continuous Classes". In: *Machine Learning. Poster Papers of the 9th European Conference on*. Springer, 1997 (Cited on page 222).

[XLL07] C. Xian, Y.-H. Lu, and Z. Li. "Adaptive Computation Offloading for Energy Conservation on Battery-Powered Systems". In: *Parallel and Distributed Systems. International Conference on*. Vol. 2. 2007, pp. 1–8. DOI: 10.1109/ICPADS.2007.4447724 (Cited on pages 36, 194).

[XMO15] XMOS. *XK-XMP-64 Development Board*. 2015. URL: https://www.xcore.com/wiki/index.php/XK-XMP-64_Development_Board (Cited on page 239).

[YTW+10] W. Yi, Y. Tang, G. Wang, and X. Fang. "A Case Study of SWIM: Optimization of memory intensive application on GPGPU". In: *Parallel Architectures, Algorithms and Programming. 3rd International Symposium on*. 2010, pp. 123–129. DOI: 10.1109/PAAP.2010.22 (Cited on page 34).

[ZBM+07] A. Zybin, D. Boecker, V. Mirsky, and K. Niemax. "Enhancement of the Detection Power of Surface Plasmon Resonance Measurements by Optimization of the Reflection Angle". In: *Analytical Chemistry* 79.11 (2007), pp. 4233–4236 (Cited on pages 13, 15, 23).

[ZKG+09] A. Zybin, Y. Kuritsyn, E. Gurevich, V. Temchura, K. Überla, and K. Niemax. "Real-Time Detection of Single Immobilized Nanoparticles by Surface Plasmon Resonance Imaging". In: *Plasmonics* 5.1 (2009), pp. 31–35. DOI: 10.1007/s11468-009-9111-5 (Cited on pages 20, 23, 125).

[ZLT01] E. Zitzler, M. Laumanns, and L. Thiele. *SPEA2: Improving the Strength Pareto Evolutionary Algorithm*. Vol. 103. TIK-Report. Zürich, Switzerland: Eidgenössische Technische Hochschule Zürich (ETH), Institut für Technische Informatik und Kommunikationsnetze (TIK), 2001 (Cited on pages 28, 164 sq., 179).

[ZSS+16] A. Zybin, V. Shpacovitch, J. Skolnik, and R. Hergenröder. "Optimal Conditions for SPR-Imaging of Nano-Objects". In: *Sensors and Actuators B: Chemical* 239 (2016), pp. 338–342. DOI: 10.1016/j.snb.2016.07.124 (Cited on pages 13, 15, 23, 73).

[Zyb10] A. Zybin. "Verfahren zur hochaufgelösten Erfassung von Nanopartikeln auf zweidimensionalen Messflächen". DE Patent 10,2009,003,548 A1. 2010 (Cited on pages 13, 23).

[Zyb13] A. Zybin. “Method for High-Resolution Detection of Nanoparticles on Two-Dimensional Detector Surfaces”. US Patent 8,587,786 B2. 2013 (Cited on pages 11, 13, 15 sq., 18, 23, 89).

Appendix A

Appendix

Table A.1: The parameters of VirusDetectionCL.

Name	Range	Description
sensorModelOption	$\mathbb{N}$	Option to select a multiplicative or additive sensor model.
currentImage	$\{0,1\}$	Flag to calculate the current image for the sensor model.
currentImageRefs	$\mathbb{N}$	Number of images to use for the signal approximation.
currentImageGroupNFrames	$\mathbb{N}$	To subsequently lower the frame rate of the camera, sensor images can be grouped.
backgroundImage	$\{0,1\}$	Flag to calculate the background image for the sensor model.
backgroundGap	$\mathbb{N}$	Number of images between current image and background image.
backgroundImageRefs	$\mathbb{N}$	Number of images to use for the background approximation.
brightnessCorrection	$\{0,1\}$	Flag to enable the brightness correction.
brightnessCorrectionReference	$\mathbb{R}$	Manually set reference brightness.
detectOutshinedSpots	$\{0,1\}$	Flag to enable the detection of outshined spots.
temporalNoiseReductionOption	$\mathbb{N}$	Option to select the temporal noise reduction method.
averageImage	$\{0,1\}$	Flag to use an average filter for noise reduction.
averageImageKernelWidth	$\mathbb{N}$	Kernel width for the average filter.
averageImageKernelHeight	$\mathbb{N}$	Kernel height for the average filter.
medianImage	$\{0,1\}$	Flag to use a median filter for noise reduction.
medianImageKernelWidth	$\mathbb{N}$	Kernel width for the median filter.
medianImageKernelHeight	$\mathbb{N}$	Kernel height for the median filter.
gaussImage	$\{0,1\}$	Flag to use the Gauss filter for noise reduction.

gaussImageSigma	$\mathbb{R}^+$	The σ for the Gauss filter.
fuzzyNoiseRemoval	$\{0,1\}$	Flag to use the fuzzy filter for noise reduction.
fuzzyNoiseRemovalX1	$\mathbb{R}^+$	First value to define the fuzzy set for noise.
fuzzyNoiseRemovalX2	$\mathbb{R}^+$	Second value to define the fuzzy set for noise.
open/closeDetections	$\{0,1\}$	Flag to use the opening/closing of the detections.
open/closeDetectionsCircleRadius	$\mathbb{R}^+$	Radius of the circle used for opening/closing
combineDetections	$\{0,1\}$	Flag to combine detections over time.
timeSeriesOption	$\mathbb{N}$	Option to select the method for the time series features.
timeSeriesDistancePatterns	$\{0,1\}$	Flag to use pattern matching over time.
timeSeriesDistancePatternSize	$\mathbb{N}$	The size of the temporal pattern.
timeSeriesDistancePatternCount	$\mathbb{N}$	The number of temporal patterns if more than one pattern should be matched.
timeSeriesDistancePatternA	$\mathbb{N}_0$	Parameter for a manual slope relaxation.
timeSeriesDistancePatternB	$\mathbb{N}_0$	Parameter for a manual slope relaxation.
timeSeriesDistancePatternSlopeRelaxation	$\{0,1\}$	Flag to automatically choose *timeSeriesDistancePatternA* and *timeSeriesDistancePatternB*.
timeSeriesAutoUpdateVariablePatterns	$\mathbb{N}_0$	Automatic slope relaxation to adapt ideal patterns with the signal model.
timeSeriesDistanceMin/MaxStep	$[0,1]$	Range for the step sizes to detect different virus sizes.
timeSeriesDistanceMin/MaxNegativeStep	$[-1,0]$	Range for the negative step sizes.
fuzzyDetectionEnhancementX1	$[0,1]$	Parameter for the fuzzy detection enhancement.
templateMatchingFeature	$\{0,1\}$	Flag to calculate the template matching features.
gaussHesseFeature	$\{0,1\}$	Flag to calculate Gauss-Hesse features.
gaussHesseFeatureSigma	$\mathbb{R}^+$	The σ for the Gauss-Hesse feature.
gaussHesseFeatureNumScales	$\mathbb{N}$	Number of scales for the Gauss-Hesse feature.

gaussHesseFeatureDetectionThreshold	$\mathbb{R}^+$	Threshold to use the Gauss-Hesse feature for segmentation.
gaussHesseBlobFeatureDetectionThreshold	$\mathbb{R}^+$	Threshold to use the Gauss-Hesse blob feature for segmentation.
hesseFeature	$\{0,1\}$	Flag to calculate Hesse features.
templateMatchingFeature	$\{0,1\}$	Flag to calculate template matching features.
templateMatchingFeatureNormalize	$\{0,1\}$	Flag to normalize the correlation value for the template matching feature.
fuzzyDetectionEnhancement	$\{0,1\}$	Flag to use the fuzzy detection enhancement method.
fuzzyDetectionEnhancementX1	$\mathbb{R}^+$	Value from which on the fuzzy set large positive starts being above zero.
segmentationThreshold	$[0,1]$	Influences which pixels are used for segmentation.
segmentationThresholdDown	$[0,1]$	Same as *segmentationThreshold* but includes negative detection values.
mergingMaxDistance	$\mathbb{R}^+$	The maximum distance to combine two virus detections to one virus.
mergingMaxFrameDistanceForPolygon	$\mathbb{N}_0$	The maximum number of frames between two detections to combine them to one.
classifyWithThresholds	$\{0,1\}$	Flag to use threshold-based classification.
gaussHesseFeaturePolygonThreshold	$\mathbb{R}^+$	Threshold to use the Gauss Hesse feature for polygon classification.
gaussHesseBlobFeaturePolygonThreshold	$\mathbb{R}^+$	Threshold to use the Gauss Hesse blob feature for polygon classification.
templateMatchingFeatureThreshold	$\mathbb{R}^+$	Threshold to use the template matching feature for segmentation.
classifyWithRandomForest	$\{0,1\}$	Flag to use Random Forest classification.
visualizationOption	$\mathbb{N}$	Option to select the desired visualization.

Table A.2: Extended version of Table 8.2 for the evaluated GPU architectures. For each architecture the number of SMPs, number of SPs, core clock rate, DRAM type, DRAM clock rate, number of TMUs, ROPs, and shader clock rate are listed. Adapted from [LMS+14].

	GPU architecture	SMPs	SPs	Core clock	DRAM	DRAM clock	TMUs	ROPs	Shader clock
Desktop	FERMI™ GF-108	1-2	48,96	700-810 MHz	GDDR-3	400-450 MHz	4-8	3-6	1400-1620 MHz
	FERMI™ GF-106	3-4	144,192	590-790 MHz	GDDR-5	450-1000 MHz	12-16	9-12	1180-1580 MHz
	FERMI™ GF-104	6-7	288,336	650-675 MHz	GDDR-5	850-950 MHz	24-28	18-21	1300-1350 MHz
	FERMI™ GF-100	11-15	352,384,416,448,480	610-780 MHz	GDDR-5	800-1000 MHz	44-60	33-45	1220-1560 MHz
Mobile	FERMI™ GF-108M	1-2	48,96	600-740 MHz	GDDR-3	800-900 MHz	8-16	4-8	1200-1480 MHz
	FERMI™ GF-116M	3-4	144,192	590-775 MHz	GDDR-5	900-1250 MHz	24-32	12-16	1180-1550 MHz
	FERMI™ GF-114M	7-8	336,384	575-620 MHz	GDDR-5	1500 MHz	56-64	21-32	1150-1240 MHz
HPC	PASCAL™ GP-107[1]	4-5	1024,1280	1200 MHz	GDDR-5	1250-1450 MHz	64-80	16-20	2400 MHz
	PASCAL™ GP-104[1]	6-8	2304,2668,3072	820-1040 MHz	GDDR-6	1500-1750 MHz	192-256	48-64	1640-2080 MHz
	PASCAL™ GP-110a[1]	12-15	4608,4992,5376,5760	860-880 MHz	GDDR-6	1500-1750 MHz	384-480	96-120	1720-1760 MHz
	PASCAL™ GP-110b[1]	14-15	5376,5760	840-890 MHz	GDDR-6	1500-1750 MHz	448-480	112-120	1680-1780 MHz

[1] Devised architectures that did not exist during performance of the experiments.

Table A.3: Extended version of Table 8.3. The best GPU configurations for every program. Values for DRAM clock, TMUs, ROPs, and shader clock are left out for brevity but can be derived from Table A.2. Extended version of [LMS+14]

Program	GPU architecture(s)	SMPs	SPs	Core clock in MHz	DRAM
VirusDetectionCL (1)	GF-110	11,14,15	352,448,480	780,780,780	GDDR-5
VirusDetectionCL (2)	GF-114M	4,7	192,336	760,590	GDDR-5
VirusDetectionCL (3)	GP-107[1],104[1],110b[1]	5,6,12,15	1280,2304,4608,5760	1020,1040,870,890	GDDR-5,6
VirusDetectionCL (4)	GP-107[1],104[1],110b[1]	4,8,13,14	1024,3072,4992,5376	1020,870,870,890	GDDR-5,6
blackScholes	GF-100	11	352	630	GDDR-5
histogram	GF-100	13,14,15	416,448,480	780,780,780	GDDR-5
quasiRandomGen	GF-100	13	416	780	GDDR-5
reduction	GF-100	11,14,15	352,448,480	780,780,770	GDDR-5
matrixMul	GF-100	15	480	780	GDDR-5
mersenneTwister	GF-100	11,13	352,416	780,780	GDDR-5
vectorAdd	GF-100	11	352	780	GDDR-5
AES	GF-116,114M	4,7,8	192,336,384	770,615,615	GDDR-5
BFS	GF-108M	1	48	740	GDDR-4
CP	GF-116,114M	4,7,8	192,336,384	770,615,615	GDDR-5
LPS	GF-116,114M	4,8	192,384	770,615	GDDR-5
NN	GF-116,114M	4,8	192,384	770,615	GDDR-5
NQU	GF-108,116M	1,2,4	48,96,192	740,740,740	GDDR-4,5
RAY	GF-116,114M	4,7,8	192,336,384	760,615,615	GDDR-5
STO	GF-116M	4	192	770	GDDR-5
backProp	GP-107[1]	4,5	1024,1280	1020,1020	GDDR-5
	GP-104[1]	7,8	2688,3072	1040,1040	GDDR-6
gaussian	GP-107[1]	5	1280	1020	GDDR-5
hotSpot	GP-107[1],104[1]	4,6,8	1024,2304,3072	1020,1030,1020	GDDR-5,6
nn	GP-107[1],104[1],110b[1]	4,6,14	1024,2304,5376	1020,1040,890	GDDR-5,6
nw	GP-107[1]	4,5	1024,1280	1020,1020	GDDR-5
	GP-104[1],110b[1]	8,15	3072,5760	1020,890	GDDR-6

[1] Devised architectures that did not exist during performance of the experiments.