Resource-Efficient Processing and Communication in Sensor/Actuator Environments

Dissertation
zur Erlangung des Grades eines

Doktors der Ingenieurwissenschaften

der Technischen Universität Dortmund
an der Fakultät für Informatik
von

Constantin Timm

Dortmund
2012

Tag der mündlichen Prüfung:	18. Oktober 2012
Dekan / Dekanin:	Prof. Dr. Gabriele Kern-Isberner
Gutachter / Gutachterinnen:	Prof. Dr. Peter Marwedel
	Prof. Dr. Heinrich Müller

Dedication

To my wife and my child.

Acknowledgements

Die folgende Danksagung richtet sich an alle, die mich auf dem Weg meiner Promotion begleitet haben.

Zuallererst möchte ich mich herzlich bei meinem Betreuer und Erstgutachter, Prof. Dr. Peter Marwedel bedanken. Insbesondere möchte ich dabei hervorheben, dass mir – neben fachlicher Unterstützung – auch die Möglichkeit gegeben worden ist, in interessanten Projekten mit internationalen Schwerpunkten zu arbeiten. Desweiteren bedanke ich mich bei meinem Zweitgutachter und Betreuer, Prof. Dr. Heinrich Müller.

Mein Dank gilt den Korrekturlesern dieser Dissertationsschrift oder Kollegen, Dr. Sascha Plazar, Dr. Frank Weichert, Pascal Libuschweski, Olaf Neugebauer, Jens Schmutzler, Dominic Siedhoff und Anneliese Bähr-Böhm. Insbesondere Sascha und Frank haben mich nicht nur fachlich – in endlosen Diskussionen – unterstützt, sondern mich auch durch wertvolle Ratschläge im Rahmen dieser Arbeit begleitet. Für letzteres bedanke ich mich auch herzlich bei Prof. Dr. Heiko Falk, der dadurch einen signifikanten Beitrag zum Gelingen dieser Arbeit geleistet hat.

Desweiteren habe ich im Zuge der Promotion verschiedenste Leute kennenlernen dürfen, die alle – in der einen oder anderen Weise – zum Gelingen dieses Vorhabens beigetragen haben, sei es als Korrekturleser oder Koautor von Veröffentlichungen, als fachlicher oder mentaler Unterstützer, als Diplomand oder als Projektpartner. Meinen Dank möchte ich deshalb den folgenden Personen aussprechen: Daniel Cordes, Andrej Gelenberg, Markus Görlich, Dr. Michael Engel, David Fiedler, Andreas Heinig, Timon Kelter, Jan Kleinsorge, Helena Kotthaus, Dr. Paul Lokuciejewski, Dr. Stefan Michaelis, Jens Nellesen, Christian Prasse und Andreas Wolff.

Einen besonderen Dank widme ich meiner Familie, die mich während der Promotion unterstützt und mir die Kraft zum Vollenden dieser Arbeit gegeben hat.

Zuletzt möchte ich mich bei der Deutschen Forschungsgemeinschaft (DFG) für die finanzielle Unterstützung im Sonderforschungsbereich SFB 876 “Verfügbarkeit von Information durch Analyse unter Ressourcenbeschränkung”, Projekt B2, bedanken.

Contents

Chapter 1

Introduction

1.1 Motivation

The future of computer systems will not be dominated by personal computer like hardware platforms but by systems assisting humans in a hidden but omnipresent manner [152]. These computer systems can, for example, be utilized in the home automation sector to create sensor/actuator networks supporting the inhabitants of a house in everyday life. These kinds of systems are often summarized in literature by the terms ubiquitous computing [152] and pervasive computing [66]. According to [84], two emerging research areas exist, in which basic technologies have to be developed on the way to pervasive computing devices (illustrated in Figure 1.1):

- Embedded and Cyber-Physical Systems
- Communication Technologies.

Embedded and cyber-physical systems comprise a large variety of different hardware platforms in a vast variety of application domains. They can be found in the automation domain, in robots, in control systems for cars or air planes and in the multimedia equipment at home. The common feature of embedded systems is that they are "*information processing systems embedded in an enclosing product*" [84]. A subset of embedded systems comprises cyber-physical systems. They are "*integrations of computation and physical processes*" [76].

The other emerging area for pervasive computing devices are communication technologies, which spawn different challenges in the scope of distributed applications and networking. Especially, communication and middleware libraries must be designed efficiently in terms of quality of service but also with respect to resources like energy, memory space or processing resources. Processing resources and energy consumption are especially important for pervasive computing devices which are often battery-driven mobile devices.

The efficient usage of resources is an important topic at design time and at operation time of embedded and cyber-physical systems. The same applies to communication technologies. Therefore, this thesis presents methods which allow an efficient use of energy and processing resources in SANETs (Sensor/Actuator NETworks). SANETs comprise different sensor/actuator nodes cooperating for a "smart" joint

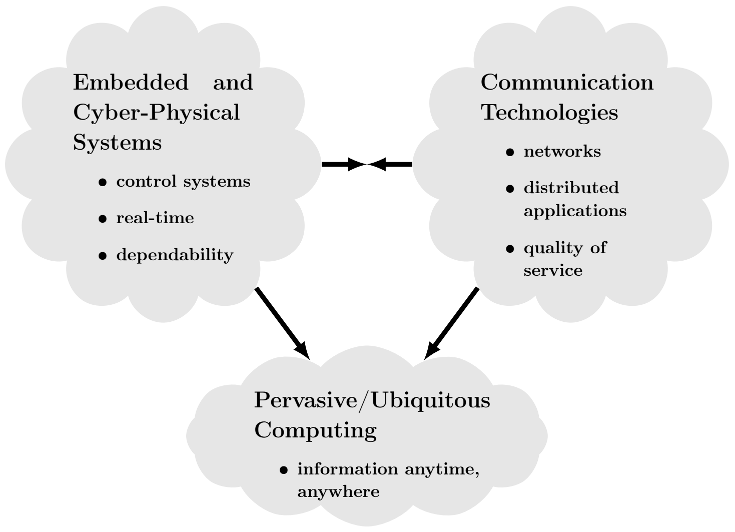

Embedded and Cyber-Physical Systems
• control systems
• real-time
• dependability
Communication Technologies
• networks
• distributed applications
• quality of service
Pervasive/Ubiquitous Computing
• information anytime, anywhere

Figure 1.1: Embedded/Cyber-Physical Systems and Communication Technologies enabling Pervasive Computing (modified [84])

control function. Sensor/actuator nodes are typical cyber-physical systems comprising sensors/actuators and processing and communication components.

1.2 Design Challenges for Networked Embedded Many-Core Systems

A special class of distributed embedded systems are *Networked Embedded Many-Core Systems* which are considered in this thesis. The design and development process for this class of devices needs dedicated methodologies focusing on efficient resource usage. The challenges with respect to these methodologies will be outlined in the following.

The future of networked embedded and cyber-physical systems is sketched in several technology development roadmaps. The *Nationale Roadmap Embedded Systems* of BITKOM [11] identifies several important development challenges towards networked embedded systems:

- Overcome diversity of programming languages and protocol heterogeneity.
- Develop efficient design methods and tools for higher productivity.
- Provide applications with sufficient quality of service and increase efficiency in terms of costs and time.

With respect to challenges in the two design areas *Embedded and Cyber-Physical Systems* and *Communication Technologies*, three additional important technology roadmaps must be taken into account: SOCRADES [124], HIPEAC [21] and ARTIST Design [24]. They will be considered in the following.

Embedded and Cyber-Physical Systems: Many-core chips are of interest for embedded system design [119] because of several reasons. On the one hand, more and more applications running on embedded systems are computationally expensive such as augmented reality. On the other hand, many of such applications are highly parallel and therefore, they can be efficiently executed on many-core systems comprising a large number of "simple" processing cores. The use of many-core chips is not only advantageous to the performance but also for energy consumption as shown in [119, 154]. One class of many-core systems in the HPC (High Performance Computing) context are modern GPUs (Graphic Processing Unit). These GPU chips made a significant paradigm change from ASICs (Application-Specific Integrated Circuits) to a grid of general purpose processing units. This change was mandatory due to the growing demands for "non-graphics" computations (physical effects and artificial intelligence) in games and the resulting requirement for flexible programming. The new paradigm is called GPGPU (General Purpose computing on GPUs). In recent times, these complex GPU chips have also been employed in embedded chips [1, 157] to accelerate computationally expensive applications.

The evolution of computer systems towards multi-core and many-core systems is subject of the HIPEAC roadmap [21]. On the one hand, the roadmap states that the development of single core processors and the availability of higher performance achieved by higher clock frequencies have reached an impasse. On the other hand, the necessary shift towards multi-core and many-core systems is cumbersome because of the change in programming paradigms and the difficulties by utilizing massive parallelism. From a design perspective, the HIPEAC roadmap comprises the following important challenges:

1. Automatically adapting compilers providing optimizations that tailor a certain application code to a particular execution platform should be developed, in order to achieve the best possible performance of the application.

2. Workflows should be designed that map computations efficiently to accelerators such as FPGAs (Field Programmable Gate Arrays) or GPUs. Especially parallelizable applications should be mapped to these systems, to provide higher performance.

3. Optimizing compilers should be created that consider energy consumption of an application or a set of applications as an additional objective beside performance.

4. The design space for high-performance platforms is often large, e.g. due to the possibility to map applications to different processors. Therefore, the conception of efficient design space techniques is mandatory.

Communication Technologies: In the design of distributed and embedded applications, SOA (Service-Oriented Architectures) and middleware libraries based on SOA became popular [9, 62, 124, 142]. The referenced SOAs are utilizing one instance of service-oriented architectures called web services. Web services are common for application designs at server level. SOAs enable a high level of abstraction and a simpler development of distributed applications. Furthermore, the utilization of service-oriented architectures was required to overcome device and protocol heterogeneity by providing a unified interface architecture. Besides these advantages, the porting of web services to the embedded and cyber-physical world is challenging due to many resource constraints in these devices.

The SOCRADES roadmap [124] targets industrial automation systems which comprise "*networked systems made up of smart embedded devices*". For theses industrial automation systems, the SOCRADES roadmap provides a summary of challenges for a framework, including service design, service execution and management functionalities. An excerpt of the challenges is given in the following:

1. Service-oriented architectures and system engineering/management
    (a) The divers resource constraints of embedded and cyber-physical systems demand for a conception of tailored high-level and process-oriented communication and interaction libraries.
    (b) The creation of efficient orchestration mechanisms is mandatory for SOA in order to design complex processes from existing services.
    (c) Context awareness for services which are sensible of the environment should be provided. This is especially important for pervasive computing devices operating in a large variety of different environments. For each environment, e.g. another version of a service can be optimal.
    (d) Dynamic deployment, efficient (re-)configuration mechanisms and an elaborated life cycle management must be developed to work in ad-hoc environments.
2. Sensor/actuator networks
    (a) The consideration of QoS (Quality of Service) in sensor/actuactor network can enable a more efficient use of resource such as communication link bandwidth.
    (b) For decentralizing of control logic, different sensor/actuator nodes should cooperate for a *smart* joint control function. This will increase the efficiency of the controlled systems.

Analogous to HIPEAC, the ARTIST design roadmap [24] gives trends towards efficient resource utilization in middleware environments. In the ARTIST design roadmap it is described that managing power consumption at middleware layer can

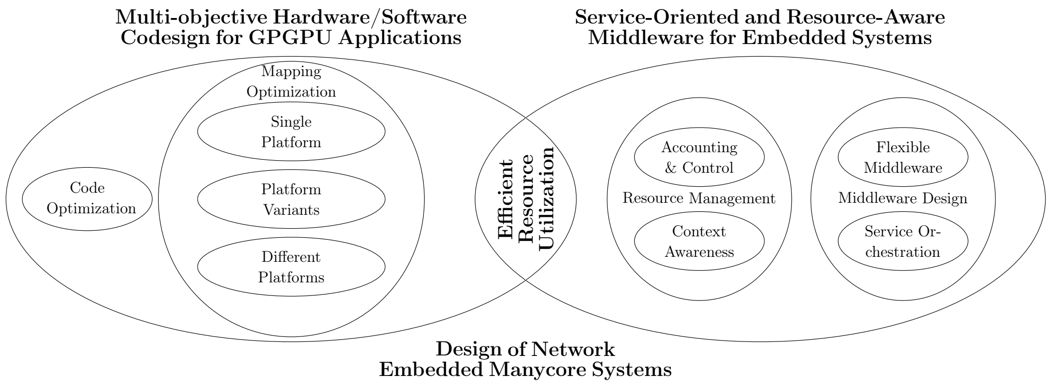

Multi-objective Hardware/Software Codesign for GPGPU Applications
Service-Oriented and Resource-Aware Middleware for Embedded Systems
Mapping Optimization
Single Platform
Code Optimization
Platform Variants
Different Platforms
Efficient Resource Utilization
Accounting & Control
Resource Management
Context Awareness
Flexible Middleware
Middleware Design
Service Orchestration
Design of Network Embedded Manycore Systems

Figure 1.2: Overview on the Overall Design Process

be advantageous to reducing the energy consumption. It can be accomplished by taking into account the system level information, such as the interaction pattern between services or the availability of workload information, which can be available at the middleware layer.

1.3 Contribution of this Work

This thesis addresses the challenges described by providing new methodologies and mechanisms for the embedded system design employing GPUs for application acceleration and the design of service-oriented architectures. An overview on the different research areas in which these methods were developed and utilized in this thesis is depicted in Figure 1.2. The methods can be divided in two classes having an efficient resource utilization as the coherent objective. The first class consists of methods related to a *Multi-objective Hardware/Software Codesign for GPGPU Applications* and the second class comprises methods related to *Service-Oriented and Resource-Aware Middleware for Embedded Systems.* The two resources considered in this thesis are processing resources and energy. Energy consumption has also been identified as one of the major objectives by the technology roadmaps of HIPEAC and ARTIST design.

Multi-objective Hardware/Software Codesign for GPGPU Applications: The need for an extension of the GPGPU application design process towards optimizing compilers arises from the HIPEAC challenge to optimize the mapping process of computationally expensive applications onto accelerators such as GPUs. GPGPU-capable GPUs can already be found in a large variety of embedded SoCs (Systems on Chip)[1, 157]. As can be noticed from the left-hand side of Figure 1.2, the GPGPU application design process comprises two large areas: the mapping optimization and the code optimization. For both optimization types, innovative methods will be introduced and novel evaluation results will be presented in Chapter 5.

The mapping optimization for combined CPU and GPU platforms is realized by means of (multi-objective) design space explorations. These design space explorations can comprise a single platform, a platform family or different platforms. All different types of design space explorations will be introduced in the scope of the *Multi-objective Hardware/Software Codesign for GPGPU Applications*. The novelties for mapping optimizations to be presented in this thesis are:

1. The mapping optimization for GPUs is extended to explicitly consider energy consumption and runtime simultaneously as objectives. Energy consumption and runtime values are provided by an automatic energy consumption and performance testbed.

2. A design space exploration is performed which targets the decision, which platform is the most suitable for a GPGPU application under the constraint of deadlines.

3. At embedded systems level, a design space exploration is conducted which is targeted towards the decision whether or not an integration of a GPGPU-capable graphics card for parallel application acceleration is beneficial to energy consumption or not.

Code optimizations should be automatically performed by an optimizing compiler [90]. Thereby, GPGPU application code can be optimized for a particular platform with the objectives of minimal energy consumption and/or of minimal runtime. These two objectives are targeted with the help of multi-objective optimization techniques. The novelties for code optimizations to be presented in this thesis comprise the following subjects:

1. Energy consumption and runtime profiling data are utilized inside an optimizing compiler for an automatic energy consumption and performance evaluation of optimization process decisions.

2. Local instruction scheduling methods in a compiler are evaluated for the capability to decrease the register pressure and to change load behavior of graphics card pipelines.

3. A global instruction scheduling method optimizes the distribution of memory accesses. This can be beneficial to the energy consumption and the performance of a GPGPU application.

4. A multi-objective evolutionary algorithm is employed in both instruction scheduling processes, which is capable of optimizing both energy consumption and runtime as objectives.

Service-Oriented and Resource-Aware Middleware for Embedded Systems: In standard programming languages, communication cannot be described efficiently. Therefore, communication and middleware libraries are needed. The SOCRADES roadmap introduces a couple of challenges in the context of service-oriented architectures. The design of network embedded many-core systems with the help of SOA is depicted on the right-hand side of Figure 1.2. Its main purpose is to enable the efficient development of large sensor/actuator networks. The service-oriented architecture based middleware framework to be introduced in this thesis, comprises the following novelties:

1. A flexible middleware core is provided which allow the runtime deployment and configuration of services. Additionally, generic *added-value-services* are introduced, which can be re-used in a wide range to application scenarios. The *added-value-services* can be modularly combined.

2. A lightweight but powerful service orchestration mechanism is designed and evaluated.

In addition to the two aforementioned novelties, resource utilization will be considered in this thesis and the design of a flexible resource management mechanism will be introduced. This resource management adapts resource utilization and services to an environmental context. Especially energy consumption is an optimization criterion for that resource management. The novelties for resource management services at middleware layer to be presented in this thesis comprise the following subjects:

1. A resource management service is described, which enables an application designer with the possibility to create energy-aware sensor/actuator networks. This is achieved by extending a publish/subscribe protocol by the capability of adding non-functional requirements.

2. A second resource management service is introduced, which extends a SOA-based middleware with the capability to request resources at runtime and control their utilization at service execution.

1.4 Organization of the Thesis

The following section provides an overview on the structure of this thesis:

- An overview on the different use cases and benchmark suites on which the optimizations and design space explorations are conducted throughout the thesis will be provided in Chapter 2.

- In Chapter 3, an energy consumption and performance testbed will be introduced which is utilized to evaluate the optimization gain and design space decisions in design space explorations.

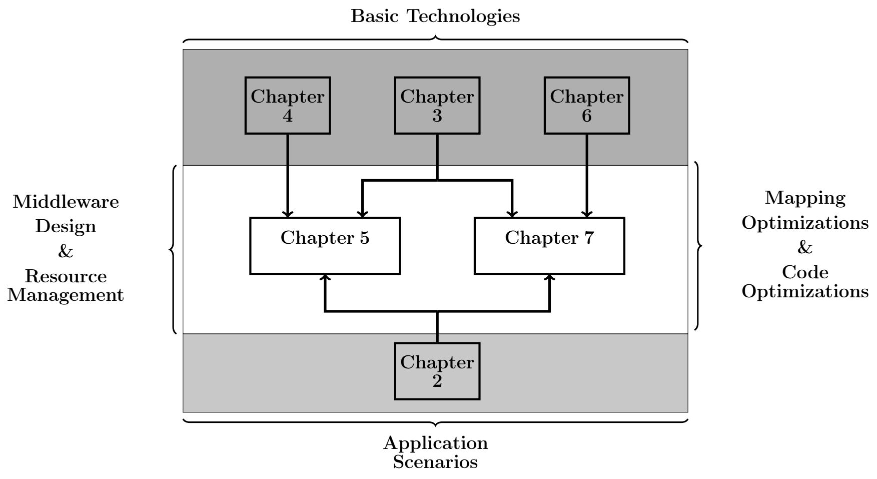


Figure 1.3: Structure of the Thesis

- In Chapter 4, the basics of GPGPU programming will be given along with compiler basics and the fundamental concepts of multi-objective optimization.
- The GPGPU application design process and methods to optimize it, will be described in Chapter 5. In addition to that, suggestions will be made for utilizing many-core chip architectures in the field of embedded systems.
- The basic technologies and example cases for specifications in the field of service-oriented architectures will be introduced in Chapter 6.
- In Chapter 7, a novel service-oriented middleware for embedded and cyber-physical systems and a novel resource management approach in the same field will be introduced.
- The conclusion of this thesis will be described in Chapter 8 and furthermore possible directions for future work will be given.

The dependencies between the chapters of this thesis are depicted in Figure 1.3. The application scenarios presented in Chapter 2 are optimized and act as use cases throughout the thesis, especially in Chapters 5 and 7. The energy consumption and performance testbed introduced in Chapter 3 is also a basic technology utilized in Chapters 5 and 7. Chapters 5 and 7 also have their own basic technology chapters (Chapters 4 and 6 respectively) in which basic concepts needed for describing the methods developed and utilized in this thesis will be introduced.

1.5 Author's Contribution to this Dissertation

§10(2) of "Promotionsordnung der Fakultät für Informatik der Technischen Universität Dortmund vom 29. August 2011" states that a separate list has to be provided in each dissertation revealing the contributions of the author to cooperative research and results. Therefore, the following enumeration lists the contribution of the author to publications which are the basis for different chapters (2, 3, 5 and 7) of the thesis. In these chapters the following publications were taken into consideration:

- Main author: [129, 130, 131, 132, 134, 135, 136, 138]
- Co-author: [4, 79, 121, 132, 133, 148, 149, 150, 151]

The detailed contribution ratio in these publications is as follows:

- Chapter 2: In this chapter use cases and benchmarks are introduced, which are optimized and evaluated in later chapters. The author of this thesis was co-author of topic-related publications. The publication [151] about a real-time GPGPU biosensor analysis pipeline was written by the author of this thesis by about 50 per cent. For publications [79, 121, 132, 148, 149, 150, 151] on the same subject, the author of this thesis wrote only a small fraction. The author of this thesis was one of the main authors of [135] with a contribution of around 50 per cent. This publication introduced the intra-logistics use case. The publication [138] was written by the author of this thesis. The other authors contributed by providing the application context/scenarios and technical/methodical support.

- Chapter 3: In all publications in which the author of this thesis was the main author, the energy and performance testbed was utilized. The publication utilizing the testbed were entirely written by the author of this thesis.

- Chapter 5: The main publications which build the basis of this chapter are [129, 130, 131, 136, 137]. They were entirely written by the author of this thesis. The other authors contributed by providing the application context/scenarios and technical/methodical support. The evaluation in this chapter was also completely done by the author of this thesis, except for the results in Section 5.6.3 which were conducted in cooperation with Andrej Gelenberg. The ideas and concepts for the compiler optimizations were developed by the author of the thesis. However, the implementation of the control flow and data dependency graph and some further compiler level functions were done by Markus Görlich.

- Chapter 7: That chapter is based on several publications [134, 135, 138]. For the publication [138] concerning the context-aware resource management, the author of this thesis was the main writer. The other authors contributed by providing the application context/scenarios and technical/methodical support. The author of this thesis wrote the parts comprising the middleware and the resource management of [135]. The author was also one of the main writers of that publication with a contribution of about 50 per cent. The concepts for the resource management were entirely developed by the author of this thesis. The service orchestration article [134] was mainly written by the author of this thesis. Parts of the introduction, the middleware architecture and deployment were written by Jens Schmutzler. The idea of the service orchestration was completely designed and implemented by the author of this thesis. The middleware presented in the chapter was work of the *MORE* consortium [4, 5, 53, 75, 87, 116, 133, 153] but essential concepts and implementations of the *MORE* middleware core module were done by the author of this thesis. The author of this thesis wrote multiple chapters of several (technical) reports [4, 133] in this context.

Chapter 2

Sensor/Actuator Environments & Applications

In this chapter several use case scenarios and application benchmarks are described which will be optimized with the techniques to be presented in Chapters 5 and 7 of this thesis. The use case scenarios and application benchmarks are evaluated with the energy consumption and performance testbed presented in Chapter 3.

Contents

In the course of this chapter, three use case scenarios are introduced which will be utilized for evaluation purposes in this thesis. In Section 2.1, a biomedical scenario will be presented. An intra-logistics scenario will be introduced in Section 2.2 and a scientific sensor network scenario in Section 2.3. In this chapter, it will be evaluated why these scenarios have been chosen for this thesis and need to be optimized. Furthermore, an overview on different benchmark suites is given (see Section 2.4), which will be also utilized in this thesis to show the general applicability of the proposed optimizations.

2.1 Biomedical Scenario

The first use case scenario is from the field of bio-medics. The development of locally available, specialized virus detection systems becomes increasingly important in the face of worldwide spreading virus infections [48]. Advances in the medical sector facilitate the utilization of optical microscopy for the detection of viruses and – in general – nano-objects. The microscopy can then be used for a rapid and distributed epidemic infection control. A novel technique which can achieve the latter is called *PAMONO* (Plasmon Assisted Microscopy Of Nano-Size Objects) [149, 159]. The high processing requirements of the biosensor demands for a many-core embedded system [149]. The processing requirements will be described by introducing the structure of the biosensor.

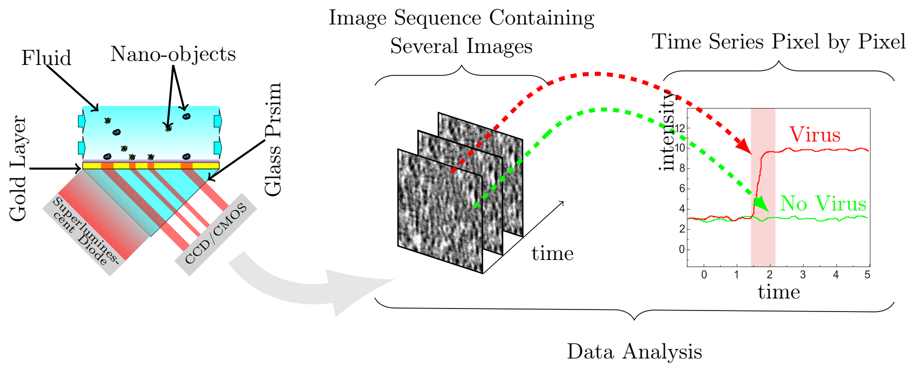


Figure 2.1: Schematic Illustration of the PAMONO Biosensor Experimental Setup & Data Analysis (modified [149])

The fundamental concept of the *PAMONO* biosensor exploits surface plasmon effects. These effects allow the identification (by optical methods) of nano-objects that are smaller than the wavelength of the light, such as viruses or fine particles of auto-mobile emissions. In order to induce this effect, a super-luminescent diode is utilized to activate surface plasmons in a thin gold layer. The gold layer is deposited on a glass prism as depicted in Figure 2.1. The reflected light is projected onto an – at least – 12-Bit CCD or CMOS camera chip, where a high resolution image is acquired. A liquid containing nano-objects is pumped through a flow cell, attached to the sensor surface. Nano-objects being close to the sensor surface are bound to the gold layer by corresponding selective antibodies or by electrostatics. The reflected intensity on the sensor surface increases locally, resulting in a bright mark appearing in the image. The increase in intensity can be identified by evaluating the time series of each pixel as depicted on the right-hand side of Figure 2.1. The detection capability of the PAMONO biosensor is influenced by several artefacts, e.g. unevenness of the gold layer, production tolerance of the optical sensor. One of the most important issues for the wide range usability of this virus detection method is the processing and the analysis of the acquired images in real-time. This is demanded by the circumstance that the result of a detection in progress should be visualized online, while inserting the specimen. An image processing and analysis pipeline is designed as a GPGPU application to accelerate the detection algorithm of nano-objects and to meet realtime requirements.

The pipeline is – from an image processing and analysis view – divided into three processing steps as depicted in Figure 2.2. The sets of images provided by the camera are transferred to the GPU memory without any image processing on the CPU. Each set of images, comprising b images, is then processed on the GPU and the results are transferred back to the host main memory. The algorithms are highly

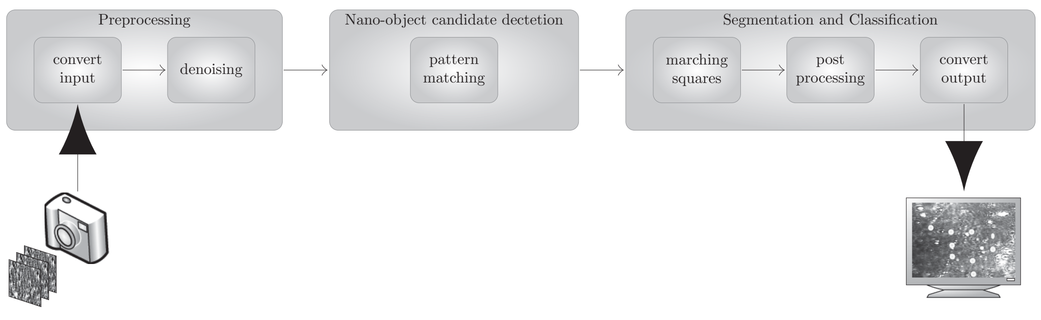


Figure 2.2: Schematic Illustration of the *PAMONO* GPGPU Image Processing and Analysis Pipeline (modified [149])

data parallel and as a result well-suited for the execution on a graphics card [151]. The processing is done on different presentations of the data. There are algorithms which work on time series basis, meaning that for $m \cdot n$ pixels sized images, $m \cdot n$ time series can be processed concurrently. The length of the time series is b. The other type of algorithms work on single images. These single images are divided into sub frames comprising several pixels. The processing can then be done for the sub frames concurrently. The processing steps on the GPU comprise – in that order – the preprocessing, the detection of time series (*pattern matching*) comprising nano-objects and the aggregation of pixels to connected areas (polygons), respectively the classification. *Pattern matching* and *denoising* are both working on a time series basis. The *Segmentation and Classification* processing steps work on single images.

As a summary, the use case scenario demands efficient design in the following area:

Many-Core ES: A high data rate and data parallel algorithms demand for a design of an efficient processing system (see Chapter 5). Especially, the choice of the most appropriate platform is interesting for this use case.

2.2 Intra-Logistics Scenario

The second use case scenario is from the field of automation systems. Traditional conveyor belt systems comprise central control mechanisms such as PLCs (Programmable Logic Controllers). They administrate a large variety of sensors and actuators. In recent times, efforts have been made which try to substitute this infrastructure by a decentralized SANET (Sensor/Actuator NETwork) [135, 147]. The network concept of this SANET will be introduced in the following, preceded by the description of the processing demands.

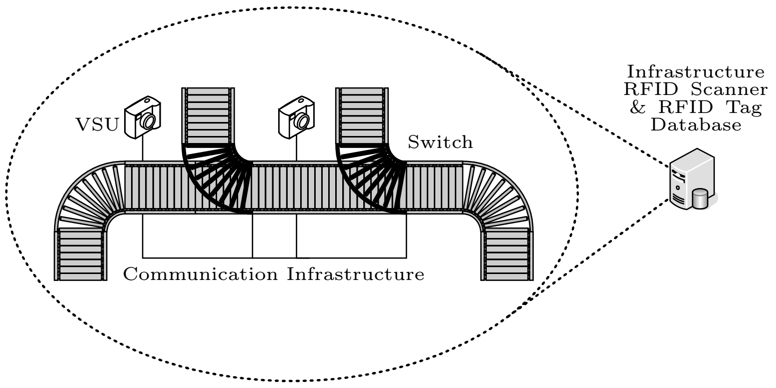

Infrastructure RFID Scanner & RFID Tag Database
VSU
Switch
Communication Infrastructure

Figure 2.3: Concept of the Intra-logistics Scenario (modified [149])

The SANET controlling the conveyor belt system (depicted in Figure 2.3) comprises different sensor nodes and actuator nodes, directly connected with the help of a middleware. This is fundamentally different, but more efficient, compared to traditional systems where there is only a central controlling instance. The routing of the bins or parcels in that conveyor belt system is done by identifying a QR code [44] on the parcel by a camera system and by requiring routes for that QR code from a central database. In addition to a QR code, the parcels are equipped with a RFID tag equipped with the same information as the QR code. Identifying QR codes in arbitrary images is a challenging task [14, 102] and therefore, the data store in a central RFID tag database is utilized to verify whether a detected and decoded QR code is available or whether there were errors in the detection. The RFID tags are recognized by a few RFID scanners. The switches of the conveyor belt systems are controlled by the camera systems. The camera systems are called Visual System Units (VSU).

The considered conveyor system as depicted in Figure 2.3 has different types of hardware platforms, based on the intended purpose. At the topmost level, large computer systems are used which run e.g. the RFID central database. At a lower level, sensor nodes such as VSUs are used which control single switches locally. They are connected among each other, to the controlled switch and to the central RFID database. The processing demands for detecting and processing QR codes are described in the following.

An example QR code is depicted on the left-hand side of Figure 2.4, next to the camera. A QR code is a two-dimensional code. As can been seen there, the QR code is built of three position patterns, an optional alignment pattern and the code area. The position patterns specify the orientation and the size of the marker. The alignment pattern is used for handling possible distortion effects – for detailed information refer to [102]. The data is encoded inside the black-white-pattern in a

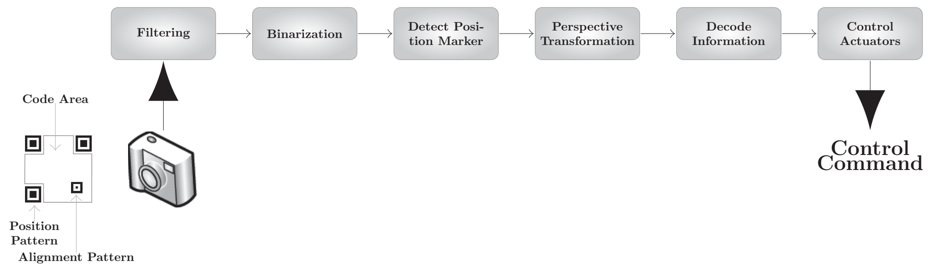


Figure 2.4: Processing Concept of Intra-logistics Scenario

hierarchical fashion. The atomic unit of a QR code is a module comprising one or more pixels. The number of pixels inside a module depends on the scaling of a QR code. The QR code size qr_{size} is defined by the number of modules in the code:

$$qr_{size} = (17 + 4 \cdot i) \cdot (17 + 4 \cdot i), \tag{2.1}$$

where $1 \leq i \leq 40$. For $i = 40$, up to 4296 alphanumeric characters can be encoded [44].

The extraction and processing of such a QR code is a challenging task for an embedded system [14, 102]. Real-time processing of the images is necessary, in order to activate actuators in time. The image is acquired by a standard digital camera (see Figure 2.4). In the first phase, the image is filtered to remove artefacts etc. In the second phase, the image is binarized. In the third phase, the position patterns of the QR code are detected. Based on the position patterns and the alignment pattern, the code area is extracted and QR code size qr_{size} is determined. The QR code is then transformed perspectively and scaled to the calculated QR code size qr_{size}. In the next to last phase, information is decoded from the QR code and in the last phase, the routing decision is performed by the controlled switch. The algorithms of the first four phases can be parallelized and are predestined for execution on many-core systems.

As a summary, the intra-logistics scenario demands for an efficient design in the following areas:

Many-Core ES: Data parallel image processing and analysis algorithms demand an efficient design of the processing system (see Chapter 5).

Sensor/Actuator Network: The decentralized control in the use case scenario requires an efficient middleware infrastructure (see Chapter 7) and the consideration of quality of service.

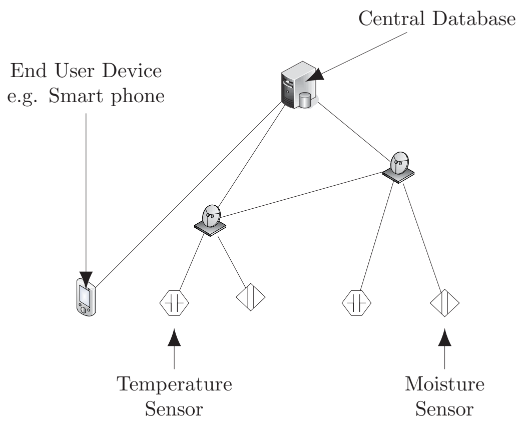


Figure 2.5: Concept of the Mitigation Management Scenario (modified [63])

2.3 Scientific Sensor Network Scenario

The third use case scenario is from the field of scientific data acquisition. The gathering of scientific data – in the *Mitigation Management* project of TU Dresden [57] – from sensors at remote locations is time-consuming and done manually. The use of a sensor network will provide a more sophisticated way of utilizing the data and will enable ad-hoc usability. The challenges for sensor network design will be described in the following.

A number of sensor nodes are distributed at remote locations in the forest. The sensors include temperature, moisture and gas sensors. In the past, data from these sensors were transferred manually to scientists for evaluation. The new architecture of this scenario, derived from user requirements, is depicted in Figure 2.5. As can be seen in this figure, there is a central data server storing all measured data. The data on this central data server is provided to end user devices such as smart phones and personal computers as raw data or processed data. The second purpose of the data server is to act as a data sink for the sensor data. Therefore, the data server communicates with remote devices in the forest. Each of them comprises one or more sensors and a wide-area link, e.g. a UMTS or GSM link. The limited bandwidth demands for an efficient design towards local processing capabilities.

As a summary, the scientific sensor network scenario demands for efficient design in the following area:

Sensor/Actuator Network: The decentralized nature of this use case scenario requires an efficient middleware infrastructure (see Chapter 7).

2.4 Application Benchmarks

The optimization and design space techniques presented in this thesis are not restricted to the described use case scenarios but can also be applied the other benchmarks. Several benchmark suites and single benchmarks are used for evaluation in this thesis for demonstrating the applicability of the proposed GPGPU application design techniques (see Chapter 5).

The following three benchmark suites comprise a large variety of application domains such as: medical imaging, data mining, image processing, pattern recognition, simulation etc. The benchmark characteristics cover benchmarks with and without extensive main memory utilization and benchmarks which are more computationally expensive:

- Nvidia CUDA examples [97]
- VSIPL-GPU-Library [58]
- Rodinia benchmark suite [33].

For comparison between applications written for CPUs and applications written for GPUs, several manually optimized benchmarks have been utilized:

1. *Matrix Multiplication*: CPU [125], GPU [99]
2. *Fast Fourier Transform*: CPU [56], GPU [99]
3. *Air Pollution Simulation*: CPU [89], GPU [89]
4. *Range-Doppler Algorithm*: CPU [58], GPU [58].

Chapter 3

Energy Consumption and Performance Testbed

In Chapter 2, the use case scenarios optimized in this thesis were outlined. In this chapter, the energy consumption and performance testbed is described which will be used in Chapters 5 and 7 to evaluate the optimization gain or single space points in a design space exploration. It is followed by Chapter 4, in which basics on the GPGPU application design process and multi-objective optimizations will be provided.

Contents

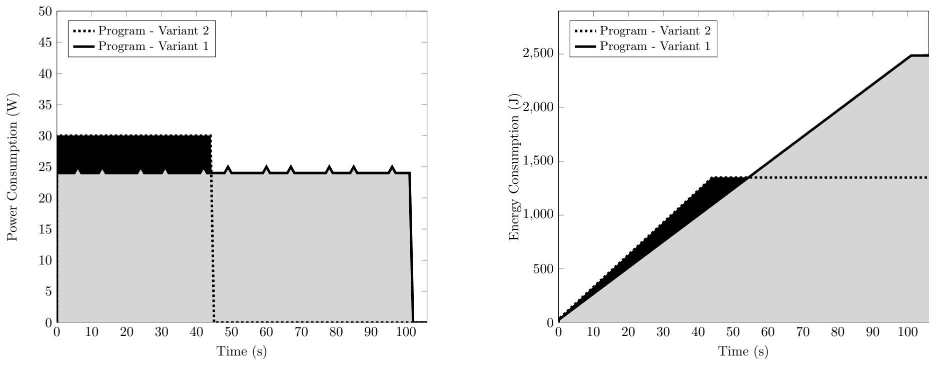
In the introduction (see Section 3.1), basics for energy and power efficiency will be given. In Section 3.2, the testbed utilized in this thesis will be presented.

3.1 Introduction

As mentioned in Chapter 1, energy-aware design is mandatory in the embedded and cyber-physical systems domain [21, 24] because of several different requirements [85]:

- Usefulness/Usability: Smart phones and mobile devices are omnipresent but their usability is directly impacted by the capacity of their batteries and the efficient use of energy.

- Green Computing: Green computing targets energy-aware design throughout the complete life cycle of embedded and cyber-physical systems in order to make their use possible in the face of scarce resources. For example, a forecast [112] estimates 2.2 millions worldwide operating ATMs (Automatic Teller



(a) Power Consumption (b) Energy Consumption

Figure 3.1: (a) Power Consumption and (b) Energy Consumption

Machines) in 2016. An energy-aware design of such stationary devices which consumes several tens of watts and operates 24/7 is beneficial in terms of green computing.

- Cost Reduction: Due to pervasive and ubiquitous computing, the use of embedded and cyber-physicals systems also emerges in the automation and business sectors. The utilization of many of them makes their energy consumption an emerging sector for saving costs. Energy-aware designs are therefore beneficial.

In order, to understand the optimizations done in this thesis, the definition of the energy consumption is important. Especially the distinction between power-aware and energy-aware optimization is important as they are not the same [84].

The energy consumption E_{cont} for a time interval $[t_{start}, t_{end}]$ is defined as

$$E_{cont} = \int_{t_{start}}^{t_{end}} P(t)dt. \tag{3.1}$$

$P(t)$ is a power consumption measured in watt (W) at time stamp $t \in [t_{start}, t_{end}]$. The unit of the energy consumption E_{cont} is Joule (J). For discrete measurements, the energy consumption e_{disc} can be calculated by using a Riemann sum:

$$E_{disc} = \frac{1}{f_s} \sum_{t \in \{t_{start}, \ldots, t_{end}\}} P(t) \tag{3.2}$$

in which f_s is the sampling frequency (Hz) of the measurements, $P(t)$ the power consumption measured in watt (W) at time stamp t, t is a sampling point in the set of measurement sampling points $\{t_{start}, ..., t_{end}\}, t_{start} \leq t_{end}$. t_{start} is the starting point for the measurements and t_{end} is the end time for the measurements. The unit of the energy consumption E_{disc} is Joule (J).

As can be easily derived, optimization for power consumption is not the same as optimization for energy consumption. In Figure 3.1 on the left-hand side, a power consumption over time diagram showing two program variants (*Variant 1* and *Variant 2*) is given. As can be seen, during the entire runtime, the power consumption of *Variant 2* is higher than the power consumption of *Variant 1*. That means, that *Variant 1* is more power efficient than *Variant 2*. On the right-hand side of Figure 3.1 the energy consumption – in the interval [0,100] – of both program variants can be seen. It can be noticed that due to the smaller runtime of *Variant 2* its energy consumption is lower than the energy consumption of *Variant 1*. Therefore, *Variant 2* is more energy efficient than *Variant 1*.

Besides a profiling-based approach, such as the one described later in this chapter, several power and energy consumption models exist. An excerpt from common literature reveals several types of energy or power models:

- Measurement-based Models: The authors of [123] and [139] provide fine-grained energy models on instruction level. They were both derived by measurements of the energy consumption for a single instruction type and for different sequences of instructions. The latter is important as different sequences lead to different energy consumptions of an instruction type. The utilization of power/energy models is not restricted to embedded systems. For example, also models for other architectures such as GPUs exist. A statistical power model for determining the average power consumption while running a GPGPU program was developed in [82]. With the help of the runtime and this power model, the energy consumption of a GPGPU program can be calculated.

- Specification-based Models: The energy model of the authors of [122] is based on the utilization of the data sheets of the different functional units such as memory or processors. Based on this information the energy consumption for memory accesses and for processor cycles are estimated.

- Analytical Models: Especially, memory accesses can be modelled with analytical methods, such as CACTI [104]. CACTI describes the memory structure by a gate-level model and estimates the power consumption, access time and circuit area with this model. Other analytical models describe the leakage and dynamic power consumption of transistors [32, 126].

3.2 Testbed

In this section, the testbed which will be utilized in the thesis will be presented. Therefore, the architecture of the testbed will be given in Section 3.2.1. In Sec-

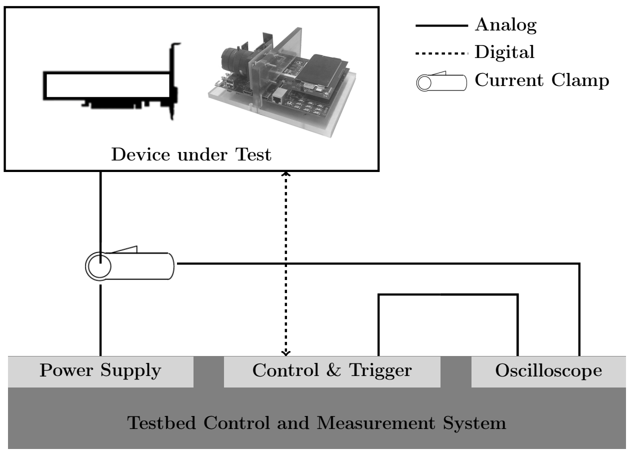


Figure 3.2: Energy Consumption and Performance Testbed

tion 3.2.2, the mechanism for measuring energy consumption for application is described and in the end (see Section 3.2.3) some results for the idle power consumption of some hardware platforms are given.

3.2.1 Architecture

The two major goals of the optimizations to be presented in Chapter 5 and 7 are energy consumption decrease and performance increase. Both can be measured with an energy consumption and performance testbed as depicted in Figure 3.2. There is always a system controlling the tests – called testing system – and DUT (Devices Under Test). For getting the power consumption P_d for a devices, the current I_d at the power supply lines is measured. The power supply must be a DC power supply. A current clamp measures the amount of current I_d running through the probed lines by a proportional voltage which can then be measured employing an oscilloscope. The power consumption P_d can then be calculated by

$$P_d = I_d \cdot V_d, \tag{3.3}$$

where V_d is the voltage of the power supply line.

The energy consumption and the runtime of an application are evaluated as follows: The runtime interval $[t_0, t_{run}]$ is delimited by a trigger signal which is initiated by the testing system and can be measured at the output of the RS232 serial port. This is also performed by the oscilloscope of the testing system. A trigger signal is triggered by marking it in the source code before start. This will be explained in Section 3.2.2. If the DUT is an acceleration device such as a graphics card, the RS232 can be directly controlled. If the DUT is an external device, the RS232 port is controlled over a socket connection or by the RS232 of the external device.

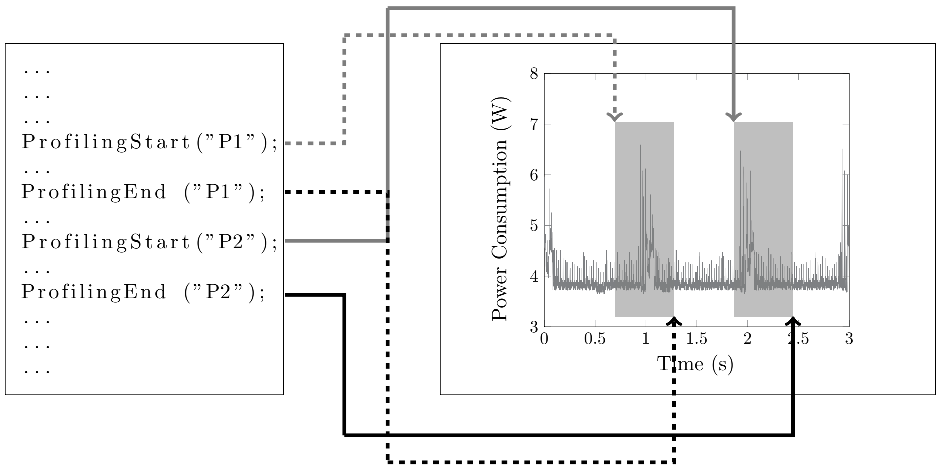


Figure 3.3: Source Code Annotated Profiling

3.2.2 Source Code Annotation

In order to measure the energy consumption of (parts of) an application, the time frame must be available in which the power consumption of the hardware platform must be measured. Therefore, several markers can be inserted into the source code of an application as depicted in Figure 3.3. As can be seen in this figure, the start and the end of the power consumption measurements are marked with a start marker *ProfilingStart* respectively with an end marker *ProfilingEnd*. In order to assign a certain source code part to a functionality, an identifier *id* is attributed to a pair of *ProfilingStart* and *ProfilingEnd*. If the identifiers of two or more application code sections are the same, average values are returned. If an application is running on remote devices, additional functions are required in the source code to force the application code section to run in the desired time interval. More details on this are given in Section 4.1. In Figure 3.3 two application code sections $P1$ and $P2$ of an application P_a on a platform P_l were measured. The testbed can provide the energy consumption

$$E_{P1}^{P_a,P_l} = energy(P_a, P_l, t_{P1_{start}}^{P_a,P_l}, t_{P1_{end}}^{P_a,P_l}) \tag{3.4}$$

respectively

$$E_{P2}^{P_a,P_l} = energy(P_a, P_l, t_{P2_{start}}^{P,P_l}, t_{P2_{end}}^{P_a,P_l}). \tag{3.5}$$

The runtimes of $P1$ and $P2$ are then given by

$$r_{P1}^{P_a,P_l} = runtime(P_a, P_l) = t_{P1_{start}}^{P_a,P_l} - t_{P1_{end}}^{P_a,P_l} \tag{3.6}$$

and

$$r_{P2}^{P_a,P_l} = runtime(P_a, P_l) = t_{P2_{start}}^{P_a,P_l} - t_{P2_{end}}^{P_a,P_l}. \tag{3.7}$$

The start triggers such as $t_{P1_{start}}^{P_a,P_l}$ always start at 0 and the end triggers like $t_{P1_{end}}^{P_a,P_l}$ are always relative to the start triggers.

Graphics Card Card	Cores	Shader Clock Clock (MHz)	Memory Interface Width (Bit)	Idle Power Consumption (W)
8400GS	8	1400	64	up to 4
NextION	16	1400	64	5.4 - 6.6
9500GT	32	1400	128	15.8 - 20.3
9600GT	64	1625	256	33.48 - 43.2
GTS250	128	1836	256	24.0 - 45.6

Table 3.1: Performance Specification of Devices under Test (data from [100] – except for power consumption)

3.2.3 Exemplary Hardware Platform Test Configuration

This evaluation explains, how a particular platform class can be evaluated. Exemplarily, the testbed configuration for graphics cards is given in the following.

A typical PCI Express graphics card is powered via different power supply connections. The eight power supply lines (12 Vand 3.3 V) of the PCI Express bus provide a maximal power of 75 W. For graphics cards, which have a higher power consumption, additional power supply lines (12 V) are directly connected to the main power supply unit of the system. For measuring the power consumption of the graphics card, current clamps at the bundled 12 Vpower lines and at the bundled 3.3 Vpower lines are utilized. For different graphics cards the idle power consumptions have been measured. The results are listed in Table 3.1. Because of the graphics card's power consumption saving techniques, for some of the graphics cards, a power consumption range is given. As can be seen, the idle power consumption increases with the number of built-in cores.

Chapter 4

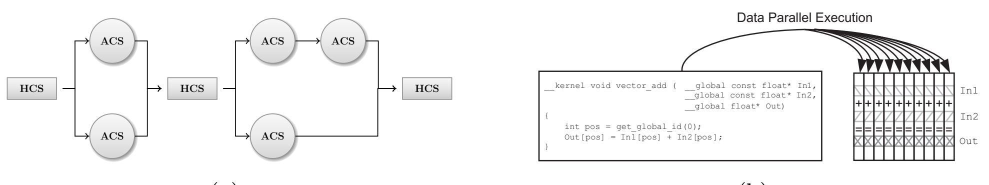
Optimizations for GPGPU Applications: Basics

After presenting the energy consumption and performance testbed in Chapter 3, in this chapter the basics of the GPGPU application design process and multi-objective optimizations will be outlined. Basic technologies and methods are provided for Chapter 5 in which optimizations and design space explorations for GPGPU applications will be described.

Contents

In this chapter, the programming and mapping process of Nvidia graphics cards with the help of *CUDA C* and *OpenCL C* (see Section 4.1) will be described. The major focus will then be the compilation process of GPGPU applications in particular. In this area, different intermediate representations will be delineated and it is shown how applications will be optimized with the help of these intermediate representations.

In addition to the GPGPU application design process, a heuristic optimization technique, called GA (Genetic Algorithms), will be outlined in Section 4.2. It will be described how genetic algorithms can be used to solve optimization problems and how genetic algorithms are specified. Furthermore, multi-objective genetic algorithms will be presented which allow to handle optimization problems pursuing two or more objectives. In the scope of that presentation, two example algorithms, SPEA2 and NSGA-II will be shown which both utilize the elitism paradigm [74].



```
__kernel void vector_add (  __global const float* In1,
                            __global const float* In2,
                            __global float* Out)
{
    int pos = get_global_id(0);
    Out[pos] = In1[pos] + In2[pos];
}
```

(a) (b)

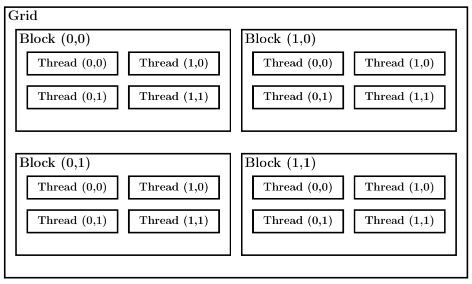
Figure 4.1: (a) Application with accelerated parts (ACS) and non-accelerated parts (HCS) and (b) Data Parallelism with OpenCL

4.1 Programming GPGPU-based Many-Core Systems

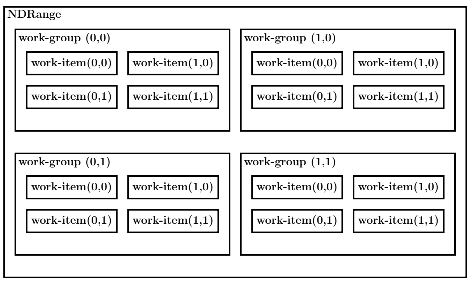
In order to tackle high throughput data-parallel applications in embedded systems, parallel processing capabilities are indispensable. In the past, this was done by designing application-specific and parallel processing accelerators such as ASICs or FPGAs. They were used to accelerate parts of an application which can be processed in parallel. An example for such a kind of application is depicted in Figure 4.1(a). The application parts denoted with HCS (Host Code Sections) are running on the host system which controls the accelerator. The application parts denoted by ACS (Accelerator Code Sections) are running on the accelerator. Typical examples for such systems are up-to-date television sets where the video and audio decoding is done on the accelerator and the other parts are running on the general-purpose CPU [140].

Since 2003 [29], graphics processing units have been equipped with programmable processing elements, capable of executing general-purpose applications and easily programmable by application designers. This capability is often summarized by the term GPGPU (General Purpose Computing on Graphics Processing Units). The parallel processing capability and the availability of such GPUs in a wide range of hardware designs made them attractive for low-cost high performance computing and as an accelerator technique. In this section two programming concepts for GPUs are described: *CUDA C* and *OpenCL*. Both concepts are used in this thesis. While the first is targeted towards the programming of Nvidia graphics cards only, the latter is designed to program a wide range of accelerator devices including multi-core CPUs [69], graphics cards [96] and FPGAs [6].

The section is structured as follows: Firstly, the programming concepts of *CUDA* and OpenCL are delineated in Section 4.1.1. Secondly in Section 4.1.2, some important hardware features of Nvidia graphics cards are described and finally, in Section 4.1.3, the mapping process of a CUDA application is outlined.



(a)



(b)

Figure 4.2: (a) *CUDA* Thread Hierarchy and (b) *OpenCL* WorkItem Hierarchy

4.1.1 Programming Concepts

Both programming concepts, *OpenCL* and *CUDA C* can be used for specifying parallel applications. There are two major types of parallelism: data parallelism and task parallelism. Data parallelism, on the one hand, means that "*similar operations ... are performed on elements of a large data structure...*" [38]. Task parallelism, on the other hand, means that, "*entirely different calculations can be performed on either the same or different data*" [38]. The focus of this thesis are data parallel applications and therefore, this section is limited to this aspect of parallelism only. *OpenCL* and *CUDA C*, both start their application specification by writing single-threaded code (as shown on the left hand side of Figure 4.1(b)), called *kernel*. A *kernel* is written in a C99 dialect including OpenCL and CUDA specific programming language qualifiers, data type modifiers and some restrictions such as the unavailability of recursion.

An example for a data-parallel application is depicted in Figure 4.1(b). In this figure, a vector addition of two arrays *In1* and *In2* into a third array *Out* is shown. As can easily be noticed, the vector addition can be done independently over the length of the array. An important concept of *OpenCL* and *CUDA* is that of a position identifier – here *get_global_id(0)*. Position identifiers allow a single thread to identify the position in the data where it can work on.

In the example in Figure 4.1(b) also the need of another important feature of *OpenCL* and *CUDA* can be derived. The arrays in this example can be quite large and therefore the processing must possibly be partitioned in order to be executed in the face of bounded processing and memory resources. For this, *OpenCL* and *CUDA* include a thread hierarchy as depicted in Figure 4.2(a) respectively in Figure 4.2(b). The hierarchy can be one-, two- and three-dimensional. In Figures 4.2(a) and 4.2(b) the two-dimensional case is depicted. A single runtime instance of a *kernel* is called *thread* in *CUDA* and *work-item* in *OpenCL*. One step higher in the hierarchy level, the *thread*s and *work-item*s are bundled in a group called *thread block* respectively *work-group.* These groups are then again bundled in a larger group called *grid* in *CUDA* and *NDRange* in *OpenCL.* With position identifiers as described above, it is possible to identify a *thread* (*work-item*) and the *thread block* (*work-group*) at runtime to coordinate the processing on the same data set.

For processing data on a common data set, the data set has to be stored in a memory available for *threads* and *work-items* . The different memory spaces are depicted in Figures 4.3(a) and 4.3(b). As can be seen in this figure, OpenCL provides two types of main memory, *global memory* and *constant memory.* Both memories are cached. The *global memory* is readable and writeable from a *work-item* and the *constant memory* is only readable. The *global memory* can be used for *work-item* to *work-item* communication in the whole *NDRange.* In order to communicate inside a *work-group*, a fast local memory is available. To provide memory consistency for this communication, an explicit barrier synchronisation statement must be used in a *kernel.* The last type of memory in *OpenCL* is the so-called *private memory.* This memory is accessible by a *work-item* only. This concept also applies to *CUDA C* by mapping the elements as depicted in Table 4. *CUDA* provides two more memory

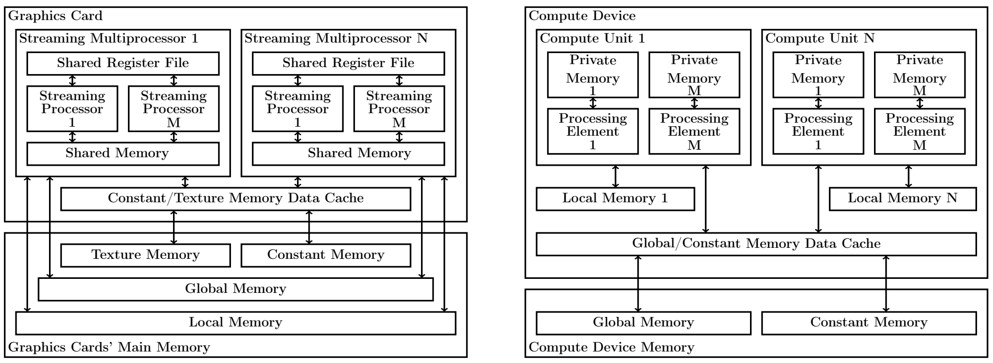
OpenCL	*CUDA*
Global Memory	Texture & Global Memory
Constant Memory	Constant Memory
Local Memory	Shared Memory
Private Memory	Registers

Table 4.1: Mapping Memory Concepts between *OpenCL* and *CUDA*

spaces *local memory* and *global memory.* Both memory spaces are unchached main memory. While *local memory* is only readable and writeable from a *thread*, *global memory* can also be used for *thread* to *thread* communication.

4.1.2 Hardware Structure and Runtime Concept

The programming language view on *OpenCL C* and *CUDA C* was described in the last section. In this section now the mapping of these programming language concepts on Nvidia hardware is explained. As this is similar for *OpenCL* and *CUDA* for Nvidia graphics, the mapping will only be outlined for *CUDA*. A schematic depiction of a Nvidia graphics card is presented in Figure 4.3(a). The atomic functional unit executing a thread is an SP (*Streaming Processor*). A *thread*'s *thread block* is allocated to a particular SM (*Streaming Multiprocessor*) comprising several *Streaming Processors* and a scratchpad memory. On one *streaming multiprocessor* a subgroup of a *thread block* called *warp* is running. A *warp* contains exactly 32 *threads* and they are executing the same instruction. This method is called SIMT (Single Instruction Multiple Threads). The different *warps* are the scheduling unit of a *Streaming Multiprocessor.* They are scheduled using a scoreboard mechanism [67].



(a) (b)

Figure 4.3: (a) *CUDA C* Memory Hierarchy (modified [98]) and (b) *OpenCL C* Memory Hierarchy (modified [61])

This mechanism is called warp scheduling at Nvidia. Warp scheduling enables a *Streaming Multiprocessor* to issue other *warps* while some *warps* are waiting for the results of instructions or for data from the *global memory.* If enough warps are available, this parallel instruction and memory pipeline mechanism can hide memory latency arising from the uncached *global memory.* [96]

4.1.3 GPGPU Application Mapping

In this section, Nvidia's GPGPU application mapping process for *CUDA C* is described and the compilation process is delineated.

4.1.3.1 General

Thread blocks are allocated – depending on the resource usage – to a *streaming multiprocessor.* This is done by the *CUDA* runtime based on statically derived parameters for shared memory and number of registers. This is accomplished as follows (depicted in Figure 4.4): First of all, the application designer specifies manually how many threads are in a block TPB (Threads Per Block). For the allocation of the blocks, it is then statically evaluated by the compiler, how much shared memory one thread needs and how many registers RPT (Registers Per thread) are needed maximally [96]. Finally, the number of blocks is evaluated which can be allocated to an SM depending on the resources. If e.g. three blocks are allocated, then RPT $\times$ TPB $\times 3$ registers are used. Furthermore, according to [96] registers are allocated in banks, so the actual register allocation count can be higher. There must be at least resources (shared memory and registers) to run one block for a program. The important performance indicator in this context is called *occupancy* [96]. It describes how efficient the different SMs of a GPU are utilized in terms of running threads. The *occupancy* o is described by a value between 0 and 1, in which $o = 1$ means, that a SM is fully utilized.

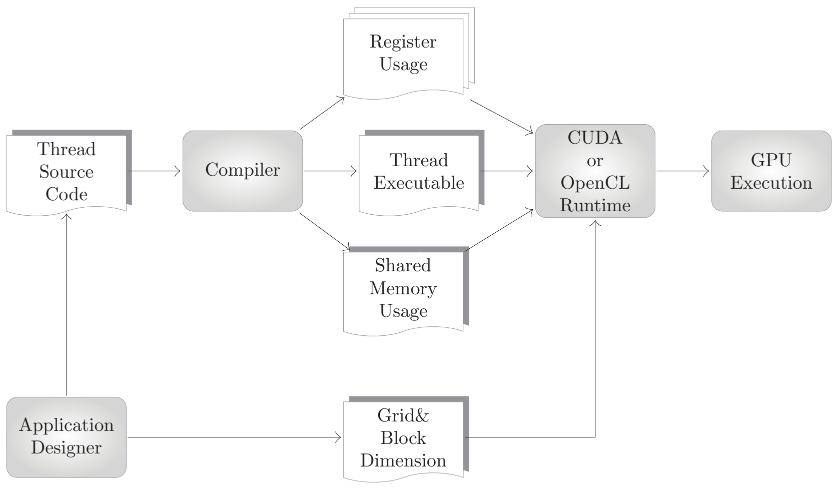


Figure 4.4: Overview on GPGPU Application Mapping and Execution Process

4.1.3.2 Compilation

The compilation toolchain for *CUDA* [101] is depicted in Figure 4.5. It starts with the specification of the application for the host and the graphics card. The host code and the GPU code are separated by the *CUDA frontend (cudafe).* The GPU code is then compiled in several steps. At first, it is preprocessed by *cudafe* again. Then it is transformed by the frontend of the *NvOpenCC* [97] to its intermediate representation, called WHIRL. WHIRL comprises a set of IRs (Intermediate Representations) from HIR (High-level IR) over MIR (Mid-level IR) to LLIR (LLow-Level IR). After the transformation to WHIRL, all functions are inlined. The most important part in the *NvOpenCC* is the backend where several optimizations take place. In the end, the WHIRL representation is transformed into a *PTX* representation. *PTX* is a MIR tailored towards the use on Nvidia graphics cards for parallel programming [96]. This *PTX* code is transformed to GPU machine code by *PTXas* with the help of register allocation, code selection and several low level optimizations. The GPU machine code is then embedded into the host code and compiled along with other host code files by a compiler for the host platform.

The GPU code can be optimized at several stages. Especially the two intermediate representations *WHIRL* and *PTX* of the compilation toolchain can be utilized for optimization purposes. An interesting optimization technique is IS (Instruction Scheduling) which works on low-level and mid-level intermediate representations. IS changes the order of instructions in the IR (while respecting the semantics), to tune the performance of an application. An example for this will be given in Section 5.3.2. The semantics is maintained by employing data dependency and control flow graphs. Instruction scheduling can be divided in two categories: local scheduling and global scheduling. Within local scheduling techniques, instructions are only

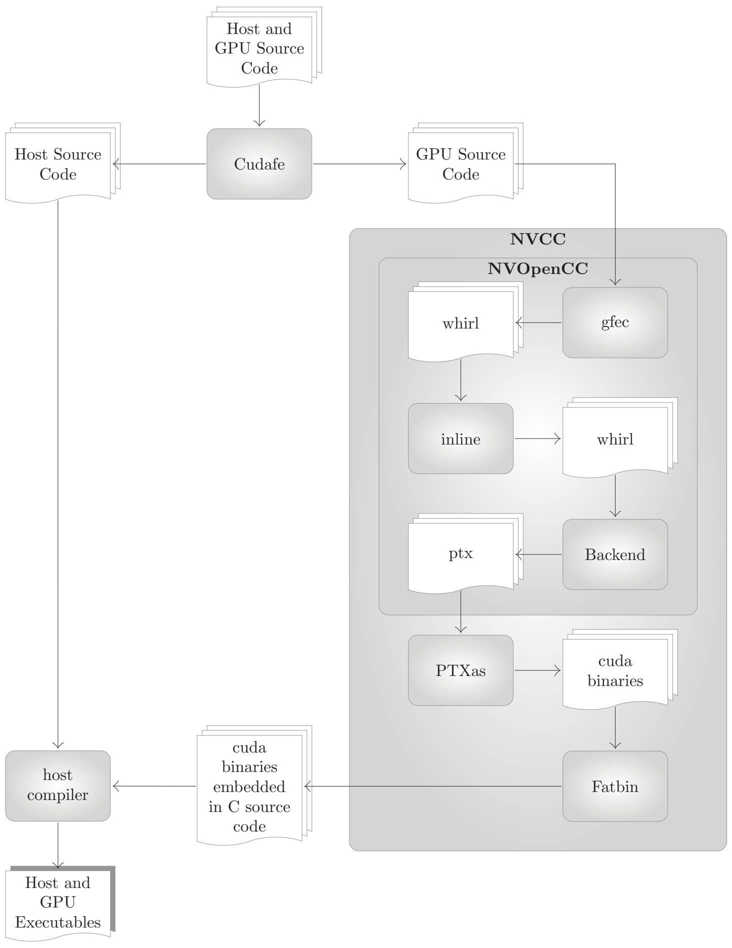


Figure 4.5: *CUDA* Compilation Toolchain (Inspired by [101])

scheduled within a single basic block (Definition 5). Global scheduling is more powerful because it also allows the movement of instructions from one basic block to another and the scheduling inside a basic block but it has several disadvantages, such as the utilization of compensation code, which is often needed for preserving program semantics while moving instructions over basic block boundaries in the face of divergent control flows. Compensation code is a duplicate of the original instructions.

4.2 Genetic Algorithms

Many optimization techniques, like the Instruction Scheduling presented in the former section, are NP-equivalent [90]. Therefore, efficient heuristics such as GAs (Genetic Algorithms) are needed which find good solutions in the solution space but require only polynomial runtime. Genetic algorithms are population-based op-

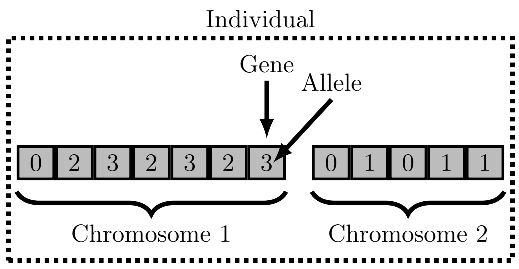


Figure 4.6: Genetic Algorithms - Individual Structure

timization methods [81] which utilize biology as a metaphor for describing heuristics. Other biology-inspired algorithms are e.g. ant colony optimization [47] or optimization techniques based on swarm intelligence [17]. The principle of genetic algorithms is simple. A population of individuals exists. An individual represents a solution of the optimization problem. Each individual is assessed with regard to a fitness criterion. Based on this criterion and affected by their environment, the population evolves and unfit individuals are discarded. At the end, the fittest individual of the population survives, which represents the optimization result.

In addition to the utilization as a heuristic, GAs have another advantage. They can be tailored towards a use in environments where the optimization function is unknown but can be evaluated [81]. For example, the energy consumption of an application can be evaluated, but the contribution of the different functional units to that consumption is unknown. This is especially important for the optimization techniques to be presented in Chapter 5.

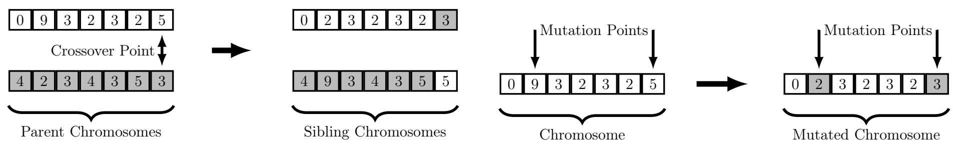
First of all in this section, the terminology of genetic algorithms is summarized in Section 4.2.1, and then multi-objective genetic algorithms are described.

4.2.1 Specification

Genetic algorithms describe and solve optimization problems with the help of biology – especially genetics – mechanisms [81]. A genetic algorithm comprises the following entities (also depicted in Figure 4.6):

Definition 1 *(Genetic Algorithm Entities)*

- ***Solution Space*** $\mathbb{X}$*: Is a* n*-dimensional space.* n *depends on the problem to be solved.*
- ***Individual*** $\mathbf{d}$*: An element of the solution space* $\mathbb{X}$*. The set of all possible individuals is denoted as* $\mathcal{I}$*. The elements of* $\mathcal{I}$ *are discrete sampling points of* $\mathbb{X}$*. Each individual* $\mathbf{d} \in \mathcal{I}$ *contains a gene sequence* $g_1, ..., g_n$*, representing a solution of the problem.*



(a) Crossover (b) Mutation

Figure 4.7: Genetic Algorithms - Genetic Operators (modified [49])

- ***Gene*** g_i*: A gene* $g_i \in \{g_1, ..., g_n\}$ *represents a certain variable parameter of the problem to be solved.*
- ***Population*** $\mathcal{I}_x$*: Set of considered individuals* $\mathcal{I}_x \subseteq \mathcal{I}$*. The population size* $|\mathcal{I}_x|$ *is denoted by* μ*.* x *identifies an iteration of the evolution, starting at zero.*

A genetic algorithm comprises the following functions:

Definition 2 *(Genetic Algorithm Functions)*

- ***Fitness Function*** f*: Based on the knowledge about the problem, a fitness function* $f : \mathcal{I} \rightarrow \mathbb{R}$ *is defined.* f *enables to evaluate the quality of each individual* $\mathbf{d} \in \mathcal{I}$*. The fitness value of a certain individual* $\mathbf{d} \in \mathcal{I}$ *is described by* $f(\mathbf{d})$*.*
- ***Crossover Function*** $\mathbf{f_c}$*:* $\mathbf{f_c} : \mathcal{I} \times \mathcal{I} \rightarrow \mathcal{I}$ *exchanges a part of the genes between two individuals* $\mathbf{d_x}, \mathbf{d_y} \in \mathcal{I}_x$*. An exemplary function is depicted in Figure 4.7(a). For the one point crossover depicted in that figure, one gene* $g_j \in G, j \leq n$ *is chosen (randomly) and then genes* g_j *to* g_n *are exchanged between* $\mathbf{d_x}, \mathbf{d_y}$*.* f_c *is applied to two individuals with a certain probability* $p_c \in [0, 1]$*.* p_c *is defined based on the optimization problem [81].*
- ***Mutation Function*** $\mathbf{f_m}$*: A randomized mutation of genes of an individual* $\mathbf{d} \in \mathcal{I}_x$ *is defined with* $\mathbf{f_m} : \mathcal{I} \rightarrow \mathcal{I}$*. An exemplary function is depicted in Figure 4.7(b). For the one point mutation depicted in that figure, genes* $g_j \in G$ *are mutated randomly with a probability* $p_m \in [0, 1]$*.* p_m *is defined based on the optimization problem [81].*
- ***Selection Function*** $\mathbf{f_s}$*: The selection function* $\mathbf{f_s} : \mathcal{I}_1 \subseteq \mathcal{I} \rightarrow \mathcal{I}_2 \subseteq \mathcal{I}$ *selects certain individuals to take part in the crossover and mutation process.*

The workflow of the genetic algorithm (as listed in Algorithm 1) starts with the generation of an initial population $\mathcal{I}_0$ with a function *FillWithRandomInvidiuals*. The population size is μ. For the individuals in that population $\mathbf{d} \in \mathcal{I}_0$, the fitness values are evaluated with the function *AssignFitness*. Based on these fitness values, a selection of the most appropriate candidates for the creation of the next generation $\mathcal{I}_{sel}$ is done. A subset of the elements of the $\mathcal{I}_{sel}$ takes part in the two evolutionary processes: mutation and crossover. The evolutionary processes create

Algorithm 1 Genetic Algorithms - Workflow

```
$\mu$ user defined
$x \leftarrow 0$
FillWithRandomInvidiuals($\mathcal{I}_x$,$\mu$)
**repeat**
    AssignFitness($\mathcal{I}_x$)
    $\mathcal{I}_{sel} \leftarrow$ Selected($\mathcal{I}_x$)
    $x \leftarrow x + 1$
    $\mathcal{I}_x \leftarrow$ PerformGeneticOperators($\mathcal{I}_{sel}$)
**until** *convergence criterion is reached or $x > Threshold$*
**return** Best($\mathcal{I}_x$)
```

new individuals. When population size μ is reached for population $\mathcal{I}_{x+1}$, the process then restarts with this population. The process is repeated until the population converges in terms of a convergence criterion or if a certain threshold of iterations is reached.

4.2.2 Elitism-based Multi-objective Genetic Algorithms

In areas where optimization is needed or genetic algorithms are utilized, the optimization often takes multiple objectives into account. Especially with embedded system design, the designers and users would like to have devices which cost little and have unlimited battery capacity. This demands for multi-objective optimization heuristics as MOGA (Multi-Objective Genetic Algorithms). The challenging task for MOGAs is the construction of the fitness function because of the multi-dimensional solution space. The simplest way to create that fitness function, is a projection into a one-dimensional solution space, e.g. by taking the average over the values for each objective. A more elaborated way are fitness functions which are based on the utilization of the Pareto dominance relation which is outlined in Section 4.2.2.1. A special group of these algorithms are elitism-based methods which will be described in Section 4.2.2.2. The latter algorithm class is based on the utilization of an individual archive containing best-performing individuals.

4.2.2.1 Pareto Optimization

Instead of one value representing the fitness of an individual **d**, in multi-objective GAs a objective value vector $\mathbf{v} = (v_1, .., v_m) \in \mathbb{R}^m$ is assigned to each individual by a function $\mathbf{f_p}$:

$$\mathbf{f_p} : \mathcal{I} \to \mathbb{R}^m, \tag{4.1}$$

where m is the number of different objectives. In order to evaluate this vector v with regard to the vectors of other individuals, a relation called Pareto dominance was developed [145] to evaluate the fitness.

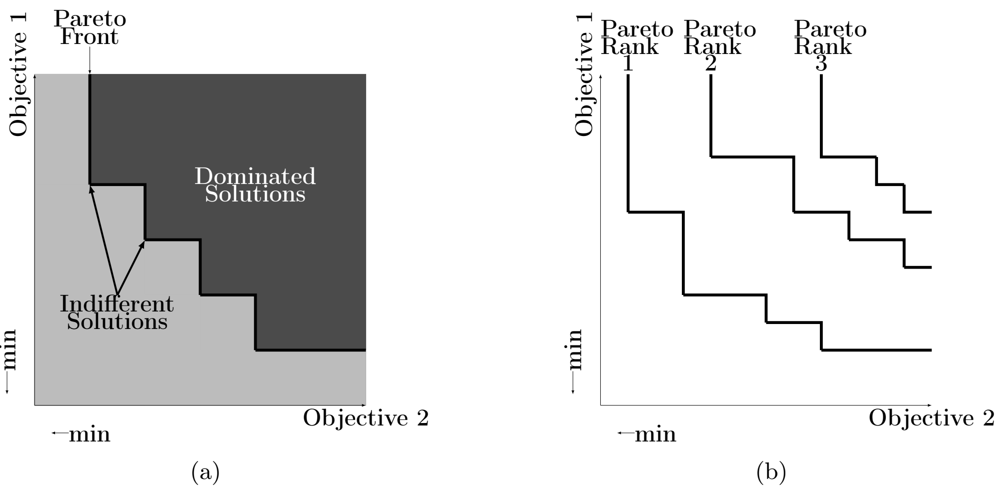


Figure 4.8: (a) Design Space with Pareto Front (modified [84]) and (b) Pareto Ranks (modified [43])

Definition 3 *(Pareto Dominance)*
Let $\mathbf{d_x}, \mathbf{d_y} \in \mathcal{I}$ *be two individuals and* $\mathbf{v} = \mathbf{f_p}(\mathbf{d_x})$ *and* $\mathbf{u} = \mathbf{f_p}(\mathbf{d_y})$,$\mathbf{v}, \mathbf{u} \in \mathbb{R}^m$. $\mathbf{d_x} \succ \mathbf{d_y}$ *is the Pareto dominance function and means* $\mathbf{d_x}$ *dominates* $\mathbf{d_y}$ *(or* $\mathbf{d_y}$ *is dominated by* $\mathbf{d_x}$*), if the condition*

$$\bigwedge_{i \in 1,..,n} (v_i \leq u_i) \wedge \bigvee_{i \in 1,..,n} (v_i < u_i) \tag{4.2}$$

is true. If neither $\mathbf{d_x}$ *dominates* $\mathbf{d_y}$ *nor* $\mathbf{d_y}$ *dominates* $\mathbf{d_x}$*, the two individuals are indifferent.*

Definition 4 *(Pareto Optimality)*
An individual $\mathbf{d_x} \in \mathcal{I}$ *is called a non-dominated solution, if*

$$\neg \exists \mathbf{d_y} \in \mathcal{I} : \mathbf{d_y} \succ \mathbf{d_x}, \tag{4.3}$$

i.e $\mathbf{d_x}$ *is not dominated by any other individual* $\mathbf{d_y} \in \mathcal{I}$.

As can be seen from Figure 4.8(a), the solution space is built upon the Pareto dominance as follows. The individuals which are not dominated by any other solution are called Pareto front. Solutions on the Pareto front are indifferent solutions.

4.2.2.2 Examples

In this section, two exemplary elitism-based multi-objective algorithms are described. The critical point while optimizing based on the Pareto dominance function is the selection of the individuals for the next generation and the creating of an efficient multi-objective fitness function. In order to avoid withdrawing *good* solutions, elitism-based MOGAs use an archive containing the solutions on the Pareto front.

SPEA2: SPEA2 [158] is a multi-objective algorithm providing a fitness function based on the Pareto dominance relation. Additionally, it provides an efficient selection mechanism for elitism-based populations. SPEA2 assigns the fitness $f(\mathbf{d_x}) = R(\mathbf{d_x}) + D(\mathbf{d_x})$ to an individual $\mathbf{d_x}$.

$$R(\mathbf{d_x}) = \sum_{\mathbf{d_y} \in \mathcal{I}_x, \mathbf{d_y} \succ \mathbf{d_x}} S(\mathbf{d_y}) \tag{4.4}$$

is the raw fitness of an individual I which takes into account the strength

$$S(\mathbf{d_y}) = |\{\mathbf{d_z} | \mathbf{d_z} \in \mathcal{I}_x, \mathbf{d_y} \succ \mathbf{d_z}\}| \tag{4.5}$$

of the individuals $\mathbf{d_y}$ which dominate $\mathbf{d_x}$. In addition to that, SPEA2 also provides a *correction function* $D(\mathbf{d_x})$ to take density information into account. For this, the Euclidean distance from one individual to all other individuals is calculated and then the inverse of the distance to k neighbours is used. This has the effect that individuals nearby get a higher fitness value and are probably not selected for the next population, because these individuals have often similar characteristics. Furthermore, SPEA2 provides a multi-objective tailored selection process, utilizing natural selection, meaning that the fitness individuals will for group the next generation. All non-dominating individuals are copied to the aforementioned archive.

NSGA-II: Another multi-objective genetic algorithm based on the Pareto dominance relation is NSGA-II [43]. It is not directly using a direct fitness function like SPEA2 but is based on non-dominated sorting. The sorting mechanism includes the definition of Pareto ranks. As can be seen in Figure 4.8(b), the solutions can be divided into different Pareto fronts called Pareto ranks $(F_1, F_2, ...)$. F_1 is the original Pareto front and $F_i, i > 1$ are sets of indifferent solutions behind F_1 (as illustrated in Figure 4.8(b)). NSGA-II uses these Pareto-ranks to select the individuals for the next generation. The population for the next generation is filled with Pareto-ranks until the populations size μ is reached or the last selected Pareto rank does not fit into the population. In order to fill the population with individuals from the last selected Pareto rank, a diversity operator is utilized which selects individuals that are equally distributed in the Pareto rank in terms of distance.

Chapter 5

Multi-objective *Hardware/Software Codesign* for GPGPU Applications

In this chapter, the GPGPU application design process will be targeted and methods to optimize it will be presented. In addition to that, primarily suggestions are made for utilizing many-core chip architectures in the field of embedded systems. In Chapters 2, 3 and 4 the basics for the code optimizations, the design space explorations and the application scenarios were provided. Especially, the energy and runtime testbed presented in Chapter 3 is utilized in this chapter for profiling. Furthermore, compiler basics and the fundamental concepts of multi-objective optimization are essential requirements for understanding this chapter. This chapter is then followed by the communication-related chapters of this thesis.

Contents

5.1 Introduction

Nowadays, industrial and scientific GPGPU applications are designed with the help of BSP (Board Support Packages) provided by the graphics chip vendors such as Nvidia with its CUDA architecture [97]. These board support packages have been released to make the programming of graphics chips as easy as possible and provide also the possibility to evaluate the performance of GPGPU applications. GPGPU programming with these board support packages has some disadvantages when utilized in special purpose systems such as embedded systems. These disadvantages will be summarized in the following.

One of the major disadvantages is that this type of GPGPU programming is dominated by manually performing code optimizations. The mapping optimization of computation kernels to graphics card cores is also done manually in order to achieve the optimal acceleration of an application. This is time-consuming and error-prone. In addition, the situation is aggravated by the fact that some parameters in the workflow of GPGPU programming, such as the number of parallel allocatable threads on each core, are static for one graphics chip generation but change from one graphics chip generation to the next. This leads to longer design times when re-using software components and libraries, since these application parts must be manually optimized again. When using many-core systems such as GPUs in embedded systems and especially in mobile systems, another disadvantage can be noticed. The energy consumption of a GPGPU application is not taken into account and the most suitable platform variant of a graphics chip family cannot be determined. These are challenges suggested to be addressed by the HIPEAC roadmap [21] and should be taken into account. Neglecting the energy consumption and choosing the wrong platform can lead to the situation that a far too powerful chip is utilized which consumes too much energy. This can have a negative impact on the usability of battery-driven systems such as mobile systems but also on large scale systems that have requirements towards an energy efficient design. For large scale systems, this is especially important when facing global warming and the resulting need for green computing and, of course, energy costs. Besides energy efficiency, the choice of the graphics cards variant is also crucial if a GPGPU application is part of an embedded or cyber-physical system in a real-time scenario. Then, the optimal GPU variant must be found to cope with real-time deadlines. Such a strategy is not natively supported by the board support packages. The last disadvantage of up-to-date board support packages is the way of programming, because the code is written in a single-thread fashion without the possibility to express parallelism in a sophisticated way.

In the following it is presented, how the disadvantages of up-to-date board support packages can be counteracted by the methods presented in this thesis:

1. The manual optimization process is substituted by an automated feedback-based optimization which provides energy consumption and performance values.

2. Static parameters of the board support packages are taken into account within the automatic optimization.

3. Energy consumption and runtime performance are the objectives of the optimization.

4. The platform selection itself is taken into account in a design space exploration.

This chapter is structured as follows: First of all, related work is presented in Section 5.2. Two GPGPU compiler optimizations exploiting instruction scheduling to optimize GPGPU applications towards energy conservation and runtime decrease will be introduced in Sections 5.3 and 5.4. In Section 5.5, a design space exploration for an exemplary GPGPU application is presented towards the same objectives but with the platform variant selection in mind. Afterwards, in Section 5.6 it will be shown that integrating a graphics card for application acceleration does not need to be counter-productive with respect to energy efficiency. This chapter ends with a conclusion for GPGPU application optimization.

5.1.1 Optimization Potential in Classical GPGPU Application Design Process

The process of how GPGPU applications are specified, mapped and executed on a GPU with the board support packages of Nvidia, called CUDA is depicted in

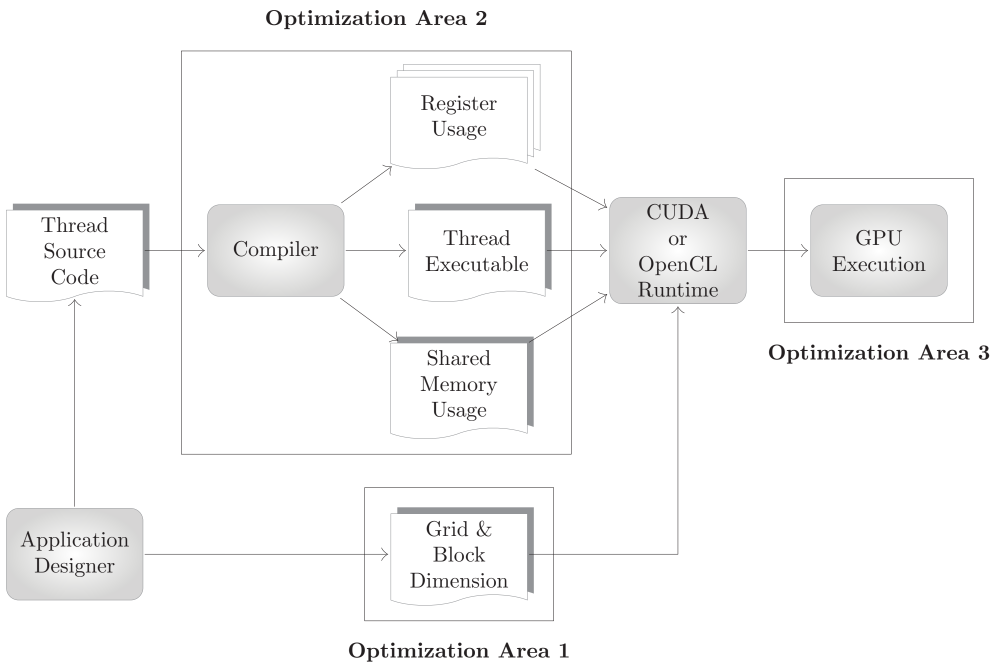


Figure 5.1: Overview on Variable Parameters in GPGPU Application Mapping and Execution Process

Figure 5.1. In this process several parameters influence the performance and the energy consumption of a GPGPU application. This section reveals those parameters which are considered by the optimizations and design space explorations of the next sections. A detailed description of the GPGPU application design process was already presented in Chapter 2.

The process starts with the specification of the application in form of *CUDA C* (or *OpenCL*) source code and the specification of the mapping/allocation parameters (*CUDA*: *grid* and *thread block* size, *OpenCL*: NDRange and *work-group* size) by the GPGPU application designer. These mapping/allocation parameters (*Optimization Area 1*) have a direct impact on the allocation of the *work-groups*, respectively the *thread blocks* onto the GPU, and therefore also on the performance and the energy consumption.

The mapping/allocation parameters are not the only parameters that have an influence on the execution time of a GPGPU application. The other parameters which are variable in the mapping and execution process of GPGPU applications are called compiler-related parameters (*Optimization Area 2*). The set of compiler-related parameters comprises the maximal number of registers that are used by a single thread of the application and the amount of shared memory used per thread. These parameters are used by the *thread block* scheduler to allocate the *thread blocks* respectively the *work-groups* on the different streaming multi-processor. The latter can result in a performance and/or in an energy consumption decrease/increase. In addition to that *Optimization Area 2* also comprises GPGPU machine code optimizations.

When talking about energy efficiency, an additional parameter is the execution platform itself, e.g. a too powerful platform is selected, it consumes too much energy. Another case is that the platform is efficient towards the consumption of energy but not powerful enough to meet real-time deadlines. Therefore, the platform decision is *Optimization Area 3*.

5.1.2 *Hardware/Software Codesign* for GPGPU Applications

Due to the use of many-core chips in the embedded system world [157], embedded system design and high performance computing design techniques are merging. Especially, *HW/SW Codesign* (HardWare/SoftWare Codesign) is one technique which can be used in both areas. A special case of *HW/SW Codesign* is the platform-based design. Platform-based design has the objective to efficiently map an application or a set of applications to an execution platform (family) such as GPUs. According to [84] and [128], this mapping problem is defined – in a simplified way – as follows:

Given:

1. Application or set of applications
2. Use case descriptions of the application(s)
3. A set of possible target platforms, including processor type, communication methods or the scheduling policy.

Find:

1. The most efficient platform (combination)
2. An efficient mapping of the application to the platform(s), including scheduling

Objectives:

1. Functional objectives such as *number of met hard deadlines*
2. Non-functional objectives such as available QoS, produced costs or energy consumption

Constraints:

1. Functional constraints, e.g. *perform task in 100*ms
2. Non-functional constraints, e.g. *perform task and consume 100*J

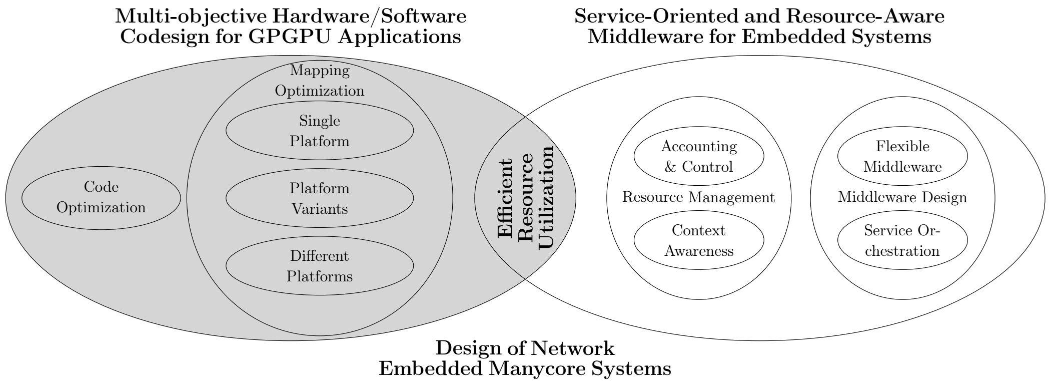

Multi-objective Hardware/Software Codesign for GPGPU Applications
Service-Oriented and Resource-Aware Middleware for Embedded Systems
Mapping Optimization
Single Platform
Platform Variants
Different Platforms
Code Optimization
Efficient Resource Utilization
Accounting & Control
Resource Management
Context Awareness
Flexible Middleware
Middleware Design
Service Orchestration
Design of Network Embedded Manycore Systems

Figure 5.2: Overview on the Overall Design Process

The code optimization and design space exploration techniques to be presented in the chapter adapt the mapping problem and are targeted towards the *Multi-objective Hardware/Software Codesign for GPGPU Applications* as described in Figure 5.2. Especially the combined consideration of mapping optimization and code optimization is important with respect to an efficient utilization in GPGPU application design. Mapping and code optimizations are also major challenges of the HIPEAC roadmap [21]. Therefore, the structure of this chapter follows a bottom-up approach towards a *Multi-objective Hardware/Software Codesign for GPGPU Applications*, meaning that at first, code optimization techniques will be introduced and then the mapping optimizations. The first mapping optimization introduces an optimization for a single platform (single GPU). Then several variants of a platform (GPU variant) are taken into account and finally, different hardware platforms (GPU or CPU) are considered. In the scope of this thesis, it is assumed that the parallel application is available as source code for the different target architectures: CPU and GPU. In addition to that, the code has to be partitioned into sequential sections for the host system and into sections which can be executed in parallel on an accelerator (ACS (Accelerator Code Sections)) connected to the host system. ACSs have to be parallelized in an adequate way, e.g. by an efficient manual parallelization or an automatic parallelization (Loopo: [60], Pluto: [20]) and they can be executed on platforms such as GPU or FPGA. The optimization and mapping of ACSs are considered in this chapter of the thesis and the following targets are focussed on:

1. Where is an ACCs executed efficiently in terms of energy consumption and runtime?

2. How can an ACS be optimized in terms of energy consumption and runtime?

3. How can energy consumption and runtime be efficiently evaluated?

These three targets are addressed with *Multi-objective Hardware/Software Codesign for GPGPU Applications*, presented in Figure 5.2. in Sections 5.3.1 and 5.4.1, code optimization and mapping optimization techniques (*Optimization Area 2*) will be presented. The code optimization techniques are based on the utilization of instruction scheduling but differ in the scope where instructions can be placed which are optimized. Due to the automatic mapping capabilities of modern GPUs it is not possible to decide on a block basis where single blocks are mapped to. In addition to that, the mapping of blocks is also influenced by the maximal register usage and therefore, both code optimization techniques cover the mapping and code optimization area. Both code optimizations pursue a profiling-based approach meaning that performance and energy values of reference platforms are determined and considered within the optimization. This has the advantage that the accuracy of the performance and energy values are much more precise in comparison to a model-based approach [82].

In Sections 5.5 and 5.6, design space explorations in the field of GPGPU applications are described. While in Section 5.5 an GPGPU application towards the most energy efficient GPU platform *(Optimization Area 3)* and mapping/allocation parameters (*Optimization Area 1*) is evaluated, in Section 5.6, the decision whether an integration of a GPU for accelerating a parallel application is energy efficient or not *(Optimization Area 3)* is focussed on. Both design space explorations are again supported by the energy consumption and performance testbed presented in Chapter 3.

5.2 Related Work

The related work section in this chapter comprises three different areas which are important for the consideration of how the work in this thesis is classified towards other work in the context. The first area presented in Section 5.2.1 comprises the methods in the context of energy-aware high performance computing and especially energy-aware GPGPU computing. With respect to the design space explorations and code optimizations in this chapter, related compiler optimization approaches will be presented in Section 5.2.2 whereas in Section 5.2.3 related work in the scope of system-level design space exploration will be discussed.

5.2.1 Energy-Aware and Embedded High Performance Computing

In this section, related work with respect to energy-aware embedded high performance computing will be presented which are related to optimization techniques of the *Multi-objective Hardware/Software Codesign for GPGPU Applications* to be presented in this chapter.

OpenCL [61], a standard for programming multi-core systems, provides high portability capabilities and the ability to run GPGPU applications even on small embedded devices [109, 157]. The authors of [109] used it for an image processing and transformation pipeline which could be executed on the CPU of a mobile phone and on the GPU of the same phone. The authors showed that the execution on the GPU is more energy efficient. Compared to HPC, the optimal platform in embedded system design – the GPU in this case – can be chosen at design time, because these systems are tailored towards a special purpose, i.e. the application or application domain is fixed at design time. GPGPU is widely utilized in the high performance computing community for scientific and industrial applications (see e.g. [30, 93]) and an increasing number of graphics chip vendors is able to integrate powerful graphics chips in smaller systems. Previous papers from the HPC community often have the objective to accelerate an application as much as possible. This is also the main objective when designing a real-time system because the system has to be accelerated in a way that the deadline is met. Several authors have taken the energy consumption of GPGPU applications into account [36, 114, 154] – not from a compiler perspective, but to answer the question how and when the integration of a graphics card/chip for GPGPU is energy efficient. Especially the authors of [34, 155] examined the potential of integrating graphics card/chip for GPGPU in terms of energy savings. The authors of [155], for instance, evaluated the influence of kernel fusion on the energy consumption. The authors of [34] evaluated the energy efficiency of the analysis of neural signals on a GPGPU cluster. In [110] the multiplication of large matrices was optimized with regard to energy consumption but without extending a compiler.

5.2.2 Compiler Optimizations

The code optimizations to be presented in this chapter are based on compiler optimizations. Therefore, relevant related work will be presented in this section.

In [143], the authors showed that in traditional single-core environments, instruction scheduling can have a negative effect on energy consumption. The authors of [72] developed an algorithm called balanced scheduling which performs scheduling of the instructions based on an availability on instruction level parallelism. However, energy consumption was not part of that work and the runtime environment was not taken into account as it has to be done for a GPU with its hardware thread scheduler. For special purpose systems such as DSPs, several approaches with instruction scheduling methods [78, 146] exist to produce optimized applications with respect to performance [78] and energy consumption [146]. Both works only targeted single-core and single-threaded code. The first work which used local instruction scheduling for optimizing GPGPU applications was presented in [46]. In that work, a performance degradation was possible, because the optimizations had no information about the later stages of the compilation process and the resulting influence on the runtime performance. Theoretical work which evaluates the register utilization of GPGPU applications was presented in [95]. The authors presented a framework for displaying the machine code of GPGPU applications which helps to identify register live ranges and to visualize the gain of optimizations. This information enables a manual optimization of a GPGPU application in a trial-and-error based fashion. Again, the energy consumption of a GPGPU application was not considered.

5.2.3 System-level Design Space Exploration

The mapping optimizations to be presented in this chapter are based on design space exploration techniques. Therefore, relevant related work will be presented in this section.

Especially in the design process in the field of embedded systems, a design space exploration can be performed at design time to obtain an optimized system configuration. In this domain, many different approaches [15, 71, 94, 158] exist which provide the capability to automatically explore the design space – often with genetic algorithms. The different design space approaches are utilized for mapping communication channels and tasks of an application (or a set of applications) on buses and processing units. Except that they are working on a completely distinct domain, i.e. MPSoC (MultiProcessor System on Chip) system design, the multi-objectively design space approaches could possibly be adopted and tailored towards designing GPGPU applications. Especially [15], where authors map applications to the IBM CELL BE [35], is important by introducing efficient techniques for scheduling and allocating of the processing jobs. The major difference to mapping optimizations to be presented is the fact that energy-efficiency was not an objective and the IBM CELL BE allows a more fine grained mapping. That is not suitable for a utilization on graphics cards.

5.3 Multi-objective Local Instruction Scheduling

The first code optimizations for GPGPU code will be introduced in this chapter. The code optimization technique is called *MOBLIS* (Multi-OBjective Local Instruction Scheduling). After a short motivation in Section 5.3.1, an optimization technique based on local instruction scheduling and the utilization of state-of-the-art multi-objective genetic algorithms is presented in Section 5.3.2. The results of applying *MOBLIS* to several benchmarks are provided in Section 5.3.3. The section ends with a conclusion in Section 5.3.4.

5.3.1 Introduction

The emergence of international research projects about optimizing compilers for GPGPUs such as Ocelot [45] followed the need for optimizing GPGPU applications beyond the code quality achieved by the board support package of modern graphics chips. Especially the time-consuming manual GPGPU programming process must be extended by an automatic evaluation of the performed code changes.

Although projects and initiatives exist and several source code optimizations have

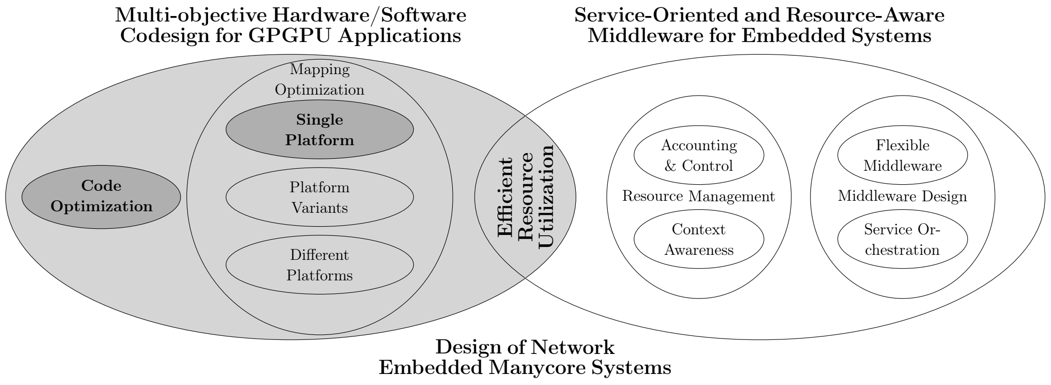

Multi-objective Hardware/Software Codesign for GPGPU Applications
Service-Oriented and Resource-Aware Middleware for Embedded Systems
Mapping Optimization
Single Platform
Platform Variants
Different Platforms
Code Optimization
Efficient Resource Utilization
Accounting & Control
Resource Management
Context Awareness
Flexible Middleware
Middleware Design
Service Or-chestration
Design of Network Embedded Manycore Systems

Figure 5.3: Overview on the Overall Design Process

been carried out (see Section 5.2) aiming at energy consumption efficiency of graphics cards and the corresponding programming, only little effort has been spent on accounting the power and energy consumption efficiency in the compilation process of a GPGPU application.
The optimization area which is considered for this optimization is *Optimization Area 2* (shown in Figure 5.1) which means that the machine code is optimized with the help of an optimizing compiler. In the scope of the overall design process depicted in Figure 5.3, this section provides a code optimization technique and will provide mapping optimization methods for a single GPU platform. In addition to that, the following important requirements should be handled in *MOBLIS*:

1. *Optimal Parallel Register Utilization:* A Nvidia GPU utilizes a shared register file for an efficient register access. Depending on the hardware generation, the number of registers ranges from 8192 up to 32768 which are shared by all allocated threads on a processor (see Section 4.1.3). Therefore, optimization potential is available by changing the number of registers per thread which leads to more threads and blocks that can be allocated to a streaming multiprocessor and to a performance increase. E.g. when 2 blocks – with 128 threads per block and 32 registers per thread – are allocated to a streaming multiprocessor, 8192 registers are required in total. A possible register reduction to 21 registers allows the allocation of 3 blocks to a streaming multiprocessor and a possible performance boost (and energy consumption reduction due to runtime reduction). This is possible if the kernel performance is not limited by the memory bandwidth [68]. Furthermore, the mapping optimization should also be evaluated if an increase/decrease in the parallel register utilization leads to a performance decrease.

2. *Pipeline Load Optimizations:* Warp scheduling and scoreboarding (see Section 4.1.3 for both) allow a graphics card to process instructions of other warps while some warps wait for data from the global memory. This is done

in order to avoid pipeline stalls and to hide the latency of un-cached main memory accesses on Nvidia GPUs [98]. A code optimization has to change the instruction scheduling of each thread and to optimize the load distribution between the instruction and memory pipeline in order to decrease the energy consumption and the runtime of a GPGPU application.

3. *Black Box Optimization:* Due to the lack of a detailed architectural or functional description, not all features of the graphics cards are publicly known. Therefore, optimization techniques have to work in the face of lacking a concrete model of the underlying hardware features. In addition to that, some of the features such as the warp scheduling are too complex to simulate for large GPGPU applications. Another area which is not under control of an application designer is the actual machine code generation, the register allocation and some low-level optimizations. Therefore, optimization on the available intermediate representation need not necessarily lead to optimizations at all.

4. *Multi-objective Optimization:* When optimizing towards performance, it is often the question whether this is also an optimization towards energy efficiency. As already shown in Section 3.1, a reduction in runtime need not necessarily lead to a reduction of the energy consumption.

5. *Platform-aware Optimization:* In comparison to optimizing compilers, the hardware characteristics of the actual platform variant are not considered explicitly in most compilers. Especially the clock speed of the memory and the processors are crucial parameters because a change in these parameters will result in a different scheduling on the GPU and that will influences the optimization of the GPGPU application. In addition to that, GPGPU programming parameters such as number of concurrently running threads or the number of concurrently allocatable blocks to a streaming multiprocessor should be taken into account.

6. *Avoiding Unfavourable Solutions:* Many compilers provide optimizations which are targeted at increasing the average-case performance. This means that for some optimizations and platforms, a possible performance degradation and a possible increase of the energy consumption can be the result. This should be avoided in *MOBLIS*. In [59], it was shown that applying a single local scheduling technique is not sufficient to optimize all benchmarks. Therefore, an adaptive approach should be applied for optimization.

5.3.2 *MOBLIS* - Materials and Methods

The optimization process of *MOBLIS* is depicted in Figure 5.4. It takes the six requirements listed in Section 5.3.1 into account. Up to now, no other code optimization techniques of GPGPU application presented in literature has been aware

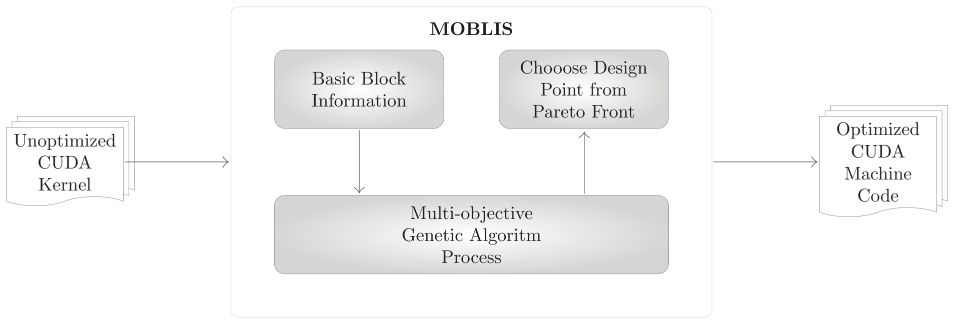
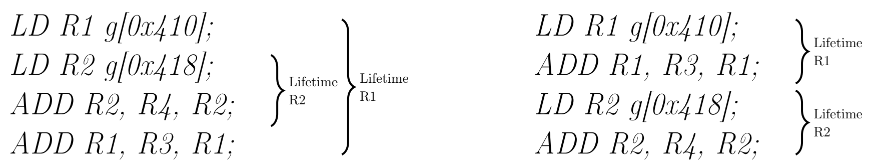

MOBLIS
Unoptimized CUDA Kernel
Basic Block Information
Chooose Design Point from Pareto Front
Multi-objective Genetic Algoritm Process
Optimized CUDA Machine Code

Figure 5.4: Multi-objective Local Instruction Scheduling (*MOBLIS*) Framework

of all of these requirements. The *MOBLIS* optimization process starts with reading in GPGPU code in C for CUDA and transforming it into a WHIRL intermediate representation (see Section 4.1.3.2). There are no dependencies between the kernels because each kernel runs a new application on the graphics cards and therefore, optimizations can be done separately for each kernel. The output of the *MOBLIS* process is optimized machine code for a GPGPU kernel. All optimized kernels are then combined into one complete optimized GPGPU application.

The optimization technique which is exploited in this section is IS (Instruction Scheduling). It is performed on the WHIRL intermediate representation. Instruction scheduling techniques can be divided in two categories: local scheduling and global scheduling. With local scheduling techniques, instructions are only scheduled within a single basic block. Global scheduling is more powerful because it allows the movement of instructions from one basic block to another in addition to scheduling inside a basic block. Instruction scheduling can cope with the first requirement which is the optimization of the register file utilization. In Figure 5.5(a), an exemplary unoptimized IR code sequence is depicted. The first two instructions load data from global memory to registers $R1$ and $R2$ (in that order). After these two instructions, firstly $R2$ is accessed and then $R1$. This leads to the situation that the value in $R1$ must be available concurrently to the value in $R2$. In Figure 5.5(b), an optimized version of the same code section is presented. The lifetimes of the register values are not overlapping any longer and therefore, more threads can possibly be mapped to the same streaming multiprocessor. The detailed mapping process was described in Section 4.1. The second requirement, the optimized pipeline load, can also be achieved by instruction scheduling, e.g. by changing the position of memory and non-memory instructions.

The last three requirements cannot be considered with the help of plain local instruction scheduling. Therefore, the local IS presented in this section was extended to consider global information about energy consumption and runtime. The energy consumption and runtime values, which are collected with profiling, are used by a



```
LD R1 g[0x410];
LD R2 g[0x418];
ADD R2, R4, R2;
ADD R1, R3, R1;
```

(a) Unoptimized Sequence

```
LD R1 g[0x410];
ADD R1, R3, R1;
LD R2 g[0x418];
ADD R2, R4, R2;
```

(b) Optimized Sequence

Figure 5.5: Example for Register Usage Reduction

genetic algorithm to find an optimal instruction schedule for a GPGPU kernel. Genetic algorithms are well suited for two different optimization scenarios. Firstly, they can be used as a heuristic in order to get improved run-times for an optimization problem. Secondly, they can be tailored to a use in environments, where no closed form of the objective function is known but the function can be evaluated [81], e.g. the energy consumption of an application can be evaluated, but the contribution of the different functional units to the overall energy consumption is unknown. Both are important in the context of this section because the search space is quite large and the *real* objective function for energy consumption and runtime is unknown but evaluable. By employing a GA evaluating profiling values, MOBLIS is able to cope with the requirements *Black Box Optimization*, *Platform-aware Optimization* and *Avoid Unfavourable Solutions*. The requirement *Multi-objective Optimization* is covered by the use of MOGA (Multi-Objective Genetic Algorithms) which were introduced in Section 4.2.2.2. The MOGAs used in MOBLIS are NSGA-II and SPEA2 which are population and elitism based algorithms approximating a Pareto-front for the different objectives meaning that *"the distance to the optimal front is to be minimized and the diversity of the generated solutions is to be maximized"* [158].

The optimization process inside the *MOBLIS* framework, as depicted in Figure 5.4, comprises three steps. Firstly, the number of basic blocks (Definition 5) and the sequence of the basic blocks are extracted from the WHIRL intermediate representation of a GPGPU kernel. The basic block information is then used in the multi-objective genetic algorithm process to create the entities of the GA as will be described in Section 5.3.2.2. The last step within the MOBLIS framework is the choice of generated CUDA Kernel machine code as the final design. This can be a CUDA kernel optimized for energy consumption and/or runtime performance.

5.3.2.1 Genetic Algorithm Specification

The instruction scheduling policy is applied per basic block. In the following, utilized terms in the scope of *MOBLIS* are presented:

- n: Number of all basic blocks in the WHRIL representation of one kernel.
- B: The set of all basic blocks is denoted by $B = \{b_i | 1 \leq i \leq n\}$.

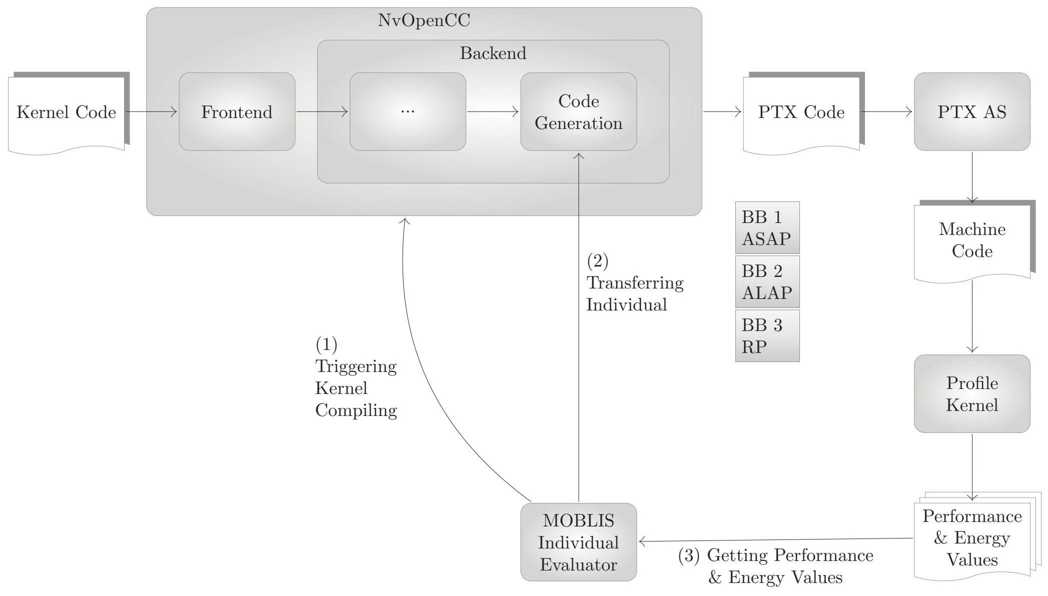

NvOpenCC
Backend
Kernel Code
Frontend
...
Code Generation
PTX Code
PTX AS
BB 1 ASAP
BB 2 ALAP
BB 3 RP
Machine Code
(2) Transferring Individual
(1) Triggering Kernel Compiling
Profile Kernel
Performance & Energy Values
MOBLIS Individual Evaluator
(3) Getting Performance & Energy Values

Figure 5.6: *MOBLIS* -Individual Evaluation Flow

- **Gene** $g_i \in G = \{g_i | 1 \leq i \leq n\}$: A gene g_i represents a scheduling policy applied to a basic block $b_i \in B$.
- **Individual d**: In the compilation process an individual $\mathbf{d}$ is an instance of a GPGPU kernel. Each individual $\mathbf{d} \in \mathcal{I}$ comprises a gene sequence $g_1, ..., g_n$. Within this gene sequence for each g_i, a value is assigned which represents a scheduling policy. Each scheduling policy is represented by a natural number as follows:

  1. Schedule unchanged
  2. ASAP schedule
  3. ALAP schedule
  4. RP [46] list scheduler

5.3.2.2 Optimization Workflow

In Figure 5.6 the *MOBLIS* individual evaluation flow is depicted. For local instruction scheduling *NVOpenCC* from Nvidia (Open64 [80]) was enhanced. *NVOpenCC's* code generation module was extended to communicate with *MOBLIS*. The *MOBLIS* individual evaluation flow starts with the generation of an individual **d**. Afterwards, **d** is utilized in the *NVOpenCC* to apply a scheduling policy to each basic block.

For evaluating the energy consumption and the runtime of the created individuals, the testbed presented in Section 3.2 is employed. As described in that section, the testbed provides the energy consumption $energy(P_{\mathbf{d}}, P_l, t_{start}^{P_{\mathbf{d}},P_l}, t_{end}^{P_{\mathbf{d}},P_l})$, where $P_{\mathbf{d}}$ is the program (corresponding to one individual $\mathbf{d}$) to be profiled, P_l the execution platform, $t_{start}^{P_{\mathbf{d}},P_l}$ the profiling start point in time and $t_{end}^{P_{\mathbf{d}},P_l}$ the profiling end point in time. $t_{start}^{P_{\mathbf{d}},P_l}$ and $t_{end}^{P_{\mathbf{d}},P_l}$ denoted the start and the end of the program and therefore, it depends on $P_{\mathbf{d}}$ and P_l. They are delivered by the testbed. The runtime of a program P on a platform P_l is therefore $runtime(P_{\mathbf{d}}, P_l) = t_{end}^{P_{\mathbf{d}},P_l} - t_{start}^{P_{\mathbf{d}},P_l}$. The optimization gain of an individual is measured as the ratio to the individual $\mathbf{d}_{un}$ ($P_{\mathbf{d}_{UN}}$) where for each gene the allele a_1 is assigned. One solution space point (individual $\mathbf{d}$) comprises the two values for runtime $r_{ratio}^{P_{\mathbf{d}},P_l}$ and energy consumption $E_{ratio}(\mathbf{d})$ for a platform P_l:

$$r_{ratio}^{P_{\mathbf{d}},P_l} = \frac{runtime(P_{\mathbf{d}}, P_l)}{runtime(P_{un}, P_l)} \tag{5.1}$$

$$E_{ratio}^{P_{\mathbf{d}},P_l} = \frac{energy(P_{\mathbf{d}}, P_l, t_{start}^{P_{\mathbf{d}},P_l}, t_{end}^{P_{\mathbf{d}},P_l})}{energy(P_{un}, P_l, t_{start}^{P_{un},P_l}, t_{end}^{P_{un},P_l})} \tag{5.2}$$

The fitness values are then calculated with methods specified in SPEA2 and NSGA-II (see Section 4.2.2.2).

5.3.2.3 Evolution Operation

The crossover and the mutation operators for *MOBLIS* are single point operators. Thus, a certain gene position is randomly chosen as crossover point. From this position on, the succeeding genes are exchanged between the chromosomes as described in Section 4.2.1. For the mutation process, genes are chosen and mutated with a certain probability.

5.3.3 Evaluation

In this section, the evaluation of the optimization potential by applying *MOBLIS* is presented. Firstly, the configuration and parameters for the evaluation are given in Section 5.3.3.1. Results on one GPU are presented in Section 5.3.3.2, then the results for several GAs are compared. In Section 5.3.3.3, the results for the first GPU are compared to another GPU variant. Afterwards, in Section 5.3.3.5 an additional analysis on the parallel register usage is provided.

5.3.3.1 Parameters / Configuration / Optimization Runtime

The population size μ equals the number of genes (respectively number of basic blocks) in an individual of a certain benchmark to ensure that the solution space is sufficiently explored. The number of basic blocks are listed in Table 5.1. As can be seen in that table, the number of basic blocks per kernel is between 3 and

Benchmark	Kernel	Number of Basic Blocks
Eigenvalues	*bisectKernelLarge_ MultIntervals*	86
Matrix Multiplication	*matrixMul*	7
DirectX Texture Compressor	*compress*	47
Binomial Option Pricing	*binomialOptionsKernel*	22
Back Propagation	*bpnn_ adjust_ weights_ cuda*	3
Recursive Gaussian Filter	*d_ recursiveGaussian_ rgba*	13

Table 5.1: Number of Basic Blocks for Kernels optimized by *MOBLIS*

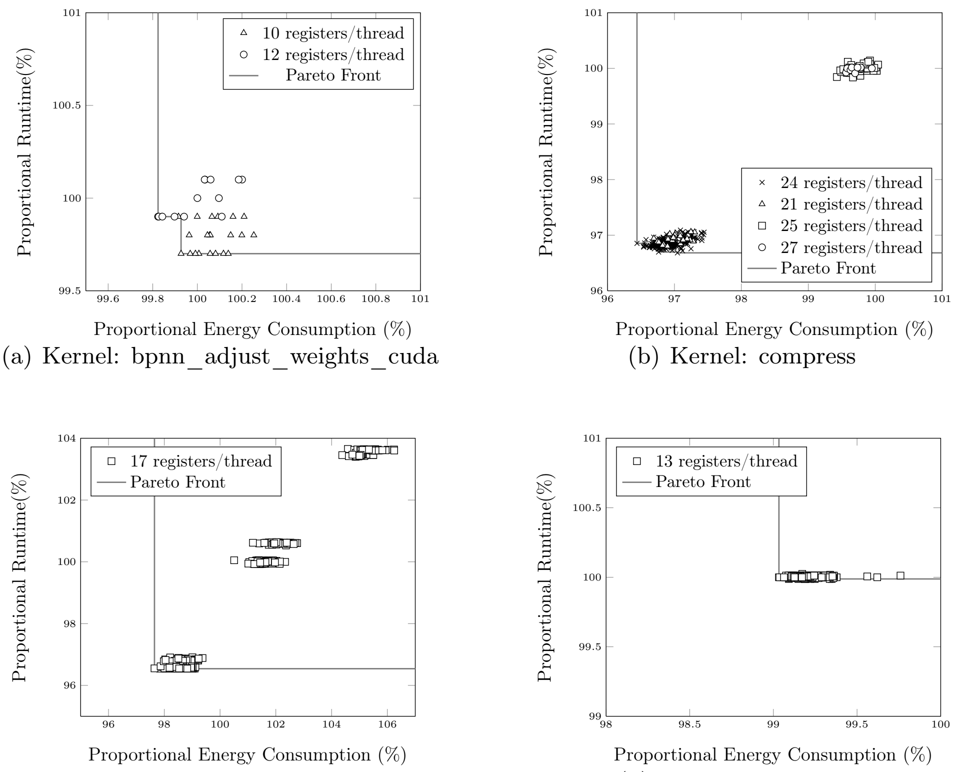
86. The genetic algorithms last 30 generations. The probability for mutating genes of a chromosome was $p_m = 0.80$ and the probability for crossover was $p_c = 0.20$ – according to the definitions in Section 4.2.1. The high mutating rate aims at exploring the solutions space in a sufficient way. Each individual is evaluated 10 times and the median was selected as average energy consumption and runtime result.

Graphics cards from Nvidia have the capability to use performance counters to estimate the performance of a kernel. In this evaluation, the performance counters were employed to verify the functionality of the optimizations and to take a deeper look to the performance indicators, such as the warp serialization rate or number of instructions needed. The results of section 5.3.3 were obtained with the presented energy consumption and runtime testbed. The graphics cards used in the evaluation comprise a 8400GS and a GTS250. The SPEA2 and NSGA-II implementations from the JECO library [113] were used in *MOBLIS*. For the Sections 5.3.3.2 to 5.3.3.4 it was not allowed to increase the parallel register utilization. This was only applied to the benchmark presented in Section 5.3.3.5.

The runtime of applying *MOBLIS* to a GPGPU kernel strongly depends on the runtime of the GPGPU kernel on a particular platform itself, because for each individual the benchmark is executed. Therefore, runtimes between several hours and multiple days were observed.

5.3.3.2 Single Platform Results

Figures 5.7(a) - 5.7(d) depict the energy consumption and runtime reductions which are achieved with *MOBLIS* for different benchmarks on a Nvidia 8400GS. The figures show the proportional energy consumption ($E_{ratio}^{P_d,P_l}$) and proportional runtime ($r_{ratio}^{P_d,P_l}$) reduction – as percentage of the unoptimized version – for all evaluated individuals. Furthermore, the Pareto front connects the Pareto-optimal points with a line. Figure 5.7(a) presents the results from kernel *bpnn_ adjust_ weights_ cuda* of the benchmark *Back Propagation* [33]. Figure 5.7(b) presents kernel *compress* of the benchmark *DirectX Texture Compressor* [97]. Figure 5.7(c) presents kernel *bisectKernelLarge_ MultIntervals* of the benchmark *Eigenvalues* [97] and Figure 5.7(d) presents kernel *matrixMul* of the benchmark *Matrix Multiplication* [97]. The *MOBLIS* optimization was able to achieve reductions of the energy consumption by up



(a) Kernel: bpnn_adjust_weights_cuda

(b) Kernel: compress

(c) Kernel: bisectKernelLarge_MultIntervals

(d) Kernel: matrixMul

Figure 5.7: *MOBLIS* - Proportional Energy Consumption ($E_{ratio}^{P_\mathbf{d},P_l}$) and Proportional Runtime ($r_{ratio}^{P_\mathbf{d},P_l}$) Reduction for All Evaluated Individuals

to 9.12% whereas the runtime was reduced by up to 12.01%. They were measured for the kernel *d_recursiveGaussian_rgba* of the benchmark *Recursive Gaussian Filter* [97]. For other kernels, such as the *matrixMul* kernel no reduction in energy consumption and runtime was achieved.

Local instruction scheduling is a tedious optimization with the possibility of performance degradation and higher energy consumption as described in [59]. In that work the author applied different scheduling policies to GPGPU applications. As it turned out, no scheduling policy was able to optimize all benchmarks. A similar behaviour can be observed in Figures 5.7(a) - 5.7(d) by the fact that also values greater than 100% exist, which means that the runtime is longer and the energy consumption higher than without scheduling. Because of its adaptive approach, *MOBLIS* is able to avoid performance degradation and higher energy consumption.

A benchmark with a higher occupancy value but no performance increase is depicted in Figure 5.7(a). The Pareto front of kernel *bpnn_adjust_weights_cuda* exclusively

```
/*01c0*/ SYNC 01d0;
/*01c8*/ ST g[0x411], R8;
/*01d0*/ NOP;
/*01d8*/ BAR.SYNC;
/*01e0*/ MOV R10, R124;
```

(a) Unoptimized Sequence

```
/*01c0*/ SYNC 01d0;
/*01c8*/ ST g[0x411], R8;
/*01d0*/ MOV R10, R124;
/*01d8*/ BAR.SYNC;
/*01e0*/ ...
```

(b) Optimized Sequence

Figure 5.8: Instruction Sequences for Kernel *bisectKernelLarge_ MultIntervals*

consists of individuals with 10 registers per thread while the unscheduled version has 12 RPT. The occupancy value increases from 0.6 to 1. As can be seen, the performance increase and the energy consumption decrease is negligible. 1% of runtime reduction is equivalent to 0.5 milliseconds which is close to the measurement accuracy. As can be observed from Figure 5.7(b), the individuals are ordered in clusters. The two clusters in Figure 5.7(b) exist due to the high occupancy value for the allocation of threads/blocks to a streaming multiprocessor. In the unoptimized version of the kernel, each thread of the kernel *compress* needs 27 registers. After the optimization for each individual on the Pareto-front, a threads makes use of less than 25 registers, which ends up with an occupancy increase from 0.333 to 0.417. This leads to an energy consumption reduction of 3.5% and a runtime reduction of 3.3%. The most energy efficient individual has a maximal register use of 21 registers per thread (RPT), while the fastest solution has 24 RPT. Figure 5.7(c) shows the results for kernel *bisectKernelLarge_ MultIntervals.* The number of registers needed by a thread is constant at 17 registers. It is also remarkable that the performance counters derived for each individual count a different number of executed instructions and the same warp serialize rate at unchanged semantics of the program. The reduction of instructions arises from the fact that less instructions were executed, due to elimination of superfluous *no-op*s. Figures 5.8(a) and 5.8(b) illustrate the dissembled machine code for the Kernel *bisectKernelLarge_ MultIntervals.* Figure 5.8(a) shows the unoptimized version and Figure 5.8(b) shows an optimized version. As can be seen in Figure 5.8(a), there is a *no-op* at address $0x01d0$ where divergent thread paths must converge (sync instruction: $SYNC 01d0$;). Figure 5.8(b) shows an optimized kernel where the *MOV* instruction, which does not depend on the barrier synchronisation at $0x01d8$, substitutes the *NOP* as synchronisation point. This leads to an energy consumption reduction of 2.4% and a runtime reduction of 3.4%. Figure 5.7(d) shows the kernel *matrixMult* from the benchmark *Matrix Multiplication.* The register usage is 13 registers per thread and there is no reduction of the energy consumption and no improvement of the runtime possible with MOBLIS.

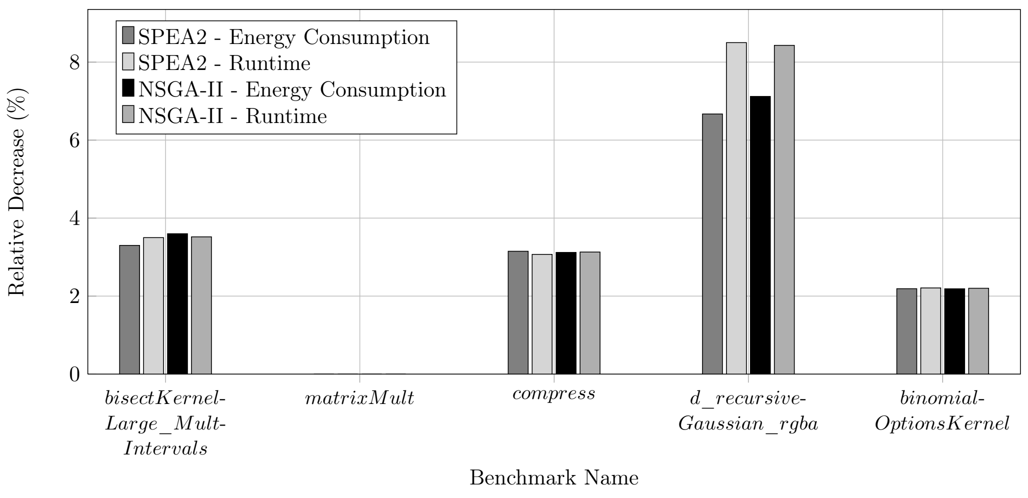


Figure 5.9: *MOBLIS* - Energy Consumption and Runtime Analysis for Different Multi-Objective Algorithms

5.3.3.3 SPEA2 and NSGA-II Comparison Results

MOBLIS can make use of different multi-objective algorithms (SPEA2, NSGA-II). This section presents an analysis on which GA is more suitable for *MOBLIS*. The results for energy consumption and runtime decreases – as percentage of the unoptimized version – for optimizations applying SPEA2 or NSGA-II are presented in Figure 5.9 (in percent on the Y axis). All values were measured on the GTS 250 and only exemplary kernels, which have an optimization gain, are shown. As can be summarized from Figure 5.9, there is no significant difference between the SPEA2 and NSGA-II. The same can be concluded from Figure 5.9 with the fact in mind that the energy consumption is less precise in comparison to the runtime measurements. In addition to the kernels presented in Figure 5.7, reductions of the following kernels are presented in Figure 5.9: kernel *d_ recursiveGaussian_ rgba* of the benchmark *Recursive Gaussian Filter* and Kernel *binomialOptionsKernel* of benchmark *Binomial Option Pricing.* For these kernels, runtime reductions between 2.19% and 7.12% and energy consumption reductions between 2.21% and 8.50% were achieved.

5.3.3.4 Platform Variants Comparison Results

In order to show that the optimization is not bound to a specific platform variant, the performance of *MOBLIS* for two different platforms is evaluated in this section. The first platform is the Nvidia 8400GS comprising one streaming multiprocessor with a memory bandwidth of 6.4GB/s. The second test candidate is a GTS 250 having 16 streaming multiprocessors with a memory bandwidth of 70.4GB/s. The latter corresponds to a memory bandwidth of 4.4GB/s per streaming multiprocessor. The

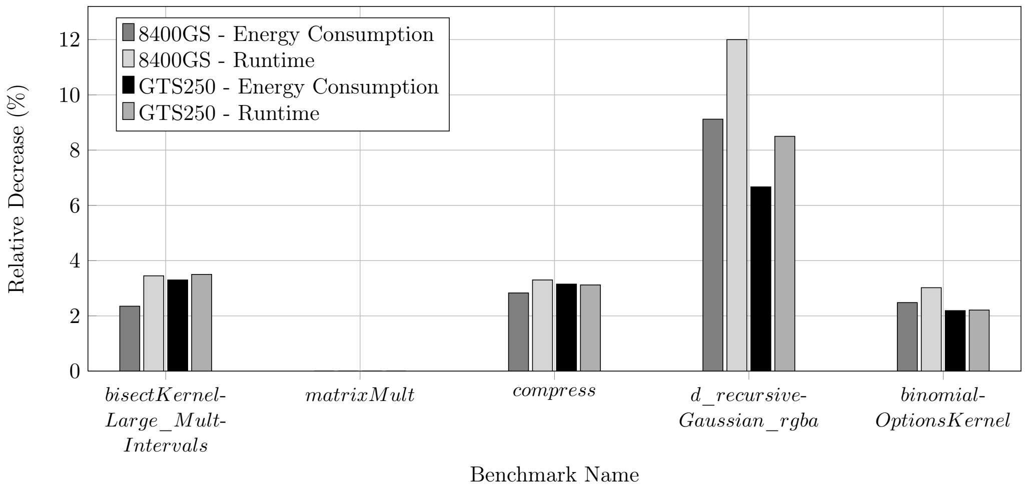


Figure 5.10: MOBLIS - Runtime and Energy Consumption Analysis for Different Graphics Cards

results for the achieved energy consumption and runtime reductions are presented in Figure 5.10. As in the former section, the same exemplary kernels are shown. Kernel *d_recursiveGaussian_rgba* has a higher optimization potential on the 8400GS. On the 8400GS, reductions in energy consumption of up to 9.12% respectively 12% in runtime can be achieved. For the GTS, only reductions in energy consumption of up to 6.67% respectively 8.50% in runtime can be achieved. For the other kernels, it can be summarized that there is only little difference between the optimization potentials on the different platforms.

5.3.3.5 Parallel Register Utilization

As described in the last section for the kernel *d_recursiveGaussian_rgba*, the runtime was optimized by up to 9% on the 8400GS and up to 7% on the GTS250. The parallel register utilization was restricted to register utilization decrease. Figure 5.11 shows that a register utilization reduction is not always the best solution. The test was conducted on the GTS 250. As can be seen a maximal energy consumption decrease of up to 14.46% and a runtime decrease of a up to 15.22% were achieved. This solution has a parallel register utilization of 37 RPT. An overview on the performance counters for two individuals with 37 RPT and 28 RPT is given in Table 5.2. As can be seen, the occupancy value (0.25) for individuals with 37 RPT is smaller in comparison to a 28 RPT solution (occupancy: 0.333). Nevertheless, the performance increases in the face of this lower occupancy value. It can be concluded that the kernel *d_recursiveGaussian_rgba* is highly sensible to the memory access pattern, the distribution of the memory instructions in the code and the allocation of the blocks onto the different streaming multiprocessors.

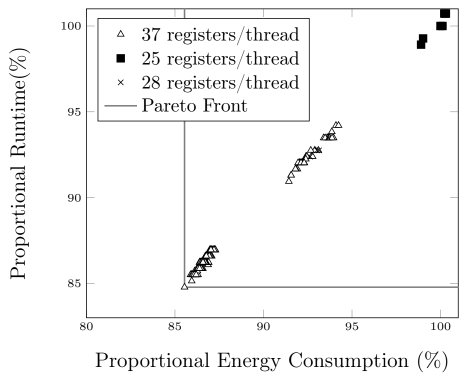


Figure 5.11: *MOBLIS* - Runtime and Energy Consumption Decrease for Kernel *d_recursiveGaussian_rgba* with Increased Parallel Register Utilization

	Individual with 37 RPT	Individual with 28 RPT
Register Ratio	0.9375 (7680 / 8192)	0.875 (7168 / 8192)
Active Blocks per SM	3/8	4/8
Active Threads per SM	192/768	256/768
Occupancy	0.25	0.3333

Table 5.2: *MOBLIS* - Performance Counter for Two Individuals of Kernel *d_recursiveGaussian_rgba*

5.3.4 Summary

Optimizations of GPGPU applications are usually performed in a manual error-prone trial-and-error process. In addition to that, the lack of energy consumption aware optimizations is unfavorable for green computing and the use of GPGPU-capable devices in mobile systems. Furthermore, the efficiency of a platform for a certain GPGPU application is not considered in the GPGPU application design process.

Therefore, this section presented an optimization process based on local instruction scheduling methods (*MOBLIS*). Six requirements listed in Section 5.3.1 had to be accomplished. The two most important requirements were optimal parallel register utilization and load balancing for concurrently running instruction and memory pipelines. These two requirements have been challenged by *MOBLIS* by changing the instruction sequence on a low-level representation with the help of local instruction scheduling. *MOBLIS* is based on a state-of-the-art multi-objective genetic algorithms to increase the performance and to decrease the energy consumption of the GPGPU applications. In addition to that, the use of a GA enables *MOBLIS* to optimize a GPGPU kernel in the face of lacking knowledge about the architecture of the platform (*Black Box Optimization*). The utilization of a profiling-based approach enables *MOBLIS* to take hardware platform characteristics into account.

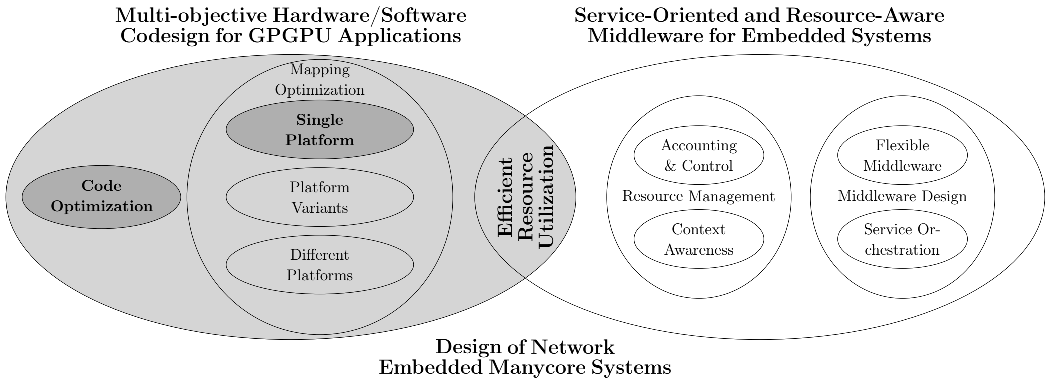

Multi-objective Hardware/Software Codesign for GPGPU Applications
Service-Oriented and Resource-Aware Middleware for Embedded Systems
Mapping Optimization
Single Platform
Platform Variants
Different Platforms
Code Optimization
Efficient Resource Utilization
Accounting & Control
Resource Management
Context Awareness
Flexible Middleware
Middleware Design
Service Orchestration
Design of Network Embedded Manycore Systems

Figure 5.12: *FALIS* - Overview on the Overall Design Process

By applying *MOBLIS* to a set of real-world benchmarks, up to 9% of energy and 12% of runtime could be saved on an Nvidia 8400GS and up to 6.67% of energy and 8.5% of runtime on a Nvidia GTS 250. Two state-of-the-art multi-objective genetic algorithms have been utilized to explore the solution space efficiently.

The technique of determining the local instruction scheduling policy with the help of a multi-objective optimization on a per basic block level is more efficient than pure local scheduling methods.

5.4 Multi-objective Global Instruction Scheduling

In this chapter, the second code optimizations for GPGPU application code will be introduced. The code optimization technique described in this section is called *FALIS* (Feedback-based and memory-Aware gLobal Instruction Scheduling) and is targeted towards an optimal distribution of memory-related instructions in a GPGPU application kernel. After a short introduction in Section 5.4.1 on why the optimization is needed, the optimization technique based on global instruction scheduling and state-of-the-art multi-objective genetic algorithms is presented in Section 5.4.2. The results of applying *FALIS* to several benchmarks are provided in Section 5.4.3. Finally, conclusions are presented in Section 5.4.4.

5.4.1 Introduction

The development of faster single core processors and the availability of higher performance due to higher clock frequency is at an impasse [21]. The shift towards multi-core and many-core systems is cumbersome because of the increasing complexity on programming and the efficient utilization of parallelism. In addition to that, the memory wall [83] still exists. On graphics cards, a larger number of processing cores exists which share a common memory. For these processing cores a sufficient amount of memory bandwidth must be available to exploit the parallel processing power. The authors of [68] showed that the number of load/store instructions, the number of all instructions and the bandwidth to the processing core of a GPU are crucial parameters to the performance of GPGPU applications. The overall number of load/store instructions is important w.r.t the performance due to possible pipeline stalls in the face of high main memory latencies. Nvidia GPUs can hide the latency of un-cached main memory accesses [98] by using concurrently running instruction and memory pipelines. If the number of the load/store instructions are too high, the full potential of the GPUs cannot be achieved [98]. In this section, an optimization is proposed which improves the performance of GPGPU applications by rearranging memory-related instructions employing instruction scheduling techniques. Therefore, the *Pipeline Load Optimization* requirement discussed in Section 5.3 is redefined to handle memory-related instructions explicitly.

Some requirements for *MOBLIS* can also be applied to the newly proposed *FALIS* and therefore, the following requirements should be handled in *FALIS*:

1. *Pipeline Load Optimization:* The placement of the memory-related instruction in the code of a GPGPU application should be optimized. The optimization objective is the load distribution between the instruction and memory pipeline and for stepping up the throughput of the application.

2. *Black Box Optimization:* Due to the lack of control in terms of the actual machine code generation, the register allocation and some low level optimizations, an optimization technique must take into account the concrete executable and the corresponding influence on the performance and energy consumption.

3. *Multi-objective Optimization:* As energy consumption is an important objective, a multi-objective approach should be used.

4. *Platform-aware Optimization:* The actual warp scheduling of the GPGPU application depends on the hardware configuration such as processor speed, memory speed, number of processing cores etc. It is therefore mandatory to optimize towards a concrete platform.

5. *Avoiding Unfavourable Solutions:* Analogous to *MOBLIS*, a possible performance degradation and a possible increase of the energy consumption is not acceptable and should be avoided in *FALIS*.

The optimization area which is considered for this optimization is *Optimization Area 2* (illustrated in Figure 5.1) which means that the machine code is optimized with the help of an optimizing compiler. Analogous to *MOBLIS*, *FALIS* provides a code optimization technique and will provide mapping optimization methods for a single GPU platform as depicted in the overall design process in Figure 5.3.

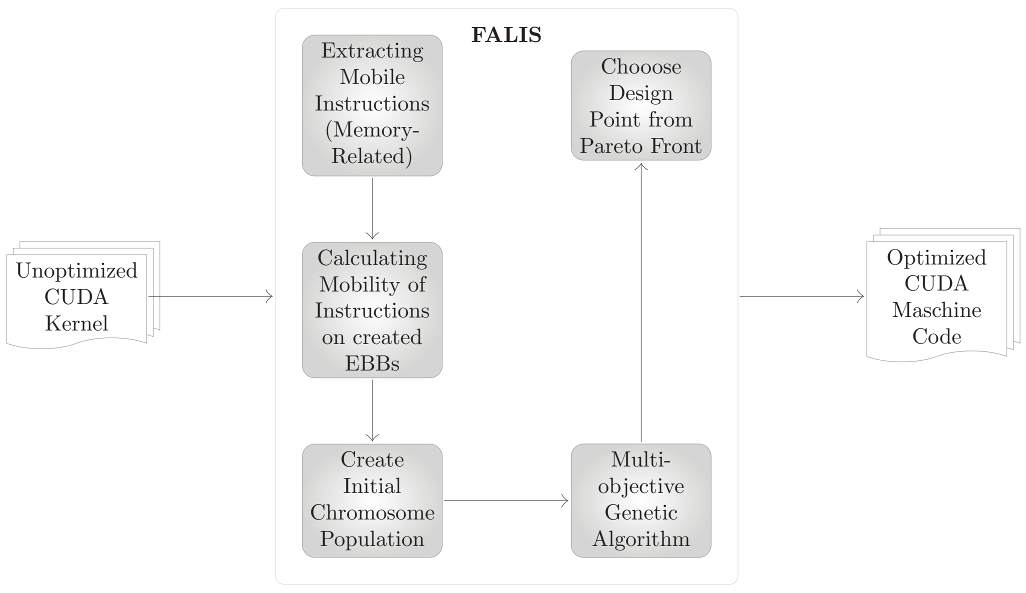

FALIS
Unoptimized CUDA Kernel
Extracting Mobile Instructions (Memory-Related)
Calculating Mobility of Instructions on created EBBs
Create Initial Chromosome Population
Multi-objective Genetic Algorithm
Chooose Design Point from Pareto Front
Optimized CUDA Maschine Code

Figure 5.13: Feedback-based and memory-Aware gLobal Instruction Scheduling (*FALIS*) Framework

5.4.2 *FALIS* - Materials and Methods

The *FALIS* framework is presented in this section. In contrast to *MOBLIS* represented in Section 5.3, only memory-related instructions are considered, and *FALIS* is a global instruction scheduling approach. The requirements *Black Box Optimization*, *Platform-aware Optimization*, *Multi-objective Optimization* and *Avoiding Unfavourable Solutions* are targeted, analogous to *MOBLIS*, with multi-objective algorithms utilizing profiling data from reference platforms.

The workflow of *FALIS* is presented in Figure 5.13. Analogous to *MOBLIS*, the *FALIS* optimization framework starts by reading in GPGPU code in C for CUDA and transforming it into a WHIRL intermediate representation (see Section 4.1.3.2). There are no dependencies between the execution of kernels because each kernel runs as a new application on a graphics card. Therefore, optimizations can be applied separately to each kernel. The output of the *FALIS* framework is an optimized machine code for a GPGPU kernel. The optimized machine code for each kernel is then combined into one optimized GPGPU application.

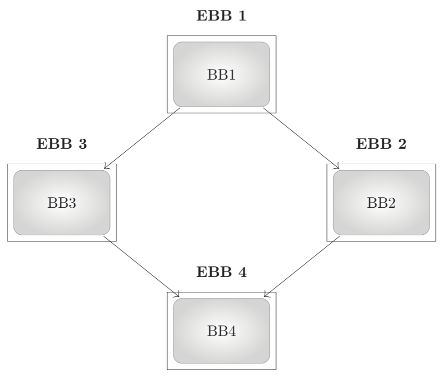
FALIS applies global instruction scheduling in order to optimize the *CUDA* kernels of a GPGPU application for a better utilization of the concurrently running instruction and memory pipelines by rearranging memory-related instructions. They are called Mob-Ins (Mobile Instructions). The optimization process of *FALIS* comprises the following steps (depicted in Figure 5.13): All memory-related instructions are extracted at the WHIRL level (see Section 5.4.2.1). Afterwards, extended basic blocks (see Definition 9) are created where an instruction can be scheduled to (see Section 5.4.2.2) and a mobility value is calculated. Finally, the sequences of instructions are optimized towards the objectives energy consumption and runtime performance by employing a genetic algorithm (SPEA2 or NSGA-II, see Section 5.4.2.3). The last step is the choice of generated *CUDA* kernel machine code as the final design. This can be e.g. a *CUDA* kernel optimized for energy consumption or runtime performance. This can also be a *CUDA* kernel which is optimized for both objectives.

5.4.2.1 Extracting Mobile Instructions

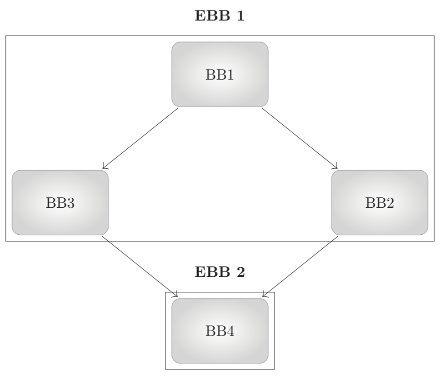
Mobile instructions for *FALIS* can comprise all load and store operations for the different memories (const, global, local, shared) and memory-related instructions such as (barrier) synchronisation statements. The reason why the latter is also considered will be described in the following. The atomic programming element of a Nvidia GPGPU application is a thread. A thread runs on a single *Streaming Processor* of a graphics card. A set of threads, called a block, is allocated on one *Streaming Multiprocessor*. At block level, there is no memory consistency and no fixed sequence on executing threads. Nevertheless, threads of a block can be forced to a consistent view on the main or shared memory and a certain execution point in the thread's code by using barrier synchronisation statements. Barrier synchronisation statements have to be added to the thread at source code level by the programmer. In Section 5.3.3 it was revealed that the performance can be decreased by such statements due to the requirement of adding additional instructions at machine code level, ensuring proper semantics of the GPGPU application. The evaluation results in Section 5.3.3 showed that it is not always mandatory to add these instructions if other existing instructions can substitute them (depicted in Figure 5.8(b)). The substitution can possibly save cycles, resulting in performance increase and energy consumption decrease. Thus, the approach presented in MOLIS is designed to also treat synchronisation statements in addition to memory instructions. For all mobile instructions, the optimal position in the code should be determined. In the scope of FALIS, the position of each extracted mobile instruction is a variable. The variability is explained in the following section.

5.4.2.2 Calculating Mobility of Instructions on Extended Basic Blocks

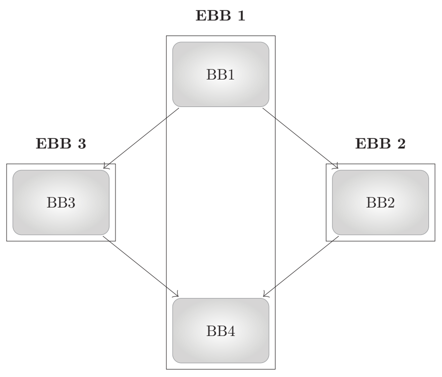
The purpose of *FALIS* is to move mobile instructions in such way that they access the GPU's memory system in a more efficient way. Especially when placed in the code far away from each other, they cannot interfere with each other. This is, in particular, important when the limited bandwidth of the graphics card's main memory should be utilized in an efficient way [68]. The authors revealed that many concurrent memory accesses can decrease the performance. Figure 5.14 shows different methods for instruction scheduling by illustrating the different structure of



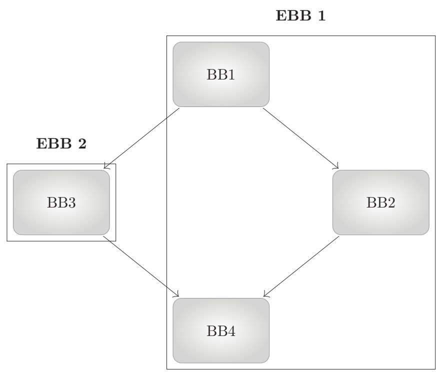
(a) Local Scheduler



(b) Treegion Scheduler



(c) Branch Head Partitioning



(d) Trace Scheduler

Figure 5.14: Coverage of Different Instruction Scheduling Methods

the extended basic blocks (see Definition 9). Therefore, exemplary CFGs (Control Flow Graphs) with four basic blocks per sub-figure are depicted. In Figures 5.14(a) - 5.14(c), different EBBLs (Extended Basics Block Label) mean that an instruction cannot be moved from one basic block to another, whereas basic blocks with the same EBBL mean that instructions can possibly be exchange between them. In Figure 5.14(a) one can see local instruction scheduling. It can only schedule an instruction inside one basic block as denoted by the different EBBL of the basic blocks. Therefore, methods considering several basic blocks for the scheduling of instructions are required. An example for a state-of-the-art technique is TREEGION scheduling [12]. It can schedule instructions to adjacent basic blocks. TREEGION scheduling uses compensation code which may adversely affect the performance because code which is executed predicated is slower than normally executed code on the GPU [65]. Therefore, a technique called *branch-head-partitioning* was introduced in [59] which enables a global scheduler to schedule along traces (e.g. extended basic blocks) without the use of compensation code but with the possibility to schedule an instruction far away from the original basic block as depicted

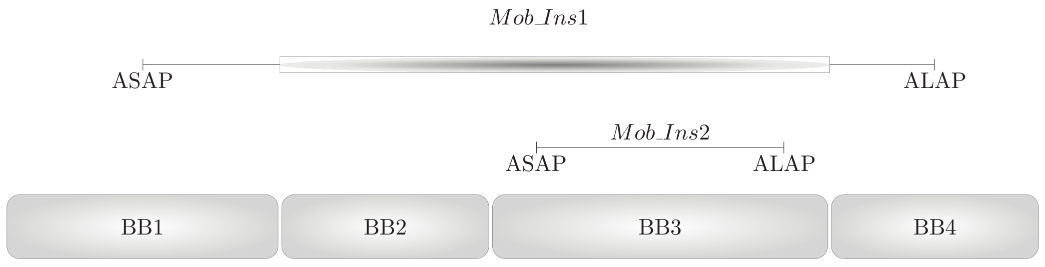


Figure 5.15: Mobility of Mobile Instructions (Mob-Ins)

in Figure 5.14(c). *Branch-head-partitioning* is related to Trace scheduling [52] (depicted in Figure 5.14(d)) from the CFG partitioning point of view but adds some restrictions/features to the partitioning of the CFG (sse Definition 6):

- A trace can be interrupted, e.g. by a divergent control flow.
- The basic blocks inside a divergent control flow are combined to sub traces.
- No compensation code is allowed, due to a possible negative effect on the runtime behaviour [65].

FALIS works on a combined control flow (see Definition 6) and data dependency graph (see Definition 7). This type of graph is called program dependency graph (see Definition 8) and is specified by $D = \langle V, E \rangle$. The set of all instructions in intermediate representation of one kernel is $V = \{i_1, .., i_n\}$ and $E \subseteq V \times V$ are the data dependencies or control flow dependencies between instructions. D should comprise also the barrier synchronisation statements. In order to take the barrier synchronisation instruction within global instruction scheduling into account, the original graph D is used as a basis. In order to maintain the semantics of a kernel, for each load/store instruction $i_x \in V$ preceding a barrier synchronisation instruction $i_y \in V$ $(x < y)$, a dependency edge is inserted between i_x and i_y. The same is done for a barrier synchronisation instruction and all succeeding load/store instructions.

For calculating the mobility of a mobile instruction, firstly, ASAP scheduling (As Soon As Possible) [141] is conducted with the help of the program dependency graph D. This reveals the lower bound to where the memory instruction can be relocated. In a second step a scheduling with scheduling policy ALAP (As Late As Possible) [73] is performed to determine the upper bound for positions. Both, ASAP and ALAP were originally used at the synthesis of hardware but also work on other graphs like combined control flow and data dependency graph utilized for *FALIS*. After each scheduling the position of the instructions are reverted to the original position. The mobility interval for a mobile instruction is the interval between the ASAP and the ALAP position. In Figure 5.15 an example is shown for Mob_Ins_1

which can be scheduled in basic blocks $BB1$ and $BB4$. The shaded area for mobile instructions in Figure 5.15 marks positions (basic blocks $BB2$ and $BB3$) where Mob_Ins_1 cannot be scheduled to.

The mobility intervals are employed by *FALIS* in order to optimize a program. As it was revealed in *MOBLIS*, a GA is an appropriate optimization technique in the field of instruction scheduling for GPGPU applications since it can cope with effects of the warp scheduler of Nvidia graphics cards.

5.4.2.3 *FALIS* Genetic Algorithm Specification

The instruction scheduling is done by changing the position of mobile instructions in the WHIRL representation. In the following, used terms in the scope of *FALIS* are introduced:

- k: k is the number of *WHIRL* instructions in the code.
- n: Number of all mobile instructions in the *WHIRL* representation of a *CUDA* kernel.
- M: The set of all mobile instructions is denoted by $M = \{Mob\_Ins\_i | 1 \leq i \leq n\}$.
- **Gene** $g_i \in G = \{g_i | 1 \leq i \leq n\}$: A gene g_i represents the position of a mobile instruction Mob_Ins_i in the WHIRL representation of a CUDA kernel.
- **Individual d**: In the compilation process, individual **d** is an instance of a GPGPU kernel. Each individual $\mathbf{d} \in \mathcal{I}$ comprises a gene sequence $g_1, ..., g_n$. For each g_i, a position is assigned to a mobile instruction. Each possible position is represented by a natural number in the interval $[asap, alap]$, whereas $asap \in [1, .., m]$, $alap \in [1, .., k]$, $asap \leq alap$.

It should be noticed that an individual can possibly comprise an invalid instruction schedule, since on the GA level no dependency tracking is performed. Handling and sorting out of invalid solutions is presented in the following section. If two mobile instructions, Mob_Ins_i and $Mob\_Ins\_j, i < j$ are assigned to the same position in the individual, Mob_Ins_i will be scheduled before Mob_Ins_j.

5.4.2.4 Optimization Workflow

In Figure 5.16, the *FALIS* individual evaluation flow is depicted. Analogous to *MOBLIS*, for global instruction scheduling, the *NVOpenCC* from Nvidia (*Open64* [80]) was enhanced. The code generation module of *NVOpenCC* was extended to read in individuals created by *FALIS* . In contrast to *MOBLIS* , invalid chromosomes can be created, which represent a GPGPU kernel which can have a semantics different from the original version. Therefore, a chromosome validator was implemented which checks if the created program has still the same semantics by evaluating the

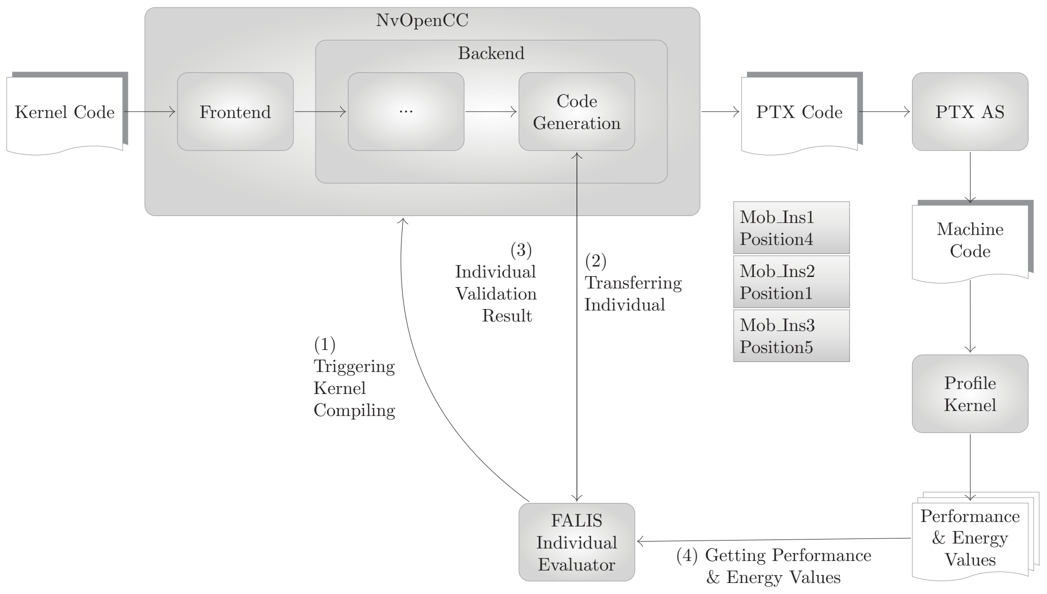

NvOpenCC
Backend
Kernel Code
Frontend
...
Code Generation
PTX Code
PTX AS
Machine Code
Profile Kernel
Performance & Energy Values
Mob_Ins1 Position4
Mob_Ins2 Position1
Mob_Ins3 Position5
(1) Triggering Kernel Compiling
(2) Transferring Individual
(3) Individual Validation Result
(4) Getting Performance & Energy Values
FALIS Individual Evaluator

Figure 5.16: *FALIS* - Individual Evaluation Flow

program dependency graph D (see Section 5.4.2.2). If e.g. in the unoptimized version of a GPGPU kernel an instruction i_x depends on the result of instruction i_y, then in the optimized version of the kernel the sequence of these instructions must not be changed, i.e. i_x must be executed after i_y. The performance and energy consumption values are profiled and processed as described in Section 5.3.2.2.The fitness values are calculated and utilized within SPEA2 and NSGA-II (see Section 4.2.2.2).

5.4.2.5 Evolution Operations

Analogous to *MOBLIS* (see Section 5.3.2.3), the crossover and the mutation operators for *FALIS* are single point operators. For all genes that are altered in the mutation process, a second random process is done. With the probability of 0.5 the gene is altered in the interval $[asap, alap]$, and with a probability of 0.5 the gene is altered only in the interval $[pos_1, pos_2]$, where $pos_1 = \max(pos_{alt} - 1, asap)$, $pos_2 = \min(pos_{alt} + 1, alap)$ and pos_{alt} is the position gene of the parent chromosome. The crossover operation is done analogously to the *MOBLIS*.

5.4.3 Evaluation

In this section the evaluation of the optimization potential by applying *FALIS* is presented. Firstly, the configuration and parameters for the evaluation are given in Section 5.4.3.1. Results for a Nvidia 9500GT are presented in Section 5.4.3.2.

Benchmark	Kernel	Number of Mobile Instructions
Euler3D	*cuda_ compute_ flux*	110
Matrix Multiplication	*matrixMul*	42
SRAD	*srad_ cuda_ 1*	64
Separable Convolution	*convolutionRowsKernel*	304
Separable Convolution	*convolutionColumnsKernel*	304
Recursive Gaussian Filter	*d_ recursiveGaussian_ rgba*	21
DCT8x8	*CUDAkernelQuantizationShort*	2

Table 5.3: Number of Mobile Instructions for Kernels optimized by *FALIS*

Afterwards, the results for several GAs are compared with each other. In Section 5.4.3.4 the results for a Nvidia 9500GT are compared to another GPU.

5.4.3.1 Parameters / Configuration / Optimization Runtime

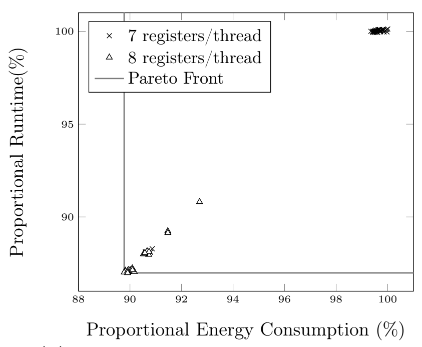
The population size μ is the maximum of $n/2$ and 10 to ensure that the solution space is sufficiently explored. The number of mobile instructions (n) for different kernels is listed in Table 5.3. As can be seen in that table, the number of mobile instructions per kernel is between 2 and 304. The algorithms last 30 generations for Section 5.4.3.2 and 10 generations for the evaluations in other sections. The probability for mutating genes of a chromosome was $p_m = 0.40$ and the probability for crossover was $p_c = 0.20$ – according to the definitions in Section 4.2.1. Each individual is evaluated 10 times and the median was selected as average energy consumption and runtime result.

The results of the evaluation were obtained with the presented energy consumption and performance testbed (see Chapter 3). The graphics cards used in the evaluation comprise a 9500GT and a GTS250. The SPEA2 and NSGA-II implementations from the JECO library [113] were used in *FALIS* analogously to *MOBLIS*.

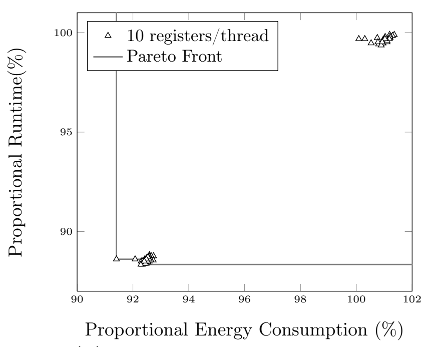
The runtime of applying *FALIS* to a GPGPU kernel strongly depends on the runtime of the GPGPU kernel on a particular platform itself, because for each individual the benchmark is executed. Therefore, runtimes between several hours and multiple days were observed.

5.4.3.2 Single Platform Results

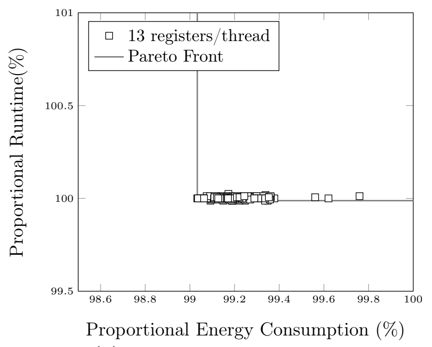
Figures 5.17(a) - 5.17(d) show the energy consumption and runtime reductions – as percentage of the unoptimized version – achieved by applying *FALIS* to different benchmarks running on a Nvidia 9500GT. Each point inside the figures shows the runtime and energy consumption for one individual. The Pareto front connects the Pareto-optimal points with a line. As can be seen in these figures, runtime and energy consumption can be decreased, due to the changed scheduling of memory-related instructions. The maximal energy consumption reduction on the 9500GT (depicted in Figure 5.17(a)) amounts to 10.23% and the maximal value for runtime



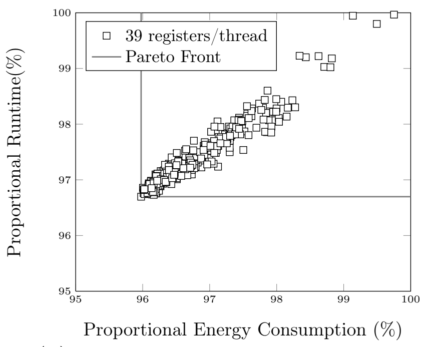
(a) Kernel: *convolutionRowsKernel*



(b) Kernel: *srad_cuda_1*



(c) Kernel: *matrixMul*



(d) Kernel: *cuda_compute_flux*

Figure 5.17: *FALIS* - Proportional Energy Consumption ($E_{ratio}^{P_{\mathbf{d}},P_l}$) and Proportional Runtime ($r_{ratio}^{P_{\mathbf{d}},P_l}$) Reduction for all Evaluated Individuals

decrease is 13.02%. These values are achieved for kernel *convolutionRowsKernel* of the benchmark *Separable Convolution* [97]. When optimizing the kernel *convolutionRowsKernel*, the change in register utilization – 7 registers per thread changed to 8 registers per thread – has a positive effect on the runtime and the energy consumption. Another kernel which was accelerated significantly is *srad_cuda_1* of the benchmark *SRAD* [33]. Detailed evaluations of explored individuals are depicted Figure 5.17(b). A runtime decrease of 11.66% and an energy consumption decrease of 8.59% can be achieved for kernel *srad_cuda_1*. As one can see from Figure 5.17(b), there are two clusters. One cluster comprises individuals which have an impact on the runtime and the energy consumption. The other cluster comprises individuals which do not lead to a runtime and the energy consumption reduction. Not all benchmarks which have been tested, can be optimized with the *FALIS* optimization process. An example where no optimization is achieved, is depicted in Figure 5.17(c). In this figure, the kernel *matrixMul* from benchmark *Matrix Multiplication* is shown. As can be seen, the individuals created exhibit no runtime decrease at all and an energy consumption decrease below 1%. The latter is below

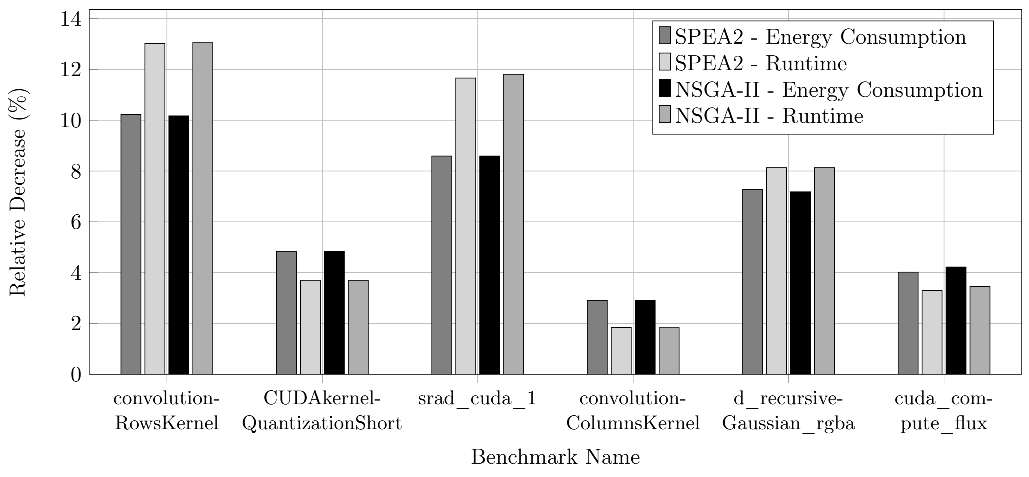


Figure 5.18: *FALIS* - Energy Consumption and Runtime Analysis for Different Multi-Objective Algorithms

the accuracy of the energy consumption measurements. In Figure 5.17(d), the kernel *cuda_compute_flux* of benchmark *Euler3D* is presented. Every created schedule has a positive effect on the runtime (reduction: 3.3%) and the energy consumption (reduction: 4.02%).

5.4.3.3 SPEA2 and NSGA-II Comparison Results

FALIS can make use of different multi-objective algorithms (SPEA2, NSGA-II). In this section, an analysis aiming at showing which GA is more suitable for FALIS is presented. The results for energy consumption and runtime decreases for optimizations applying SPEA2 or NSGA-II are presented in Figure 5.18 (in percent on the Y axis). All values were measured on the 9500GT and only exemplary kernels which were optimized are shown. In addition to the kernels optimized in Figure 5.17, reductions of the following kernels are presented in Figure 5.18: kernel *d_recursiveGaussian_rgba* of the benchmark *Recursive Gaussian Filter*, the kernel *convolutionRowsKernel* of the benchmark *Separable Convolution* and kernel *CUDAkernelQuantizationShort* of the benchmark *DCT8x8*. For these kernels, runtime reductions between 1.84% and 8.13% and energy consumption reductions between 2.91% and 7.18% were achieved. In total, there is no significant difference in the runtime decrease between the SPEA2 and NSGA-II. The same can be concluded for the energy consumption of the benchmarks.

5.4.3.4 Platform Variants Comparison Results

In order to show that the optimization is not limited to a specific platform variant, the performance of *FALIS* for two platforms is evaluated in this section. The

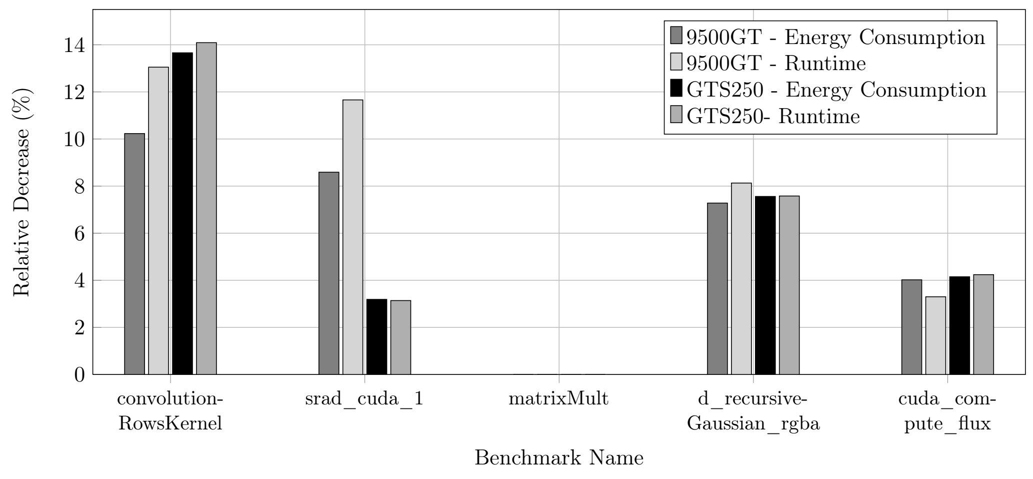


Figure 5.19: *FALIS* - Energy Consumption and Runtime Analysis for Different Graphics Cards

first candidate is a Nvidia 9500GT comprising four streaming multiprocessors with a memory bandwidth of 25.6GB/s (6.4GB/s per streaming multiprocessor). This card was utilized in the former sections. The second test candidate is a Nvidia GTS 250 having 16 streaming multiprocessors and a memory bandwidth of 70.4GB/s. The latter corresponds to a memory bandwidth of 4.4GB/s per streaming multiprocessor. The energy consumption and runtime reduction values for different kernels are depicted in Figure 5.19. As can be seen in this figure, the achievable reductions are similar for the different hardware platforms. For the kernels *d_recursiveGaussian_rgba* and *cuda_compute_flux*, the performance is increased and the energy consumption is decreased. Kernel *matrixMul* cannot be optimized for any platforms.

A different situation can be noticed from the kernel *srad_cuda_1*, as its energy consumption decreases by 8.59% for the 9500GT and 3.19% for the 250GTS. The runtime decreases by 11.66% for the 9500GT and 3.14% for the 250GTS. A similar situation can be noticed from kernel *convolutionRowsKernel*. The kernel has a higher optimization gain on the GTS250.

This effect was further examined by optimizing the load and store instructions for the different memory spaces (global, const, shared, param) separately. The energy consumption reduction values for the different kernels and memory-spaces are depicted in Figure 5.21 (runtime reduction values in Figure 5.20). These figures depict the energy consumption and runtime decrease for the different aforementioned memory spaces and the kernels *convolutionRowsKernel*, *d_recursiveGaussian_rgba* and *cuda_compute_flux* and *srad_cuda_1*. For kernel *convolutionRowsKernel*, only the combined optimization on all memory spaces has a positive effect on the energy consumption and the runtime. For the kernels *d_recursiveGaussian_rgba* and

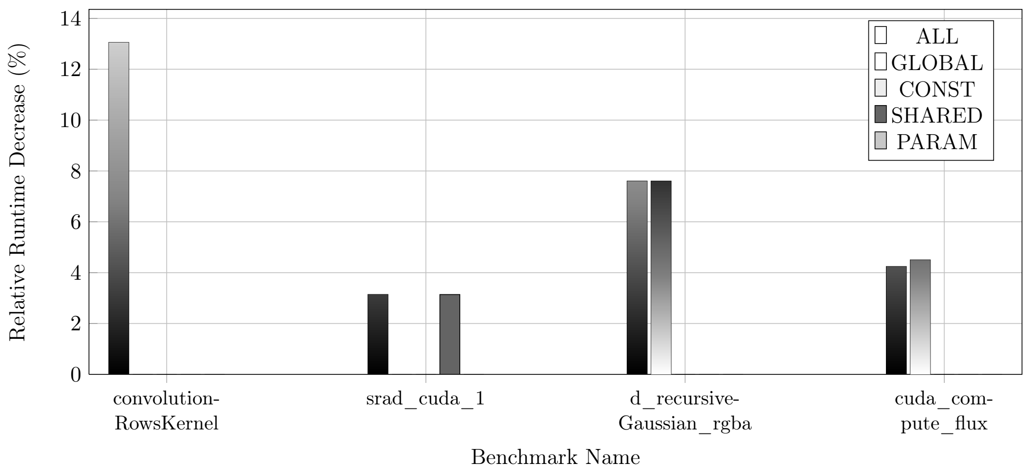


Figure 5.20: *FALIS* - Runtime Analysis of Different Memory Spaces

cuda_compute_flux the access to the global memory was the bottleneck. Kernel *srad_cuda_1* shows different optimization gains for both graphics cards 9500GT and GTS250. It turned out, that the access to shared memory is the bottleneck of the kernel which has a different effect on the runtime and energy consumption on the different platforms.

5.4.4 Summary

Within the manually performed GPGPU application design process, little interest was focussed on the placement of the memory-related instructions in the GPGPU application. This was nearly impossible due to lack of efficient compiler support and the lack of automatic performance analysis. In addition to that, the energy consumption of a GPGPU application was never considered at compiler level before.

A promising technique which can alter the schedule of memory instructions inside the application code is instruction scheduling. It is performed on a medium-level intermediate representation and can optimize the performance of an application. The latter is accomplished by a better utilization of the graphics card's pipelines. The *FALIS* approach utilized global instruction scheduling of memory-related instructions to optimize a GPGPU application. This is especially important in the face of the memory wall problem. With *FALIS*, reductions of up to 10.23% in energy consumption and 13.02% in runtime can be achieved for real-world benchmarks on different graphics cards..

An idea for future work is, amongst others, whether global instruction scheduling of all types of instructions is even more beneficial or not. Due to the fact that by only altering the positions of load/store instructions and the barrier synchroni-

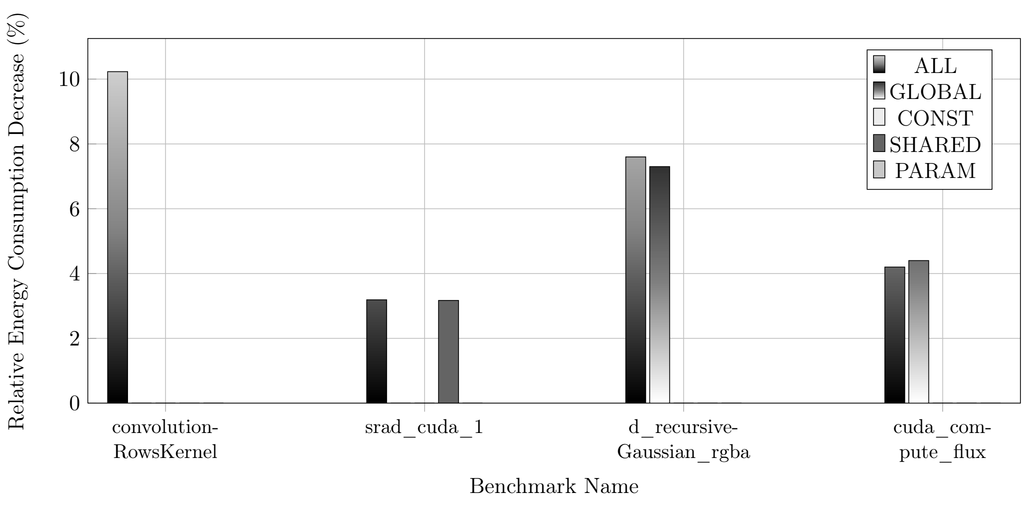


Figure 5.21: Energy Consumption Analysis of Different Memory Spaces

sation statements, it was not possible to construct an optimization as presented in Figure 5.8 which substitutes the unused *no-op* instructions in the code. Since the WHIRL and PTX intermediate representations lack these instructions, *FALIS* could not imitate processes and optimizations of the actual code generation. A possible solution for this is to implement the code generation for GPGPU machine code itself and work on a low-level intermediate representation.

5.5 Design Space Exploration for Embedded Image Processing Systems

The first DSE (Design Space Exploration) for GPGPU application code will be introduced in this chapter. Firstly, in Section 5.5.1, an introduction and a presentation of the use case application considered in the DSE is presented. Secondly, the concept of the DSE is introduced in Section 5.5.2. The design space exploration is accomplished for an image processing system to select a platform from a given set of platforms with energy efficiency as a second objective in addition to real-time requirements. In the scope of the overall design process, depicted in Figure 5.22, in this section a mapping optimization method for a GPU platform family is provided. The corresponding results for the DSE are given in Section 5.5.3. The section ends with a summary in Section 5.5.4.

5.5.1 Introduction

Up to now, in Sections 5.3 and 5.4 only *Optimization Area 2* (Figure 5.1) was considered. In this section, a mapping optimization with a design space exploration

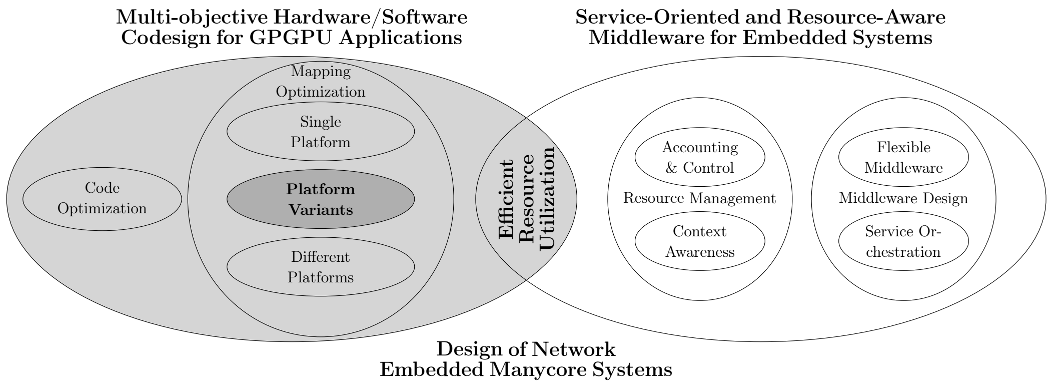

Multi-objective Hardware/Software Codesign for GPGPU Applications
Service-Oriented and Resource-Aware Middleware for Embedded Systems
Mapping Optimization
Single Platform
Code Optimization
Platform Variants
Different Platforms
Efficient Resource Utilization
Accounting & Control
Resource Management
Context Awareness
Flexible Middleware
Middleware Design
Service Orchestration
Design of Network Embedded Manycore Systems

Figure 5.22: Embedded Image Processing Systems - Overview on the Overall Design Process

is considered. Variable parameters of the DSE are:

- *Thread Set Partitioning:* The first parameter in this design space exploration is the partitioning of the set of threads which comprises *Optimization Area 1* (see Figure 5.1). GPGPU application code allows an unlimited thread parallelization grade at programming level, but due to restrictions for parallelism on the real hardware platform, not all threads can run concurrently. Therefore, the threads must be partitioned into groups. The partitioning of the *work-items* set into *work-groups* in *OpenCL C* has a direct impact on the allocation or mapping of the *work-groups* onto the GPU. This has in turn a major impact on the energy consumption of the system. The partitioning into *work-groups* has to be done individually for each kernel of the GPGPU application.

- *Platform Selection:* When talking about energy-efficiency and efficient mapping, an additional parameter is the execution platform itself. For example, a too powerful platform could be selected for a GPGPU application, which then consumes too much energy. Another case is that the selected platform is efficient w.r.t. the consumption of energy but not powerful enough to meet real-time deadlines. Because of these two considerations the platform decision (*Optimization Area 3*) must be taken in account in the GPGPU application design process. With this optimization area, the scalability of the algorithms is an important factor. This is crucial because in case that a GPGPU application is memory bounded, an increasing number of processing cores will not increase the performance when reaching a certain threshold. Furthermore, the performance can possibly not scale with the number of cores if there are too many data dependencies inside the kernels making synchronization necessary. If the performance does not scale with the number of cores, this has also an effect on the energy consumption of the system because the performance to energy ratio probably worsens. The theoretical background for speedup and

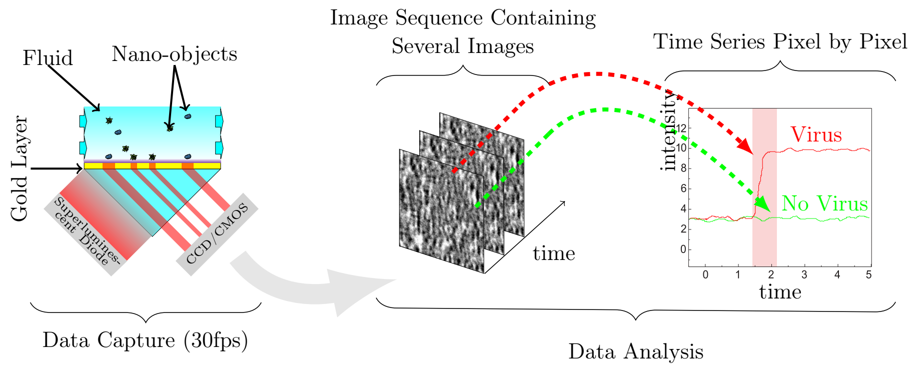


Figure 5.23: Processing Requirements of the *PAMONO* Biosensor (modified [149])

scalability is described by Amdahls Law [7] which can be used to estimate the performance speedup that can be achieved by parallelizing an application and running on a parallel processor. Amdahl separates the application in parts (represented by application fraction f) which can be computed in parallel and parts that must be computed sequentially (represented by application fraction $(1 - f)$) as depicted in Equation 5.3. The acceleration of the parallel fraction is then expressed by $\frac{f}{n}$ if the parallel part can be independently computed on the n processors. The overall speedup *sup* can be calculated as follows:

$$sup = \frac{1}{(1-f) + \frac{f}{n}}. \tag{5.3}$$

The process of designing a GPGPU accelerated system should involve a design space exploration (described in Section 5.5.2) which optimizes the described parameters of the system under resource constraints at design time. The constraints for the design space exploration to be presented in this section are

- soft deadlines
- energy consumption.

To the best of the author's knowledge the consideration of the energy consumption and the platform decision has not been done before for a GPGPU application.

Use Case - Biomedical Scenario: The *PAMONO* biosensor described in Section 2.1 provides an in-situ detection of nano-objects and a detection in realtime which means that the result of a detection in progress can be visualized online while inserting the specimen. The processing system (depicted in Figure 5.23) must cope with a frame rate of the camera. In the scope of this section, a frame rate of 30 fps

(frames per second) is assumed.
The expected local availability of such biosensors at distributed sites can lead to the situation that no wired power supply is available. Therefore, the final platform should be a mobile and battery-driven hardware platform. Energy awareness should be one of the objectives when designing such a GPGPU-accelerated system. The major requirements for the design of such a virus detection system can be summarized as follows:

1. Provide a precise virus detection rate.
2. The system should be capable of working in real-time.
3. Take energy-efficiency into account.

5.5.2 Materials and Methods

The design space exploration process is depicted in Figure 5.24. It should take the two parameters (*Thread Set Partitioning* and *Platform Selection*) listed in Section 5.5.1 into account. The DSE process starts with the specification and implementation of the GPGPU application. It is assumed that the application is a parallel application and that it can be accelerated by executing it on a GPU. The output of the design space exploration is an optimal mapping onto a GPU and the selection of the most suitable platform. The design space exploration starts with the selection of an execution platform, then the application is mapped onto the platform by the GPGPU application design process presented in Section 5.1. The latter can also include the code optimization techniques presented in Sections 5.3 and 5.4. The output of the mapping onto the selected platform is evaluated with regard to performance and energy consumption employing the testbed presented in Section 3.2.

Design Space Exploration Target: The energy consumption should be minimized under the restriction that the system is capable of coping with the input data rate of the camera. Figure 5.25 illustrates a scheme of the periodically processing of the image processing and analysis pipeline presented in Section 2.1. The processing interval for one image (frame) lasts 33.3 ms, due to the camera speed. A new frame is captured by the camera each 33.3 ms and must be processed by the image processing and virus detection pipeline. The dashed curve represents the power consumption of the graphics card while processing a frame. This comprises transferring the data to and from the graphics card's main memory, creation of the threads for different kernels and the execution of the kernels.

As already described in Section 3.2, the testbed can provide the energy consumption $energy(P_h, P_l, t_{start}^{P,P_l}, t_{end}^{P_h,P_l})$, where as $P_h, h \in H$ is the program to be profiled, $P_l \in K$ the execution platform consuming power over time, $t_{start}^{P_h,P_l}$ the profiling start

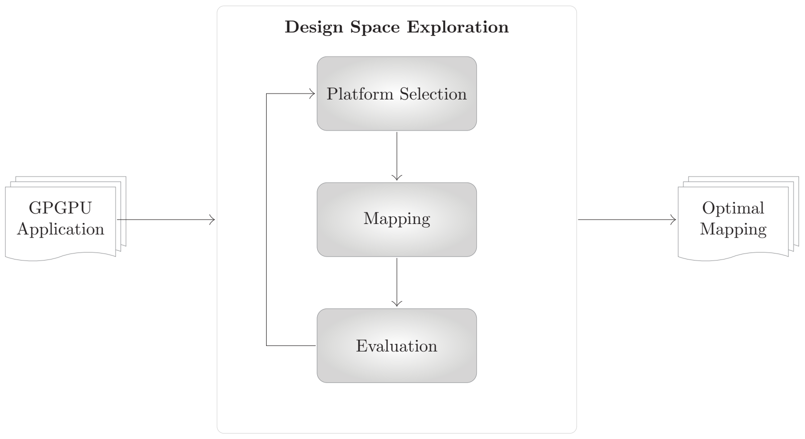


Figure 5.24: Design Space Exploration Workflow for Embedded Image Processing Systems

point in time and $t_{end}^{P_h,P_l}$ the profiling end point in time. K is in this context the set of available graphics cards. In this design space exploration, the partitioning of threads is important and therefore, H is the set of available partitions. The energy consumption $E_{dse}^{P_h,P_l}$ for processing one frame is described as

$$E^{P_h,P_l} = energy(P_h, P_l, t_0^{P_h,P_l}, t_{SLOT}^{P_h,P_l}), \tag{5.4}$$

where $P_l \in K$ and $P_h \in H$. The energy consumption $E_{T_0,T_{run}}^{P_h,P_l}$ for the processing is:

$$E_{T_0,T_{run}}^{P_h,P_l} = energy(P_h, P_l, t_0^{P_h,P_l}, t_{run}^{P_h,P_l}) \tag{5.5}$$

and the energy consumption $E_{T_{run},T_{SLOT}}^{P_h,P_l}$ for the idle phase is

$$E_{T_{run},T_{SLOT}}^{P_h,P_l} = energy(P_h, P_l, t_{run}^{P_h,P_l}, t_{SLOT}^{P_h,P_l}). \tag{5.6}$$

The testbed provides the points in time for the following triggers for the P_l and P_h (depicted in Figure 5.25):

- $t_0^{P_h,P_l}$: Start time for the process interval and task processing.
- $t_{RUN}^{P_h,P_l}$: End time for task processing plus data transfers between the graphics card and the host.
- $t_{IDLE}^{P_h,P_l}$: Start time for the idle phase.
- $t_{SLOT}^{P_h,P_l}$: End time for the process interval.

The solution set E is the set of all evaluated graphics card and partitioning configurations:

$$E = \{E_{dse}^{P_h,P_l} | P_l \in K and P_h \in H\}. \tag{5.7}$$

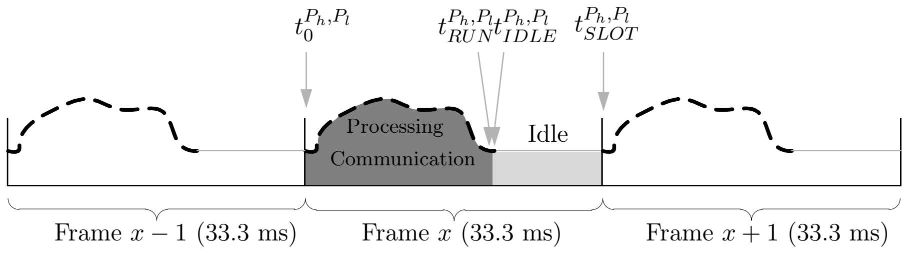


Figure 5.25: Energy Consumption for Frame Processing

The minimal energy consumption E_{min} is then:

$$\exists E_{min} \in E, \forall E_{el} \in E : E_{min} \leq E_{el}. \tag{5.8}$$

E_{min} has the requirement, that all processing must be completed before the arrival of the next frame ($t_{run}^{P_h,P_l} < t_{slot}^{P_h,P_l}, t_{slot}^{P_h,P_l} = 33.3\text{ms}$). The energy consumption for the host system is not considered. In addition to E_{min} also the runtime r_{min} is provided.

5.5.3 Evaluation

In this section, the evaluation of the design space exploration is presented. Firstly, the configuration and parameters for the evaluation are given. Then the results for the different input data are presented, followed by the results for the scalability of the GPGPU application and the results for the energy consumption on different platforms.

5.5.3.1 Parameters and Configuration

The three-step image processing and analysis pipeline presented in Section 2.1 and depicted in Figure 2.2 is mapped to 19 GPGPU kernels in software. Seven kernels initialize the data structures and download the analysis results after completing one run. The other kernels are executed for the processing of each frame and are therefore the input of the design space exploration. The optimal partitioning has to be found for these 12 kernels.

In the context of this DSE, the ratio of program parts running sequentially on the CPU is very small and therefore can be neglected, because the image processing and virus detection pipeline is completely implemented as GPGPU application and the CPU parts are only for uploading/downloading the image data and for managing the processing on the graphics card.

The different tested platforms are depicted in Table 3.1 (Nvidia ION, 9600GT, GTS250), along with the most important performance factors. The last column of

Work Group Size		Wavelet Denoising		Pattern Matching	
X	Y	portrait (ms)	landscape (ms)	portrait (ms)	landscape (ms)
16	2	1.682	1.696	4.005	4.257
16	4	**1.640**	**1.634**	3.341	3.376
16	8	1.645	1.660	3.538	3.491
2	16	2.235	2.408	3.388	3.316
2	2	6.376	6.360	8.345	7.677
2	4	3.674	3.721	5.014	4.523
2	8	2.647	2.776	3.809	3.517
4	16	1.894	1.974	3.345	**3.266**
4	2	3.569	3.570	4.925	4.656
4	4	2.480	2.494	3.621	3.425
4	8	1.933	2.011	3.376	3.364
8	16	1.941	1.873	3.516	3.509
8	2	2.509	2.472	3.696	3.886
8	4	1.898	1.900	3.337	3.384
8	8	1.874	1.863	**3.311**	3.363

Table 5.4: Runtimes for Different Input Data Formats on 9600 GT

this table lists the idle power consumption of the platforms. The lower values represent the power consumption with power saving techniques of the graphics cards whereas the higher value represents the power consumption without power saving techniques.

5.5.3.2 Input Data Dependency

In this section, the influence of the input data format in the design space exploration process is examined. Today's GPGPU applications are often designed to provide the best possible average performance for different input sizes. For example, video decoding is optimized to work with different video resolutions. This is not suboptimal if the system can be designed for a special purpose such as the considered biosensor. It can be assumed that the size and the format of the input data will not change after the deployment of the system. Therefore, the system can be designed in such a way that the application is optimized to run in an optimal way for a fixed input data format. Due to physical and optical reasons the input data format is a rectangle with a size of e.g. 300×1000 or 1000×300 pixels. There are two orientation formats of the input data: landscape and portrait. The choice of using these formats is motivated by the assumption that the optimal partitioning for a kernel will change significantly from a data input format to the other. Table 5.4 presents the runtimes for different *work-group* sizes for the two data formats. As can be noticed from the values of *Pattern Matching*, greater *work-group* sizes are not always advantageous. For example, the runtime for an image in the portrait format is 3.311 ms for a *work-group* size of $X = 8$ and $Y = 8$, whereas the runtime for the larger *work-group* size ($X = 16$, $Y = 8$) is 3.538 ms. For the same kernel it can be

	16 → 64 Cores	16 → 128 Cores
Ideal	4	8
Min	0.79	0.92
Max	5.92	9.31
Avg	2.88	3.43

Table 5.5: Kernel Speedup with Increasing Number of Cores

seen that the shortest runtime for the portrait format is 3.266 ms (*work-group* size: $X = 8$, $Y = 16$) respectively 3.311 ms (*work-group* size: $X = 8$, $Y = 8$) for the landscape format. It can be therefore concluded, that a GPGPU application must be optimized towards a certain input data set size.

5.5.3.3 Parallel Processing Scalability

As depicted in Table 5.5, for the ideal case (calculated with Amdahls Law $(1-f) = 0$) the speedup is 4 when switching from 16 to 64 cores and 8 for switching from 16 to 128 cores. Furthermore, Table 5.5 shows that the performance of the pipeline changes with number of cores, but that the ideal speedup can not be reached. For example, the increase from 16 to 64 core achieves an average performance speedup of 2.88 while a core increase from 16 to 128 has only a performance speedup of 3.43. For some kernels such as the *Pattern Matching*, these speedup factors can be reached by moving from 16 to 64 cores and also by moving from 16 to 128 cores. On the other hand, there are kernels with high data dependencies with no speedup at all. The superlinear speedup for some kernels are due to the higher memory speed and bandwidth of the 9600 GT and GTS 250. From Figure 5.26(a), it can be seen that especially kernels that implement the aggregation and classification of pixels to polygons have high data dependencies avoiding higher speedups. Figure 5.26(a) shows the accumulated runtime over the different steps of the image processing and analysis pipeline for the different graphics cards.

5.5.3.4 Energy/Runtime Considerations

As for the runtime, the energy consumption is now analyzed for different load partitions and for the scaling with different numbers of cores. Energy efficiency is defined in this context as the minimal energy consumption over all configurations which meet the real-time requirements. The power consumption over the time for the image processing and analysis pipeline is depicted in Figure 5.26(b). As can be seen in this figure, the ION graphics cards consumes less power (up to 15 W) in respect to the other two graphics cards (up to 65 W), but also needs far more processing time. The ION requires 60 ms for one frame with the optimal configuration while consuming 0.7 J (see Table 5.6). This exceeds the real-time requirement of 33.3 ms but the energy consumption is minimal. The graphics cards 9600 GT and 250 GTS have runtimes of approximately 18 ms and 17 ms per frame. Considering

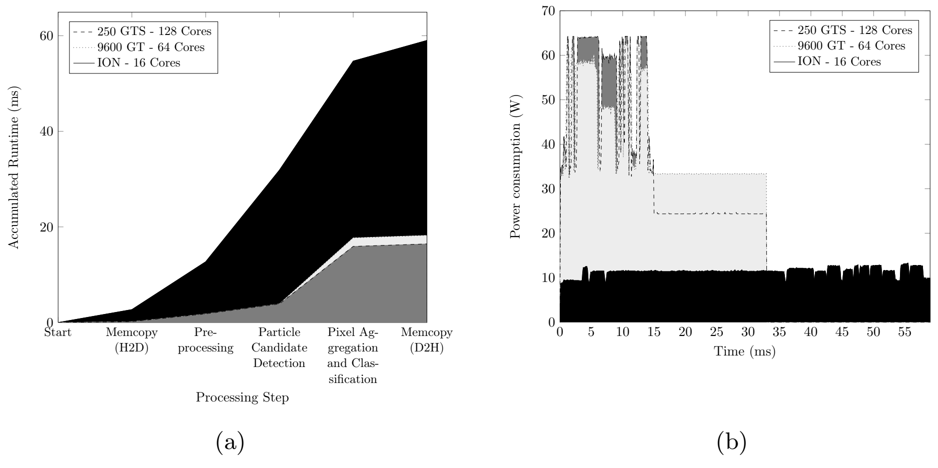


Figure 5.26: (a) Accumulated Runtime for Optimal Work Group Size and (b) Power Consumption for Optimal Work Group Size

the energy consumption $E^{P_h,P_l}_{T_0,T_{run}}$ for the actual processing of a frame the 9600GT needs only 0.95 J while the 250 GTS needs around 1.01 J. Since the GTS 250 has the lowest idle power consumption $E^{P_h,P_l}_{T_{run},T_{SLOT}}$ (see Table 5.6), the GTS 250 is the platform with the highest overall energy efficiency. Overall, it can be summarized that the design space exploration showed that the load balancing and the scalability of the application kernels have a direct impact on the energy efficiency of the application and should be considered at design time.

5.5.4 Summary

In the face of green computing and the utilization of GPGPU capable chips in small and mobile systems, it is mandatory to take the energy consumption as an objective during the design process of a GPGPU based system into account. In this section, a design space exploration was described which shows the design process towards an energy aware GPGPU-based image processing and virus detection system. The design space exploration exploited the parameters load balancing and the parallel processing scalability for a GPGPU application and showed that these parameters have a direct impact on the energy efficiency of the application and must be considered at design time. Furthermore, it was demonstrated that the minimization of the energy consumption under the restriction that only a limited amount of time is available for processing limits the design space for an energy efficient system configuration. Therefore, it can be concluded, that energy can be saved, due to a platform selection in the face of deadlines.

The design space exploration did not cover the decision if the integration of a GPU for accelerating is energy efficient at all. This is considered in the following Section 5.6.

Graphics Card	r_{min} (ms)	Minimal $E^{P_h,P_l}_{T_0,T_{run}}$ (J)	Minimal $E^{P_h,P_l}_{T_{run},T_{SLOT}}$ (J)	E_{min} (J)
ION	59.98	0.7	-	-
9600GT	18.16	0.95	0.39	1.34
250GTS	17.22	1.01	0.31	1.32

Table 5.6: Runtime and Energy Consumption for Optimal Work Group Size

5.6 Design Space Exploration for GPGPU-Accelerated Embedded Systems

In this section the second design space exploration for GPGPU applications presented in this thesis is introduced. After a short introduction in Section 5.6.1, on why the design space exploration is needed for an efficient GPGPU application design, basic techniques are presented in Section 5.6.2. In the scope of the overall design process depicted in Figure 5.27, this section (see Section 5.6) provides a mapping optimization for different platforms (GPU and CPU). The results for the design space exploration of several benchmarks are provided in Section 5.6.3. Section 5.6 ends with a summary in Section 5.6.4.

5.6.1 Introduction

Section 5.5 presented a design space exploration for the most efficient GPU variant platform with the objectives energy consumption and meeting deadlines. When talking about these objectives, it is not enough to take only the graphics card variant into account, e.g. it can be possible that executing the application on a CPU can be more efficient in comparison to a GPU execution. Especially when the graphics card is only integrated in the platform design to accelerate a parallel application, an analysis has to be carried out to determine if the system is still energy efficient (*Optimization Area 3*). The following factors should be taken into account for the system design:

1. *Additional Idle Power Consumption*: The graphics cards can be used as a parallel application accelerator in almost the same manner as traditional application accelerators like FPGAs. When the graphics card or chip is not integral part of the design, an extra graphics card or chip contributes to a higher power consumption.

2. *Additional Communication*: The availability of two main memories implies extra communication to transfer data. This creates extra overhead for communication and for the energy consumption.

3. *Algorithm Design*: It is only beneficial to use a graphics card for accelerating applications if the application itself can be parallelized efficiently. According

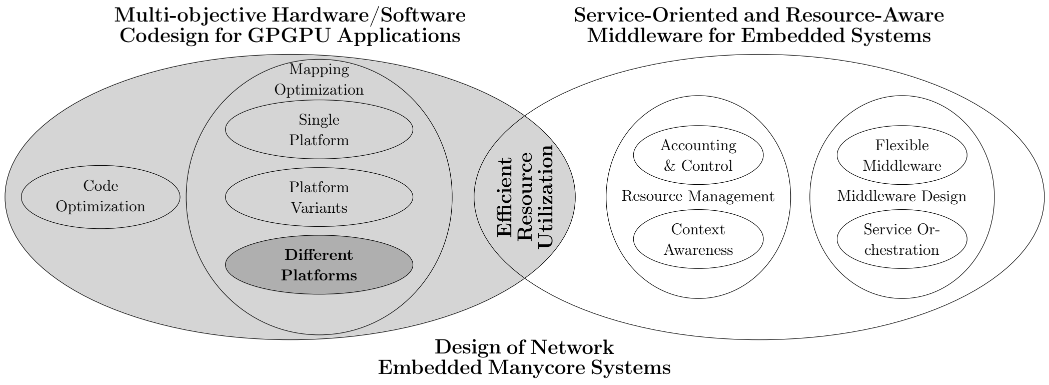

Multi-objective Hardware/Software Codesign for GPGPU Applications
Service-Oriented and Resource-Aware Middleware for Embedded Systems
Mapping Optimization
Single Platform
Platform Variants
Different Platforms
Code Optimization
Efficient Resource Utilization
Accounting & Control
Resource Management
Context Awareness
Flexible Middleware
Middleware Design
Service Orchestration
Design of Network Embedded Manycore Systems

Figure 5.27: CPU/GPU-DSE - Overview on the Overall Design Process

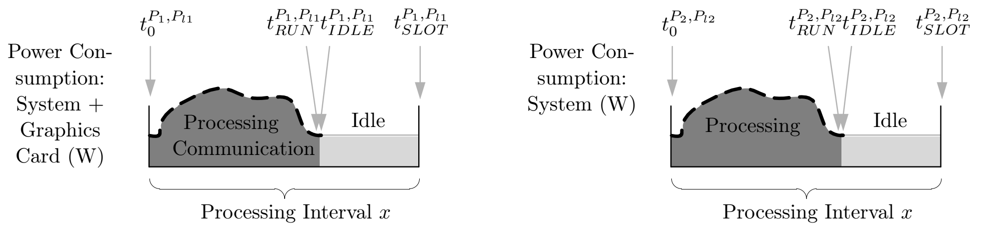
to Amdahl's Law, the higher the ratio of parallel code in an application is, the more the application can be speeded up by a parallel execution.

5.6.2 Materials and Methods

For a multi-platform (CPU and GPU) mapping optimization, one major decision is whether integrating an additional graphics card for the acceleration counteracts the objective to be energy efficient or not. Therefore, several system configurations must be evaluated with respect to their energy consumption: Systems without the graphics card for application acceleration and systems with an additional graphics cards for application acceleration. In Figures 5.28(a) and 5.28(b), the most important parameters for evaluation of the energy consumption are depicted.

For evaluating energy consumption and runtime, the profiling testbed presented in Chapter 3 is utilized. As already described in that chapter, the testbed can provide the energy consumption $energy(P, P_l, t_{start}^{P,P_l}, t_{end}^{P,P_l})$, where P is the program to be profiled, P_l the execution platform consuming power over time, t_{start}^{P,P_l} the profiling start point in time and t_{end}^{P,P_l} the profiling end point in time. For this DSE, the accelerated program running on the GPU is called P_{l1} and the execution platform comprising the GPU host system and the graphics card is called P_1. The system without a GPGPU-capable graphics card is called P_2 and the corresponding program P_{l2}. As can be seen from Figure 5.28(a) the testbed provides the points in time for the following triggers for the P_{l1} and P_1:

- $t_0^{P_1,P_{l1}}$: Start time for the process interval and task processing
- $t_{RUN}^{P_1,P_{l1}}$: End time for task processing plus processing between systems
- $t_{IDLE}^{P_1,P_{l1}}$: Start time for the idle phase



(a) Energy Consumption for System with Graphics Card (b) Energy Consumption for System without Graphics Card

Figure 5.28: Energy Measurements for the Different System Configurations

- $t_{SLOT}^{P_1,P_{l1}}$: End time for the process interval

and the following triggers for the P_{l2} and P_2:

- $t_0^{P_2,P_{l2}}$: Start time for the process interval and task processing
- $t_{RUN}^{P_2,P_{l2}}$: End time for task processing
- $t_{IDLE}^{P_2,P_{l2}}$: Start time for the idle phase
- $t_{SLOT}^{P_2,P_{l2}}$: End time for the process interval.

It is assumed that

$$x = t_{SLOT}^{P_1,P_{l1}} - t_0^{P_1,P_{l1}} = t_{SLOT}^{P_2,P_{l2}} - t_0^{P_2,P_{l2}}. \tag{5.9}$$

This means that both systems have the same deadline constraints. The energy consumption values for both systems can then be retrieved from the testbed as follows:

- $E_{PROC}^{P_1,P_{l1}} = energy(P_1, P_{l1}, t_0^{P_1,P_{l1}}, t_{RUN}^{P_1,P_{l1}})$: Energy consumption for task processing of the GPGPU program

- $E_{PROC}^{P_2,P_{l2}} = energy(P_2, P_{l2}, t_0^{P_2,P_{l2}}, t_{RUN}^{P_2,P_{l2}})$: Energy consumption for task processing of the CPU program

- $E_{IDLE}^{P_1,P_{l1}} = energy(P_1, P_{l1}, t_{IDLE}^{P_1,P_{l1}}, t_{SLOT}^{P_1,P_{l1}})$: Energy consumption for idle phase of the GPGPU program

- $E_{IDLE}^{P_2,P_{l2}} = energy(P_2, P_{l2}, t_{IDLE}^{P_2,P_{l2}}, t_{SLOT}^{P_2,P_{l2}})$: Energy consumption for idle phase of the CPU program.

Component	Configuration	Idle Power Consumption (W)
System	Intel Atom 270 2GB-DDR2-Memory Linux OS	10
Graphics Card	Nvidia 8400 GS 512MB-DDR-Memory	4

Table 5.7: System Configuration

In order to be energy efficient for a GPGPU system, one of the following inequalities must be true:

$$E_{PROC}^{P_1,P_{l1}} + E_{IDLE}^{P_1,P_{l1}} < E_{PROC}^{P_2,P_{l2}} + E_{IDLE}^{P_2,P_{l2}} \tag{5.10}$$

or

$$E_{PROC}^{P_1,P_{l1}} < E_{PROC}^{P_2,P_{l2}}. \tag{5.11}$$

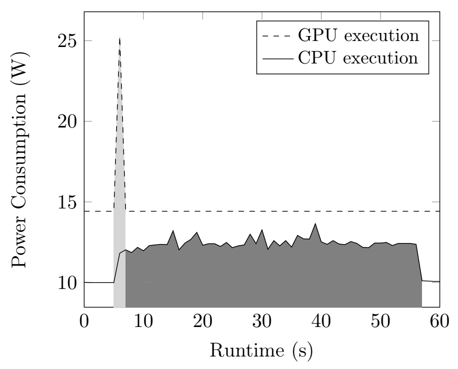
The values $E_{IDLE}^{P_1,P_{l1}}$ and $E_{IDLE}^{P_2,P_{l2}}$ are optional, if Equation 5.9 is not true and because of the following considerations. In a first case, it can be assumed that the processing is done in a periodic processing interval x and the timeframe is the same for the CPU and the GPU application. Then, one can compare the energy consumption values directly (Equation (5.10)). When the timeframe is not the same, a comparison is difficult because energy consumption is measured over time. But for an aperiodic processing task the Equation (5.11) can be utilized, when one would like to consider the energy consumption for a specific task.

5.6.3 Evaluation

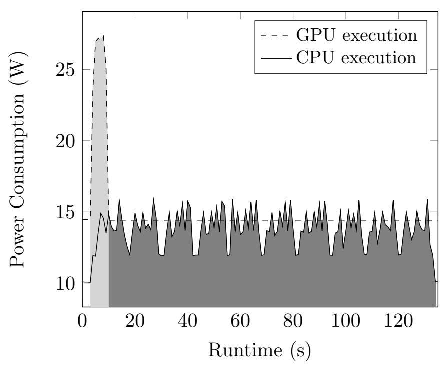
In this section, the evaluation of the several benchmarks towards their energy consumption on different systems is presented. Firstly, the configuration and parameters for the evaluation are given in Section 5.6.3.1 and secondly, the measurements will be presented in Section 5.6.3.2.

5.6.3.1 Parameters and Configuration

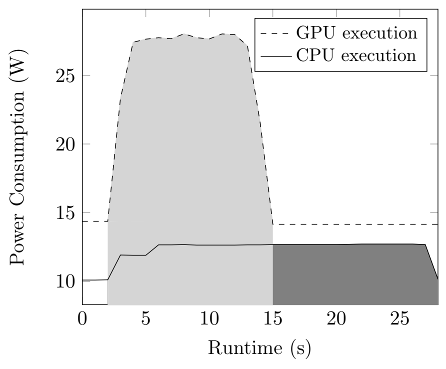
The GPGPU and CPU versions of four different benchmarks (see Section 2.4) have been tested in this evaluation. The system configuration was chosen as depicted in Table 5.7. The implementations of the benchmarks are optimized versions for the different platforms (CPU and GPU). Due to the lack of an embedded platform hosting the graphics card, the power consumption (Idle: 10 W; Load: 16 W) of the system presented in Table 5.7 was added to the graphics card's measurement, to approximate the power consumption of an embedded platform hosting a GPGPU-capable graphics card.



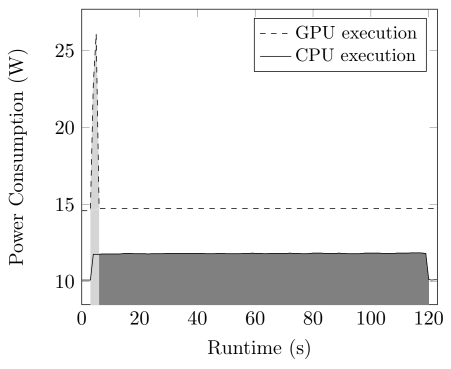
(a) Fast Fourier Transform



(b) Range-Doppler-Algorithm



(c) Matrix Multiplication



(d) Air Pollution Simulation

Figure 5.29: Power Consumption for CPU and GPGPU Versions

5.6.3.2 Results

The power consumption results for all four benchmarks are depicted in Figures 5.29(a)-5.29(d). The dashed lines in theses figures represent the power consumption for the GPGPU application and the black line the power consumption for running the CPU version of the benchmark. As can be noticed from these figures, executing the application on a GPUs is faster than executing the application on a CPU, but has a higher power consumption. As can been seen from Figures 5.29(a) -5.29(d), the maximal power consumption for the GPU execution is much higher than for the CPU execution, e.g. for the *Range-Doppler-Algorithm* the power consumption is up to 27 W. The CPU implementation has a power consumption of up to 16 W.

The light grey areas in these figures represent the energy consumption values $E_{PROC}^{P_1,P_{l1}}$ (GPGPU variant) which are listed in Table 5.8 for each benchmark. The dark grey areas represent the energy consumption values $E_{PROC}^{P_2,P_{l2}}$ (CPU variant) for each benchmark which are also listed in the aforementioned table. The idle power consumption of the system is assumed to be fixed depending on the measured energy

Benchmark	CPU Runtime (s)	GPU Runtime (s)	Speed up CPU $\rightarrow$ GPU	$E_{PROC}^{P_2,P_{l2}}$ (J)	$E_{PROC}^{P_1,P_{l1}}$ (J)	E_{red} (%)
Matrix Mult.	22.46	13.66	1.6	344	209	39
Air Simulation	117.32	0.89	132	1394	20	99
FFT	50.64	0.74	68	655	16	98
Range-Doppler	129.43	4.67	28	1833	108	94

Table 5.8: Runtime and Energy Consumption for Different System Configurations

consumption values presented in Table 5.7. The speedup factor (CPU $\rightarrow$ GPU) for this benchmark is 28 meaning the possibility to parallelize this benchmark is high. The same applies for the benchmarks *Fast Fourier Transform* and *Air Pollution Simulation* which can be speeded up by a factor of 68 and 132, respectively. The only benchmark where the runtime reduction is not significant, is the *Matrix Multiplication*. Only a speedup by a factor of 1.6 can be achieved. The reason for this is that the data dependencies within the benchmark prevent high parallelization and a higher speedup.

The decision if a system should be equipped with an additional graphics card as an application accelerator can only be made with the considerations made in Section 5.6.2. When idle phases are not considered, the energy reduction E_{red} can be calculated by

$$E_{red} = \frac{E_{PROC}^{P_1,P_{l1}}}{E_{PROC}^{P_2,P_{l2}}} * 100. \tag{5.12}$$

For the benchmark *Matrix Multiplication*, E_{red} is 39% and for the benchmark *Air Pollution Simulation* up to 99%. When idle phases are considered, the processing interval x must be the same for CPU and the GPU execution. For the benchmark *Range-Doppler-Algorithm*, the integration of an extra GPU for application acceleration makes only sense as long as: $108J + E_{IDLE}^{P_1,P_{l1}} < 1833J + E_{IDLE}^{P_2,P_{l2}}$, i.e. that the energy consumption reduction due to application acceleration must not be compensated by a long idle phase. In the idle phase energy can be saved by applying e.g. DVFS (Dynamic Voltage and Frequency Scaling) techniques. Tests have shown that changing the frequency and voltage level of a Nvidia GTS250 tooks about 160ms. This circumstance makes the DVFS technique unsuitable for applications with a short idle phase, such as the *PAMONO* sensor.

5.6.4 Summary

A design space exploration for the most efficient execution platform (CPU,GPU) for parallel applications with the objective energy consumption was presented. It was shown that integrating an additional graphics card for GPGPU can accelerate a parallel application and furthermore, that this does not necessarily counteracted the objective to be energy efficient. Several considerations were made how energy efficiency can be evaluated for application acceleration with GPUs and it has been shown that under certain circumstances, an energy reduction of 99% can be achieved even if the power consumption is much higher.

5.7 Conclusion

One of the major disadvantage of today's GPGPU programming is that it is dominated by manually performing code optimizations. The mapping optimization of computation kernels to graphics card cores is also done manually in order to achieve the optimal acceleration of an application. This is time-consuming and need not reveal the best possible acceleration.

In this chapter, the classical GPGPU application design process was analyzed and three optimization areas have been identified:

- *Optimization Area 1 - Mapping:* The manually chosen mapping to the processing cores should be substituted by an automatic approach.
- *Optimization Area 2 - Code Generation:* The code generation can be optimized towards pipeline load optimization and automatic mapping support.
- *Optimization Area 3 - Platform:* Only the most suitable platform should be used in the end, e.g. in terms of energy efficiency.

Based on these optimization areas, a *Multi-objective Hardware/Software-Codesign for GPGPU Applications* was proposed which includes profiling-based optimizations of the code and the mapping and comprises multi-objective optimization towards runtime efficiency and energy consumption efficiency. In the scope of this process, two code optimization techniques have been proposed and two design space explorations have been conducted.

The two code optimization techniques presented in this section are both based on instruction scheduling. The first approach, called *MOBLIS*, assigns scheduling policies to single basic blocks in the PTX representation of a kernel. It was possible to decrease the runtime by up to 14% and the energy consumption up to 15%. The evaluation was conducted on different hardware platforms. The second code optimization approach called *FALIS* is also based on instruction scheduling but is able to schedule instructions globally in a GPGPU kernel. In contrast to *MOBLIS*, it only focusses on memory-related instructions in the face of the memory wall. With *FALIS*, reductions of up to 10% in energy consumption and 13% in runtime could be achieved for real-world benchmarks. The novelties of the presented code optimizations can be summarized as follows:

- Adaptive instruction scheduling mechanisms have been developed which tune a GPGPU application towards a specific platform.
- Energy consumption and performance as optimization objectives for GPGPU applications inside a GPGPU code compiler have been taken into account.

The two mapping optimization techniques presented in this section, conducted design space explorations towards energy efficiency. The first DSE targets the decision, which platform is the most energy efficient for a GPGPU application under the constraint of deadlines. The second design space exploration targets the decision whether an integration of a GPGPU-capable graphics card for parallel application acceleration is beneficial to the energy consumption or not. The novelties of the presented mapping optimizations can be summarized as follows:

- It was shown that taking energy consumption and performance as optimization objectives for GPGPU applications within a design space exploration into account, is worthwhile for an efficient system design.
- The selection of the most energy-efficient execution platform for GPGPU applications was evaluated in two design space explorations.

Overall, it can be summarized that the energy consumption is an important target for optimization in the GPGPU application design process and should not be neglected. It is particularly important to develop special code and mapping optimizations targeted towards energy consumption in the face of more and more battery-driven mobile systems and with respect to green computing. From the GPGPU applications point of view – such as the *PAMONO* image processing and analysis application – the following can be concluded: The *Multi-objective Hardware/Software-Codesign for GPGPU Applications* is important to chose the most energy-efficient platform and to optimize the GPGPU application code as good as possible.

Chapter 6

Embedded System Middleware: Basics

In Chapter 5, methodologies related to *Multi-objective Hardware/Software Codesign for GPGPU Applications* were introduced. This chapter is the first one, which is related to *Service-Oriented and Resource-Aware Middleware for Embedded Systems* and in it basic technologies and paradigms in the context of service-oriented architectures will be provided. They are the bases for the methods to be presented in Chapter 7.

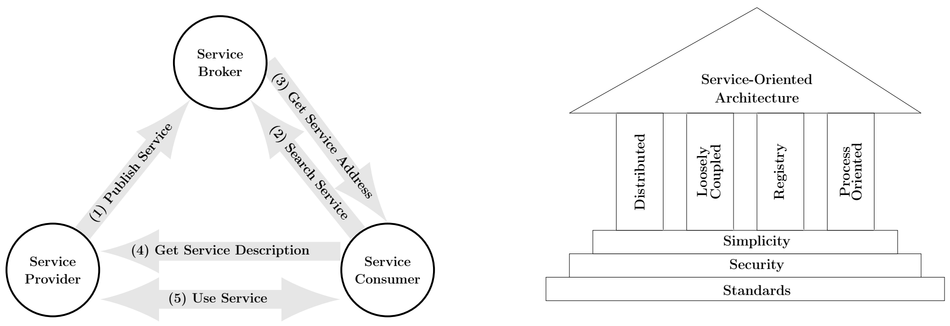
Contents

This chapter includes an introduction to SOA (Service-Oriented Architecture). Therefore, in Section 6.1, basic terminology with respect to SOA will be described. In Section 6.2, basic technologies such as web services and derived technologies will be presented.

6.1 Service-Oriented Architectures

The basic principle of SOA is the utilization of term *service* to denote the access to a certain functionality with a predefined interface. With regard to object orientation, this allows a higher level of abstraction because functionality is not bound to a certain object [86].

The typical workflow for SOA – called SOA triangle – is depicted in Figure 6.1(a). A service provider shares some functionality, which is published to a service broker or in a broadcast fashion into a network. The service consumer searches for some functionality at the service broker or in the network. It retrieves the address/location where it can find the requested functionality. Then the service consumer



(a) (b)

Figure 6.1: (a) SOA Triangle (modified [86]) and (b) SOA Temple (modified [86])

requests a service description and the description of the service interface from the service provider. Finally, the service consumer uses the functionality provided by the service provider.

The principles of SOA are summarized by the so-called SOA-Temple (see Figure 6.1(b)). The first principle of SOA is that functionality can be distributed, so it is not bounded to a place nor object. The second principle is that a service is only loosely coupled, meaning that there is only a connection between the service provider and the service consumer when requested. The third principle of SOA is that, due to the loosely coupled behaviour and the distribution of services, the service address/location must be available in some kind of registry. This can be done in a centralized way or in a distributed fashion. The last principle of SOA is that it is process-oriented. The three basements of SOA are the utilization of standards, a secure access of the service functionality and simplicity. [86]

6.2 Projects and Specifications based on SOA

Service-oriented architectures can be adopted to a wide range of different scenarios. Therefore, in this section several projects/concepts based on SOA will be described. At first, a software component framework called OSGi will be presented in Section 6.2.1, followed by Section 6.2.2, in which, an embedded web service framework is outlined. In Section 6.2.4, a service composition language standard will be described.

6.2.1 OSGi

OSGi [127] provides a platform for bundling software components and service-based interaction. It was originally not designed for distributed service-oriented architectures but towards an efficient local software management. The structure of OSGi follows a layered approach as depicted in Figure 6.2. The central concept of OSGi is

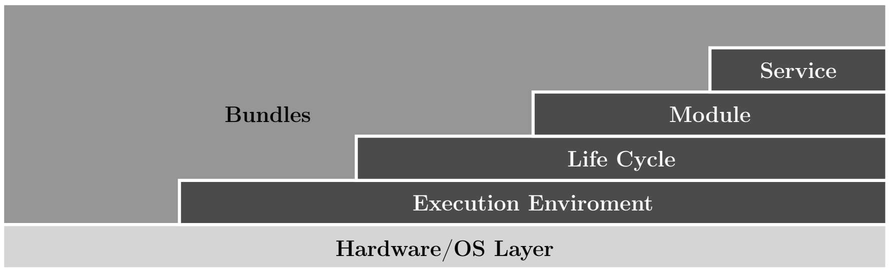


Figure 6.2: OSGi Layers [127]

that of a bundle which is a software container including executables, a description of the included functionality and descriptions of the dependencies to other bundles. The management of these bundles is done by the OSGi framework along with the management of the dependencies of the services. Services provide the actual functionality of a bundle. A local service registry exists. OSGi supports a service life-cycle management by providing functionalities such as installing, starting, stopping and uninstalling. The module layer defines interfaces and access rules for services and bundles. The execution framework layer is responsible for executing bundles on a pre-defined execution environment, e.g. some bundles can only run on a certain platform because only for this platform native libraries are available.

A reference implementation of the OSGi framework in Java exists. Nowadays, the usage of OSGi becomes more and more interesting for the embedded and cyber-physical systems world [108].

6.2.2 Device Profile for Web Services

The most utilized instance of service-oriented architectures are WS (Web Services) [86]. Basic technologies of WS are standard internet protocols such as TCP [106], UDP [105] and HTTP [51]. In addition, web service communication is based on XML [28] which is the standard for structuring information. The standard communication is done via the SOAP protocol [27] which utilizes XML. A WS interface is described by a description language WSDL [37]. A standard registry protocol for web services is UDDI [13], but it is not necessarily needed when creating a SOA with web services.

DPWS (Device Profile for Web Services) [31] is a set of web service specifications tailored towards the use in field of embedded and cyber-physical systems. An exemplary protocol stack of DPWS is illustrated in Figure 6.3(a). In contrast to original web services, DPWS services are bound to a particular DPWS device where DPWS services are running on (depicted in Figure 6.3(b)). The DPWS device itself is called *hosting service* providing meta-data about the *hosted services*. These *hosted services* implement the application functionality. As can be seen in Figure 6.3(b), DPWS

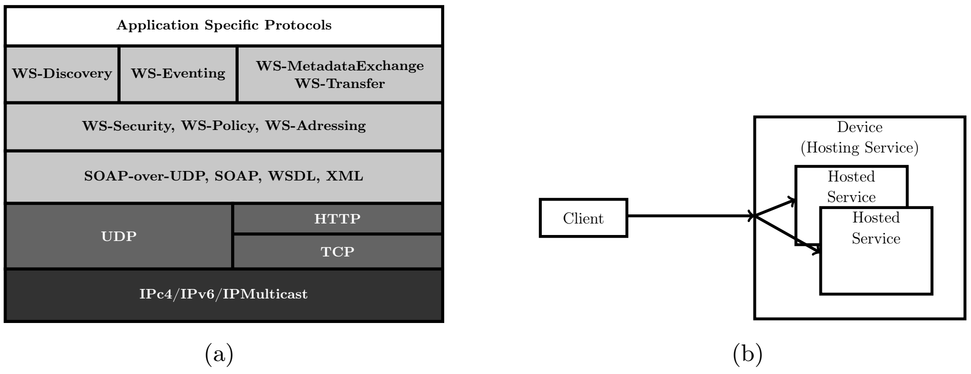


Figure 6.3: (a) DPWS Stack [156] and (b) Interaction between Clients and Devices (modified [31])

follows a client/server approach. The client searches for certain DPWS devices like printers in the network. On a particular discovered printer, several services are located like "printing a document" or "providing the level of the printers' ink". The client can then choose one of the services and invoke the functionality it wants.

6.2.2.1 Basic Features

In this section two basic features of web services are described: SOAP [27] and WSDL [37]. SOAP protocol is the standard communication protocol and WSDL is the description language for describing a service interface.

SOAP: A basic internet protocol which is utilized for web services is SOAP. It is used for providing a communication protocol to exchange information based on XML. An XML message can be transferred over arbitrary underlying network protocols. The atomic entity of this protocol is a SOAP message as listed in Listing 6.1. A SOAP message includes a SOAP envelope, which comprises a SOAP header with information for and about the receiver of the message and information about the sender. The SOAP body includes the information to be transmitted itself.

```
<SOAP–ENV:Envelope ..>
  <SOAP–ENV:Header>
  </SOAP–ENV:Header>
  <SOAP–ENV:Body>
  </SOAP–ENV:Body>
</SOAP–ENV:Envelope>
```

Listing 6.1: Structure of the SOAP Envelope

WSDL: WSDL (Web Service Description Language) is the basic technology for describing a web service interface. The UML schema of a WSDL file is shown in Figure 6.4. A WSDL comprises six different kinds of elements: *types*, *message*,

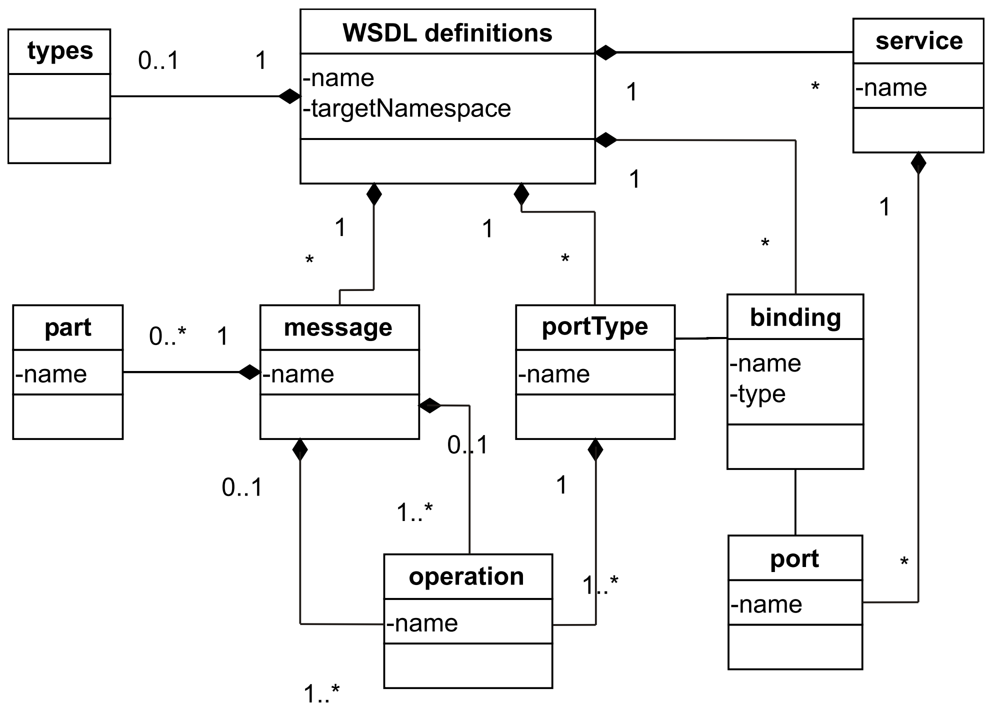


Figure 6.4: WSDL Scheme (modified [39])

portType, *binding*, *port* and *service*. *types* define the data types for the *messages*. The different parts of an abstract message are defined with the element *message*. An abstract operation with incoming and outgoing *messages* is specified with *portType*. The concrete assignment of protocols to a particular *portType* is done with a *binding* – especially for the defined operations and messages. The *port* defines an address of a *binding* and a *service* element aggregates several ports.

6.2.2.2 Protocols

The specification and protocol stack of DPWS [156] is depicted in Figure 6.3(a). Its specialities include the replacement of the service registry mechanism with an ad-hoc search protocol and the introduction of a publish/subscribe protocol. The most important specifications are as follows:

- WS-Discovery [88]: Advertise devices and services in the network and receive publications from other devices joining the network. It is implemented with the help of the SOAP-over-UDP protocol and utilizes multicast to publish and listen on discovery messages.

- WS-Addressing [26]: WS-Addressing specifies an addressing scheme for web services containing additional information for the SOAP header such as a unique identifier for a web service.

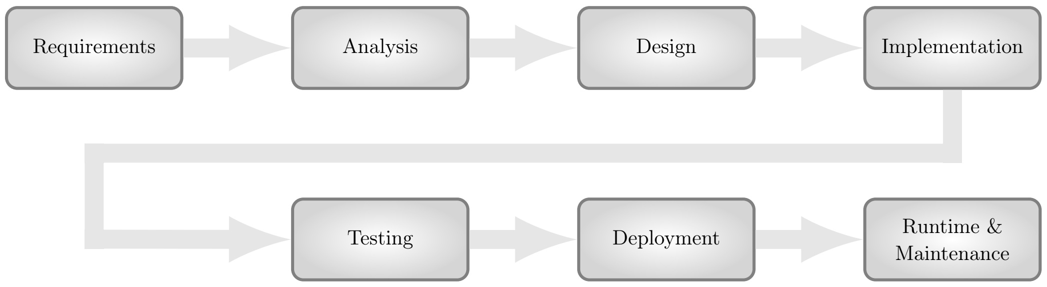


Figure 6.5: Web Service Life Cycle Phases (modified [64, 77])

- WS-MetadataExchange [41]: Web services specification for WSDL data and other metadata transfer is stated in the WS-MetadataExchange specification. Other metadata, except WSDL can comprise the device description or other service related metadata.
- WS-Eventing [25]: In the WS-Eventing specification a publish/subscribe protocol for web services is introduced. This is highly important in order to employ DPWS in the automation sector where sensors and actuators are communicating with each other. With respect to polling-based communication scheme, publish/subscribe has the advantage of utilizing communication bandwidth only when communication is needed at all.
- WS-Security [92]: A security protocol for web services in the DPWS context is specified in WS-Security. WS-Security is especially important when communication is done over a public wide area network link.

6.2.3 Classical Web Service Development Process

The WS (Web Service) implementation methodology for service-oriented architectures follows a multi-phases approach employing an agile software development work flow [64]. The overall web service implementation methodology workflow is depicted in Figure 6.5. The workflow starts with recording the user and business requirements. These requirements comprise the actual needs of the user, but also the demands for non-functional and functional requirements to the overall infrastructure. The second phase is the analysis of the requirements, meaning that functional components and non-functional requirements for these components are specified in formal way. The third phase of the web service development process is the design phase, where the design of the interfaces by specifying e.g. the data types is done. In addition to that, the interaction between the different web services and clients is layouted. The services are implemented along with the WSDL (Web Services Description Language) descriptions in the implementation phase. This phase includes porting and wrapping of already available software components to the current application. In phase five, the functionality of single services and non-functional requirements to a single service are tested. The deployment phase comprises service deployment and the configuration to the actual platform. Furthermore, post-deployment tests are conducted and the registration to a web service registry takes place. The last phase is the runtime and maintenance phase. In this phase, the system can e.g. be redefined to extra requirements which arise after deployment. Also errors can be fixed or new features can be installed. [64]

6.2.4 Web Service Business Process Execution Language

Sequential invocations of several web services and handling their response can be implemented by an application designer manually. Service orchestration languages try to ease the automatic composition of different web services. WS-BPEL (Web Services Business Process Execution Language) [8] is a description language for business processes which imports the functionalities of a set of other web services and exports their functionality as a new service. One objective of WS-BPEL is a formal description for the exchange of messages between the different participating services.

A WS-BPEL description file contains an activity. This activity specifies business processes which interact with its participating web services. An executable business process can be e.g. hosted in a WS-BPEL engine or can be statically compiled. Besides this main activity a WS-BPEL description file can contain a set of participating web services (*<partnerLinks>*), variables to store internal states or to bind messages to them, handlers for events/faults or a correlation set for identifying the instance of an addressed web service. Important concepts of an activity comprise e.g. the creation of a new web service (*<receive>* and *<reply>*). The processing of the messages can be sequential (*<sequence>*) and parallel (*<flow>*). Dependencies between different web services are modelled as links (*<links>*) consisting of targets (*<targets>*), sources (*<source>*) and the corresponding variables which are bound to the messages of the web services. An WS-BPEL activity can include standard structures from programming languages such as *if-then-else*, *while* and *for*.

Chapter 7

Service-Oriented and Resource-Aware Middleware for Embedded Systems

The scope of this chapter targets the *Service-Oriented and Resource-Aware Middleware for Embedded Systems.* In it, a distributed application design process with the help of SOA will be described and methods to optimize it for utilization in the embedded and cyber-physical system domain will be presented. In Chapter 6 the fundamentals for efficient middleware design to be presented in this chapter were provided. Additionally, the energy consumption and performance testbed presented in Chapter 3 is utilized by the methods in this chapter for profiling the resource utilization. Furthermore, the principles of web services and DPWS are the basic techniques enhanced by the methods to be introduced. This chapter is then followed by the conclusion chapter of this thesis.

Contents

7.1 Introduction

In the automation domain, e.g. in the automotive sector and in the home automation sector, SANETs (Sensor/Actuator NETworks) are reality [107] and trends towards the use of SOA (Service-Oriented Architectures) have been started [42]. The shift towards SOA was necessary due to the increased complexity of system software and the need to re-use of software components in the face of shorter and shorter time to market cycles.

As described in Section 1.1, the SOCRADES roadmap [124] summarizes several challenges for the utilization of service-oriented architectures, such as high-level communication libraries, service orchestration mechanisms and context-aware services. These challenges will be addressed in the chapter by providing an efficient web service based middleware. At first, the optimization potential in the classical web service development process is evaluated (see Section 7.1.1) and then the different novelties which will be presented in the chapter are sketched (see Section 7.1.2).

7.1.1 Optimization Potential in Classical Web Service Development Process

From the field of embedded and cyber-physical systems several requirements arise when using SOA and web services [124]. The major points of refinement to the classical web service development process presented in Section 6.2.3 comprise the following phases: design, deployment and runtime. In the design phase, it is mandatory to take into account the execution platform and environment of a service. For example it has to be designed in which environment/platform a service should run, e.g. not all features of classical web services are available. The deployment phase should be redefined to work with ad-hoc deployment of services which means the services are encapsulated in a component bundle and a services is made publicly available in an ad-hoc environment without the use of a central registry. The runtime phase is the last refinement point. This phase must support context awareness of services and the platform in order to cope with the highly dynamic environment in sensor/actuator networks.

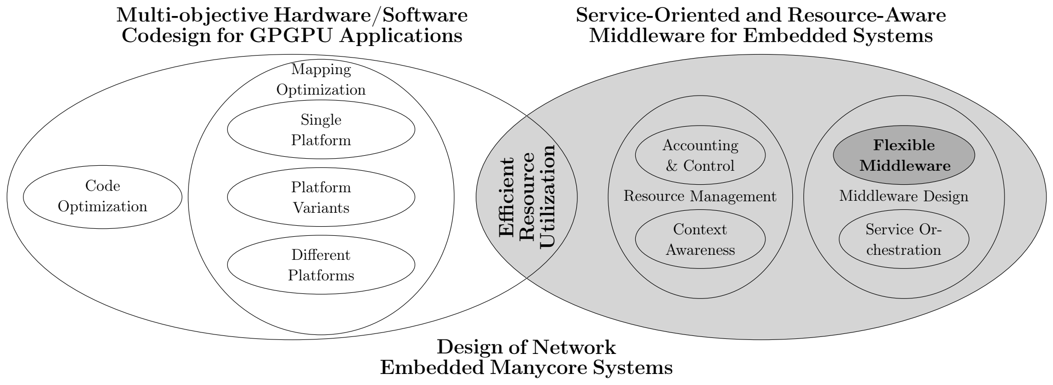

Multi-objective Hardware/Software Codesign for GPGPU Applications
Service-Oriented and Resource-Aware Middleware for Embedded Systems
Mapping Optimization
Single Platform
Platform Variants
Different Platforms
Code Optimization
Efficient Resource Utilization
Accounting & Control
Resource Management
Context Awareness
Flexible Middleware
Middleware Design
Service Orchestration
Design of Network Embedded Manycore Systems

Figure 7.1: *MORE* - Overview on the Overall Design Process

7.1.2 Flexible Middleware Techniques and Resource Awareness

Based on the considerations done in the former section of this chapter and in Section 1.1, several new approaches for creating sensor/actuator networks with heterogeneous devices will be introduced. In Sections 7.2 and 7.3, a highly flexible communication middleware for embedded and cyber-physical systems is described which includes a novel design and deployment process for web services. In addition to that, a new service orchestration technique will be presented. In Section 7.4 techniques for an efficient and context-aware resource management are introduced which enable single platforms in a web service environment to adapt the resource utilization to a more efficient configuration.

7.2 Embedded System Middleware Architecture

In this chapter, the fundamental concepts of *MORE* (Network-centric Middleware for GrOup communication and Resource Sharing across Heterogeneous Embedded Systems) will be described. In the scope of the *Design of Network Embedded Many-Core Systems*, it provides a flexible middleware design (illustrated in Figure 7.1). Firstly, the overall concept of *MORE* will be introduced in Section 7.2.1 and then, in Section 7.2.2 related work will be introduced. Afterwards, in Section 7.2.3 the architecture of *MORE* will be provided. Two use case scenarios for *MORE* will be evaluated in Section 7.2.4, followed by the summary of this chapter. *MORE* provides the basic methods for the techniques which will be presented in Sections 7.3 and 7.4.

7.2.1 Introduction

MORE was designed in the scope of the *MORE* project [4, 5, 75, 133, 87, 116, 53, 153] and is based on a combined utilization of two different paradigms (as depicted in Figure 7.2) [4]: *Service-Oriented Architectures* and a *Multi-Layer Software De-*

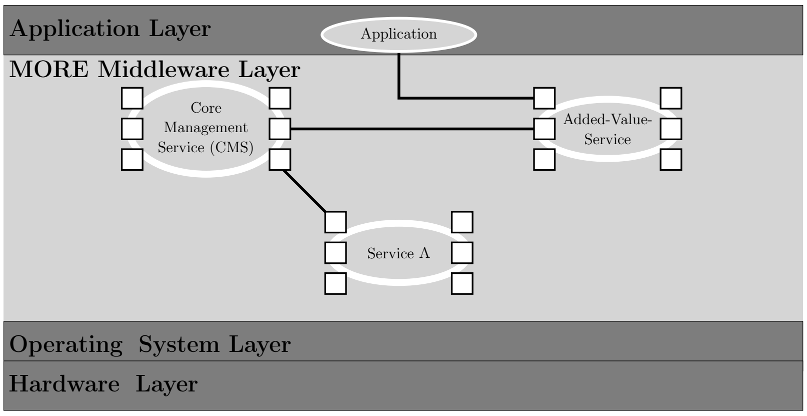


Figure 7.2: *MORE* Concept (modified [4, 153])

sign Approach. While the former is mandatory to achieve a high flexibility and the integration of heterogeneous devices, the latter is demanded to achieve high efficiency in terms of the runtime environment, hardware-dependent feature usage and the utilization of resources. The *MORE* layers are divided into *Application Layer*, *Middleware Layer*, *Operation System Layer* and *Hardware Layer*. *MORE Services* and application are represented as eclipses.

Derived from the challenges of the SOCRADS roadmap [124], the following important requirements should be handled in *MORE* [4]:

- *Capability Advertisements:* Each device in an application scenario should provide its hardware capabilities to other devices in the network. This is important in order to decide, whether certain computationally expensive services can be located on a node or not.
- *Publish/Subscribe:* A polling-based communication approach is not efficient to network types considered in *MORE*. Especially, when energy and bandwidth are scarce, unneeded communication should be avoided in terms of efficiency.
- *Connector Concept: MORE* should provide a unified interface for different types of connectors. This is mandatory when being used in combined local area and wide area networks. Connectors in the *MORE* context are illustrated as the six tiny rectangles at the edge of services depicted as ellipses (see Figure 7.2). The three rectangles of the left side denote incoming connectors and the right side are outgoing connectors.
- *Runtime (Re-)configurability:* Application scenarios and user requirements can change over time. In order to cope with these changes, *MORE* should provide mechanisms which allow to adapt to a new situation or context.

Furthermore, *MORE* should provide a middleware that allows interaction between heterogeneous embedded and cyber-physical systems. *MORE* follows a three-tier

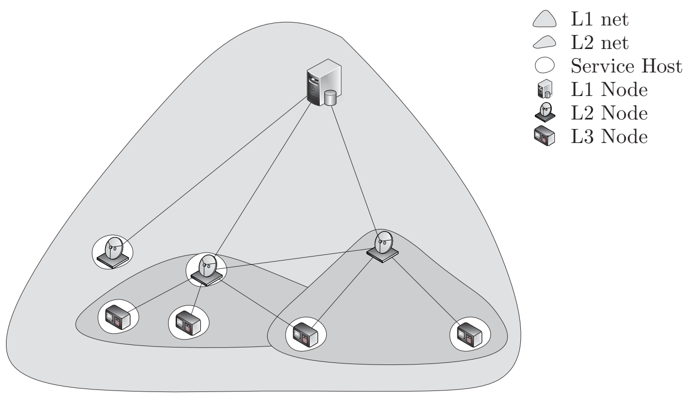


Figure 7.3: *MORE* - Network Structure (modified [4, 153])

approach by distinguishing between different types of hardware classes and network connection types [4, 153]. An overview on the *MORE* network and system structure is given in Figure 7.3. As can be seen in this figure, the following nodes classes exist:

- *L1 Nodes:* L1 nodes are server level devices comprising full-blown *MORE* service engines and possibly web service registries such as UDDI. The nodes in this class comprise wide area communication links (denoted as L1 net).

- *L2 Nodes:* Embedded systems or personal computer devices which also comprise full-blown *MORE* service engines are in this class. They include often several link connections such as wide area communication links (e.g. UMTS or GSM) but also local area network communication links (e.g. IEEE 802.11 (WiFi) or IEEE 802.15 (ZigBee)). Because of the capability to bridge between these different network areas, they are also called *MORE Gateway*.

- *L3 Nodes:* L3 nodes are typical cyber-physical systems. They only include local communication links and they are mostly directly connected to a L2 node. L3 nodes only comprise a subset of the *MORE* functionalities and software stacks. L2 and L3 nodes are included in L2 nets as illustrated in Figure 7.3.

There can also be devices in a *MORE* application scenario which host no *MORE Services*. As *MORE* is based on web service technology, classical WS and external middleware components based on WS, can also be included.

7.2.2 Related Work

Several middleware architectures for embedded and cyber-physical systems exist. An excerpt of current middleware architectures based on SOA will be presented in this section will. They will be explained in addition to the state-of-the-art technologies presented in Chapter 6.

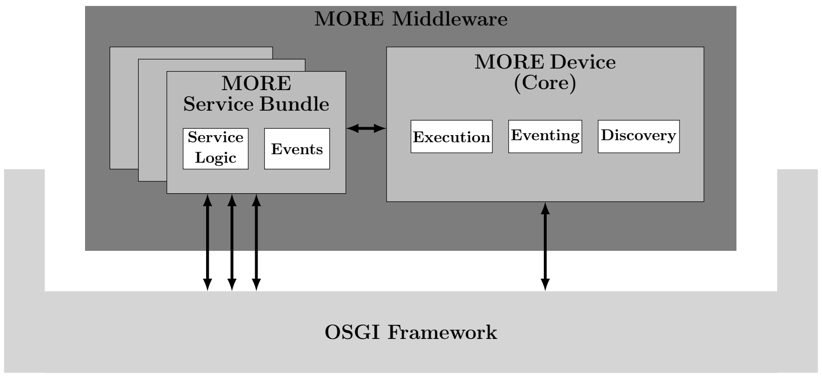


Figure 7.4: *MORE* Architecture [116]

PLASTIC [9, 115] is a project targeted towards an efficient communication middleware for Beyond3G networks. It is based on a two layered approach. On the lower layer, PLASTIC comprises a communication middleware, employing a web service based communication and a P2P routing mechanism. On the upper layer, PLASTIC provides several generic services such as service discovery, context awareness management and security services. However, PLASTIC is not targeted towards sensor/actuator networks like MORE. In comparison to *MORE*, PLASTIC has a focus on high-level context awareness and security. The focus of *MORE* is towards group communication and resource management.

The Amigo [142] project introduces the semantic web to networked home environments. As PLASTIC, it comprises security and context awareness but from the perspective of semantic web services. Amigo allows to seamlessly integrate local and remote services with the help of a component framework. Several reference implementations with OSGi and Microsoft .Net exist. Another basic technology of Amigo is the utilizations of ontologies to create services and service compositions automatically based on semantic information.

Several projects target the integration of OSGi and DPWS for a combined framework. Two of them are described in [50] and [22, 23]. In [22], the ANSO project is described which developed the DPWS Discovery Base Driver [23]. The DPWS Discovery Base Driver enables an OSGi framework to interact with remote DPWS services and extends OSGi with a registry including remote services. A similar technique was proposed in [50]. By utilizing these registry extensions it is possible to use remote and local services in a transparent way, meaning that the service consumer interacts with remote services in the way as for local services. In contrast to these projects, *MORE* uses OSGi only for bundling software components and for the on-the-fly service deployment.

7.2.3 *MORE* - Materials and Methods

The basic component of *MORE* is the *MORE Core*. This *MORE Core* is running – with some exceptions – on all devices in an application scenario realized with *MORE*. Those devices are called *MORE Devices*. The services specified in *MORE* (short: *MORE Service*) can be accessed via a unified interface which hides different connector types. Connector types comprise bindings for internal (local) communication as well as SOAP and *μSOA* [153] bindings for external (remote) communication. The *MORE* design workflow distinguishes two types of services, application-related services, which have to be written or ported to a particular application and *added-value-services* [4, 5, 75, 133, 87, 116, 53, 153] which offer common functionality like data, group, and resource management services. These *added-value-services* can be selected for certain applications and may be loaded to a *MORE Device* on demand. This is especially important to provide a middleware with a small footprint. The *added-value-services* can be distinguished in different groups:

- Communication Services: Efficient communication is mandatory in mobile and networked environments. Therefore, different communication services can be provided, e.g. message prioritisation services, routing services etc.

- Group Services: Group communication is one of the major aspects in *MORE*. Features of these services include group management and sending/receiving messages to groups in a multicast fashion. For group services a policy-based approach was developed [54].

- Security Services: Security services should provide different mechanisms for secure message communication.

- Data Services: Central or distributed data stores are provided by services in this domain.

- Resource Management Services: Not only the services themselves must be efficient but also the execution on the different platforms. The adaptation of the platform to a service and the adaptation of the service to a platform is the domain of the resource management.

The architecture of *MORE* is depicted in Figures 7.4 and 7.5. The *MORE Core* is the fundamental component of the *MORE* stack. Without it no other *MORE Service* can run, be deployed or communicate. The underlying software structure framework is OSGi which starts the *MORE* core at nodes' startup. Each *MORE Service* is encapsulated by one OSGi bundle (as depicted in Figure 7.4). When starting a *MORE Services* bundle in the OSGi framework, it automatically registers to the *MORE Core* and is being made ready for external and internal communication.

The communication protocols used in *MORE* are based on DPWS specifications [31].

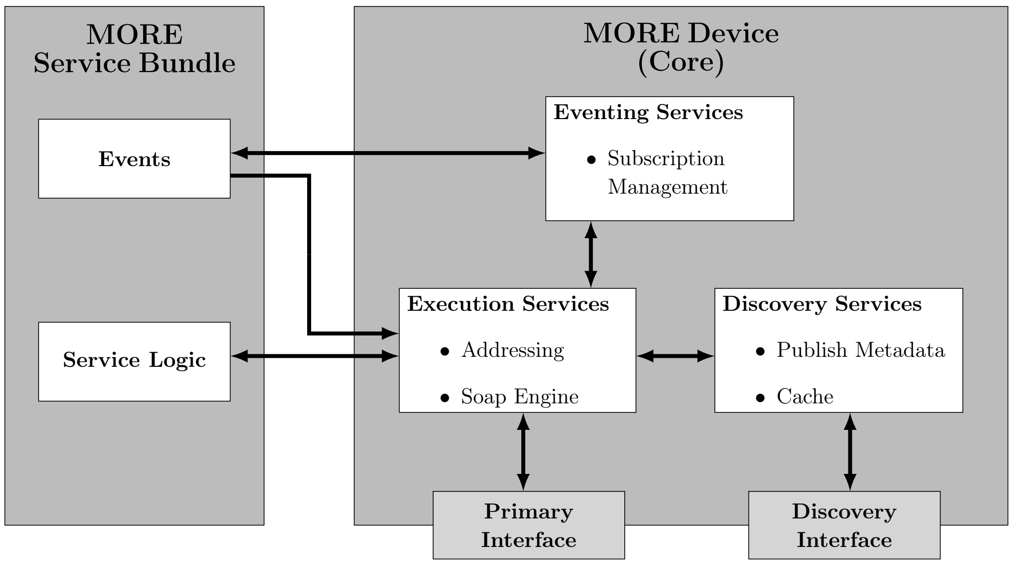


Figure 7.5: DPWS [70] in *MORE* (modified [116])

The interaction between the *MORE Core* and the *MORE Services* is shown in Figure 7.5 in detail. According to the DPWS specification two types of services exist: Hosting services and hosted services. Hosting service are the basic services running on a device. In *MORE*, these hosting services are part of the *MORE Core* and of the *MORE Core* bundle (as depicted in Figures 7.4 and 7.5). The other type services are the hosted services. In *MORE*, these services are the application services or the *added-value-services*. The *MORE Core* utilizes WS-Discovery for publishing/receiving devices and services in/from the network. Additionally, the *MORE Core* provides the access to the metadata and WSDL definitions of other hosted services located on remote devices. The eventing and notification services (WS-Eventing) comprise a publish/subscribe protocol, while the execution services offer the functionality for service invocation.

7.2.3.1 Service Development and Deployment

All *MORE Services* are based on the *MORE Service* development process (as depicted in Figure 7.6). Developing a *MORE Service* starts by defining its WSDL description which includes information about available functionality, data types used as parameters and return values. The application designer can focus on the service functionality itself, as stub and skeleton classes are generated from WSDL descriptions. *MORE Service* bundles are installed by deploying them to a local directory – available on each *MORE Device* – where they are automatically detected and made available to the *MORE Core*. Service updates are done in this way also. As soon as MORE services are registered to the *MORE Core*, they are able to be discovered, to

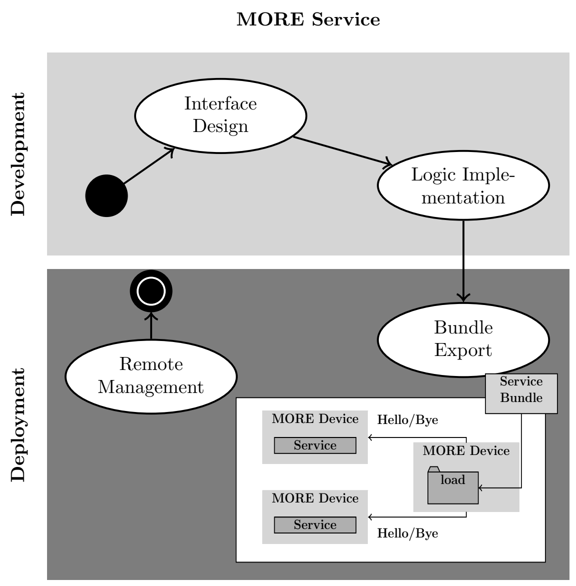


Figure 7.6: *MORE Service* Development and Deployment Cycle (modified [134])

exchange metadata, and to be invoked locally or remotely. This approach simplifies the process of service management on embedded devices.

7.2.4 Use Case Evaluation

In this section, two use case scenarios will be shown which can be solved by the multi-tiered network approach of *MORE*. Firstly, in Section 7.2.4.1, a scientific use case will be presented and then a use case scenario from intra logistics will be outlined in Section 7.2.4.2. The use cases themselves were already described in Section 2. In this section, a realization of these use cases with the help of *MORE* is explained.

7.2.4.1 Scientific Sensor Network Scenario

The original idea of *MORE* was to support scientific application scenarios employing sensor networks. An example application scenario was outlined with *MORE* [57] which will be presented in the following – according to the description in [116]. The use case scenario was entitled with the term *Mitigation Management*. It is described as follows: A number of sensor nodes are distributed at remote locations in the forest. The sensors of the sensor nodes include temperature, moisture and gas sensors. In former times, the data from these sensors were transferred manually to scientists to evaluate this data. The new architecture of this scenario with *MORE* is depicted in Figure 7.7. As can be seen in this figure, a central data server exists having L1 node characteristics. The data on this central data server is provided to end user devices such as smart phones and personal computers by one or more *MORE Services*, either by providing raw data or processed data. The second purpose of the

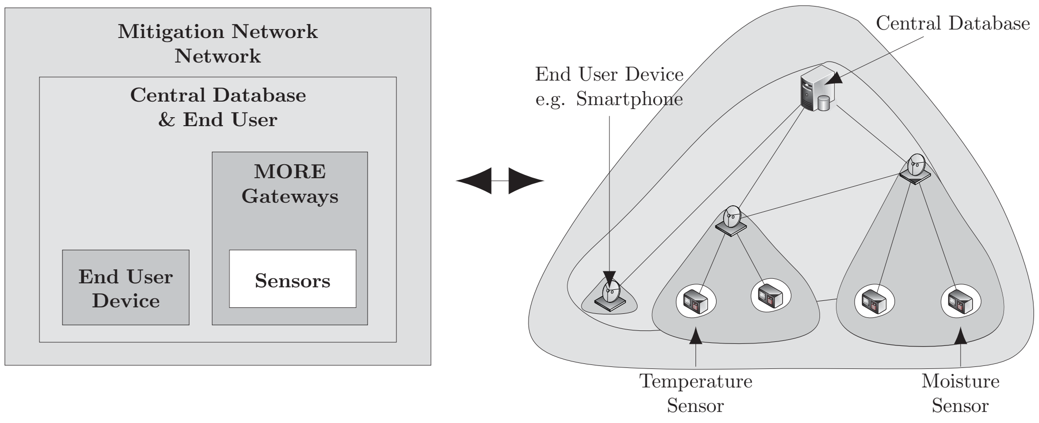


Figure 7.7: Network Structure within the Mitigation Management Scenario (modified [63])

data server is to act as a data sink for the sensor data. Therefore, the data server subscribes to the *MORE Gateways* which are located at remote sites. They have the hardware characteristics of L2 nodes. Each of them comprises one or more sensor nodes and a wide area communication link, e.g. UMTS or GSM. *MORE Gateways* publish in a certain interval the sensor data to the central database. An add-on feature was to provide *MORE Gateways* with extra logic, such as the possibility to react to extreme sensor data directly. For example, a *MORE Gateway* can create a communication group in case of high temperature and low moisture values to inform forest departments or fire brigades about a high risk of forest fires.

7.2.4.2 Intra-Logistics Scenario

Traditional conveyor belt systems comprise central control mechanisms such as PLCs (Programmable Logic Controllers). They administrate a large variety of sensors and actuators. In this section, an exemplary SANET for a conveyor belt system from the field of automated facility logistics systems is the considered use case as described in Section 2.2. This SANET controls a conveyor belt system (depicted in Figure 7.8) comprising different sensor nodes and actuator nodes, directly connected with *MORE*. This is fundamentally different, but more efficient, compared to traditional systems where there is only a central controlling instance. The routing of the bins or parcels in that conveyor belt system is done by identifying a QR code [44] on the parcel by a camera system and by requiring routes for that QR code from a central database. The switches of the conveyor belt systems are controlled by the camera systems. The camera systems are called VSU (Visual System Units). The main task of VSUs is to observe the conveyor belt system for bins or parcels and to route them according to centrally stored routing information. In addition to that each VSU subscribes to the outputs of predecessor VSU to look forward to

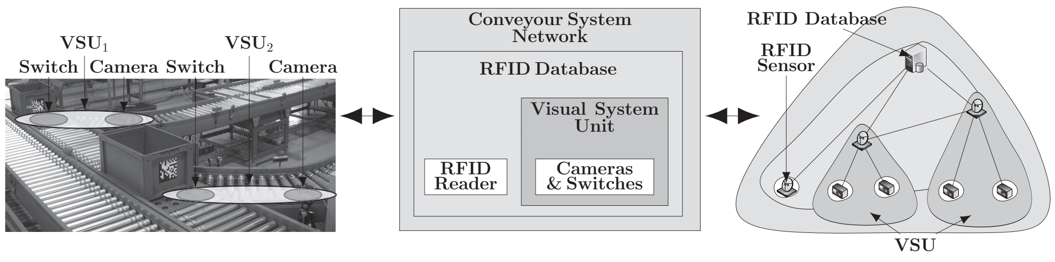


Figure 7.8: Network Structure within the Conveyor System

bins routed in its direction.

The considered conveyor system as depicted in Figure 7.8 is a typical hierarchically structured SANET. At the topmost level, large-sized computer systems are used which run e.g. the RFID/QR code central database. They have L1 node characteristics, i.e. they are equipped with full featured web services and web service orchestration engines. At this level no resource constraints aside from economical or environment protection reasons exist. At a lower level (depicted in Figure 7.8 at left-hand side) components of the sensor network such as VSU are used to aggregate and forward information. They are connected among each other, to the controlled switch and to the central RFID database. These systems are equipped with standardized local-area communication interfaces to allow a communication with the sensor/actuator from outside. They have L2 node characteristics, i.e. the resources of these systems are constrained in processing and memory capabilities and they often have energy constraints. In order to cope with these constraints, software running on these systems have to be developed by keeping them in mind. For example, not all features of topmost level systems can be deployed. Systems of the lowest level are highly resource constrained. These systems are the sensor/actuator nodes like the switches with minimal processing power and only limited communication capabilities. These systems have L3 node characteristics.

7.2.5 Summary

In this section a publish/subscribe based middleware was introduced which was aimed at the provision of an efficient and flexible service-oriented architecture approach for heterogeneous (embedded) systems. It is targeted towards the use of service-oriented architectures in highly dynamic application scenarios with hardware platforms reaching from full-blown servers, personal computers down to embedded and cyber-physical systems. *MORE* is able to provide efficient machine-to-machine and machine-to-human facilities and ease the development and deployment of distributed applications. For the latter, *MORE* provides techniques such as over-the-air deployment and updates.

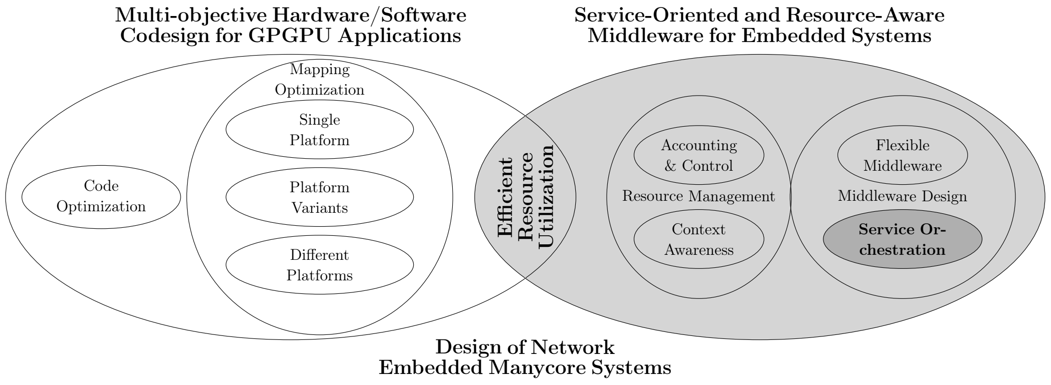

Multi-objective Hardware/Software Codesign for GPGPU Applications
Service-Oriented and Resource-Aware Middleware for Embedded Systems
Mapping Optimization
Single Platform
Platform Variants
Different Platforms
Code Optimization
Efficient Resource Utilization
Accounting & Control
Resource Management
Context Awareness
Flexible Middleware
Middleware Design
Service Orchestration
Design of Network Embedded Manycore Systems

Figure 7.9: *MSC* - Overview on the Overall Design Process

In the following sections, the *MORE* design is extended by methods for efficient service composition (see Section 7.3) and resource management (see Section 7.4).

7.3 Lightweight Service Orchestration

In this chapter, a service orchestration method for *MORE* – called *MSC* (MORE Service Chaining) will be introduced. Following the introduction in Section 7.3.1, in Section 7.3.2 an overview on the state-of-the-art and related work for service orchestration will be given. The proposed *MSC* mechanism will be described in Section 7.3.3 and in Section 7.3.4 results on resource consumption and on performance indicators, such as runtime are highlighted. Finally, the chapter is concluded (see Section 7.3.5) and directions for possible future work are given. In the scope of the overall design process, depicted in Figure 7.9, in this chapter a service orchestration mechanism for a flexible service-oriented middleware is provided.

7.3.1 Introduction

Sequential invocations of several web services and handling their response can be implemented manually by an application designer. Service orchestration languages try to ease this by providing methods to an automatic composition of already available web services. *MSC* will provide methods to compose *MORE Services* into large services called service chains.

The concept of service orchestration with *MSC* is shown in Figure 7.10(b). A sequential invocation of *MORE Services* on a remote node is not efficient, especially when communication to that node is performed on a link with high latency and low bandwidth. An example of a sequential invocation of *MORE Services* is illustrated in Figure 7.10(b). In this figure, firstly, *Service A* is invoked from remote and then

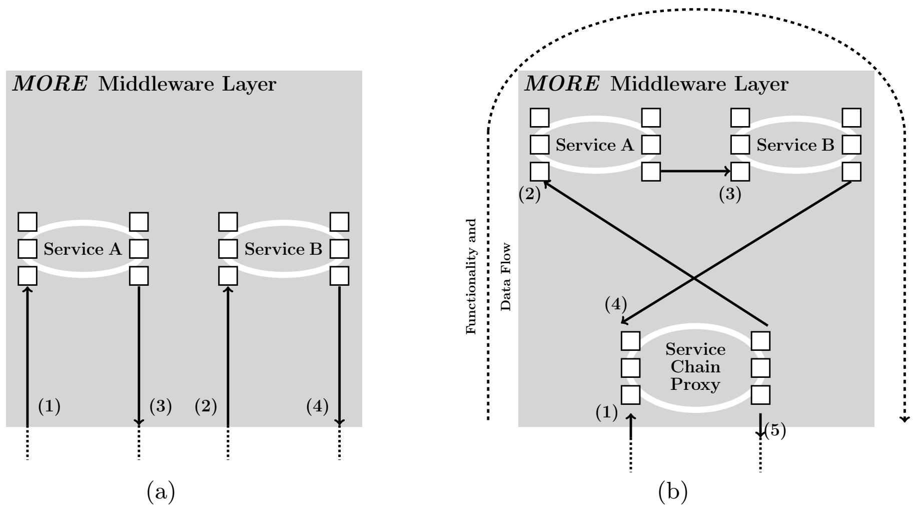


Figure 7.10: (a) Sequential *MORE Service* Invocation and (b) *MSC* Concept

Service B. In Figure 7.10(b), a service chain proxy is depicted which invokes the participants of a service chain (Service A and B) and is responsible for conducting the external communication. The functionality and data flow for a service chain invocation is annotated by indexes (1)-(5). As can be seen there, the external communication is done by the service chain proxy. The invocation of the other services, is accomplished by the service chain proxy.

In order to fit into the architecture of *MORE*, the following important requirements should be handled in *MSC*:

- *Seamless Integration:* The idea of *MSC* is to combine functionality which is used in a certain way repeatedly, in a new service. Due to the DPWS-based approach in *MORE*, it is mandatory that *MSC* should provide this new service as every other atomic service which also includes concepts such as service discovery and WSDL provision.

- *Maximal Flexibility and Dynamicity:* As *MORE Services* can be deployed at runtime on a *MORE Device*, the same functionality should be available for service chains. In addition to that service chains should support dynamic discovery of the participants, the invocation of local and remote services and, *MSC* should be not based on a statically created service orchestration, as this would be to inflexible.

- *Low Resource Utilization:* The target towards employing *MSC* on embedded systems makes it mandatory to provide a solution with a small memory footprint.

7.3.2 Related Work

Service orchestration provides efficient ways for re-using existing services and for composing a set of them to a new overall service functionality. In this section, a short overview on standards in this area is provided and existing approaches for service orchestration are described. Especially service orchestration techniques tailored towards the embedded and cyber-physical system domain are outlined.

WS-BPEL Extension for DPWS: In WS-BPEL several ways exist for utilizing participating services. One possibility is to provide an a priori known service address, e.g. a hotel room reservation system being reachable through a static address on a server. In the embedded and cyber-physical system domain, a central registration mechanism is not always available and therefore, services have to be actively discovered in the network. DPWS specifies WS-Discovery to locate devices and their services in a local area network. In [18, 19] the authors extended WS-BPEL with several DPWS-related activities in the namespace *http://www.ws4d.org/bpeldiscovery.* Therefore, the authors designed a special WS-BPEL engine that supports WS-Discovery, WS-MetadataExchange and WS-Transfer. The extensions include the following features: asynchronous and synchronous discovery of devices and services, validation of discovered services and fault/compensation handling.

```
<ws4d:discoverDev
 scopes="QName-list"? types="QName-list"? devRef="EPR-list"
/>

<ws4d:discoverSvc
 scopes="URI-list"? types="QName-list"? devRef="EPR-list"?
 devAds="QName-list"? checkRef="EPRlist"?
 svcRef="EPR-list" partnerLink="QName"?
/>

<ws4d:validateSvc
  svcRef="EPR" partnerLink="QName"
/>
```

Listing 7.1: WS-BPEL Extension Activities for Synchronous Discovery [18, 19]

In Listing 7.1 the extensions made to WS-BPEL in order to use synchronous DPWS service discovery are depicted. The discovery process for DPWS services comprises two phases as described in Section 6.2.2. In the first phase, all DPWS devices with a certain scope and type are discovered and their addresses are returned. In the second phase, the hosted services are discovered. The first action in Listing 7.1 describes the discovery of devices which returns a list of devices. The action can be constrained to certain scopes and types. The service discovery activity which is the second activity in Listing 7.1 searches for certain services and also validates if the discovered services are compatible to a WSDL or have certain attributes. In order to validate available services, an additional validation service activity was specified.

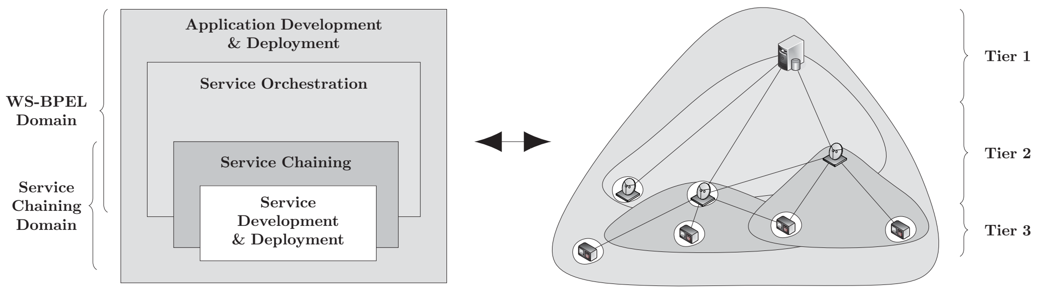


Figure 7.11: *MSC* - Application Areas

The authors of [18, 19] implemented a WS-BPEL engine that generates Java byte code from the WS-BPEL process and the WSDL descriptions. The compiled WS-BPEL process can then be executed on a DPWS-enabled device. However, this approach is not sufficient for ad-hoc networks, were high flexibility is needed.

Sliver: Sliver [62] is a project which aims at the provision of WS-BPEL to pervasive computing. Therefore, lightweight implementations towards the usage in mobile environments have been provided. In contrast to *MORE*, Sliver was not designed to cope with limited resources besides memory. In addition to that it is not that flexible, e.g. due to the lack of an ad-hoc services discovery mechanism or runtime update/deployment of services.

7.3.3 Service Chaining - Materials and Methods

The creation of a service chain and the integration of *MSC* in *MORE* will be described in this section. *MSC* is typically applied to *MORE Gateways* (see Figure 7.11), where they can be utilized for flexible service orchestration without the utilization of complicated WS-BPEL engines. The typical *MSC* application scenario is the following: A *MORE Gateway* is connected to some L3 sensor nodes via a local area network. Each L3 sensor node provides a *MORE Service* that appends its sensor values to a sequence of sensor values. This aggregated data is then available via a *MORE Service* representing the service chain. The flexibility of the *MSC* can be used e.g. for easily integrating new sensor nodes.

Configuration and Construction: *MSC* starts with designing interface description. In order to compose *MORE Services*, it is essential that the structure of the messages and the types inside the different message parts of the consecutive *MORE Service* operations are compatible with each other. The description of a service chain is an XML schema (see example in Listing 7.2) – called SCDF (Service Chain Description File) – which comprises the namespace, the identity of the service

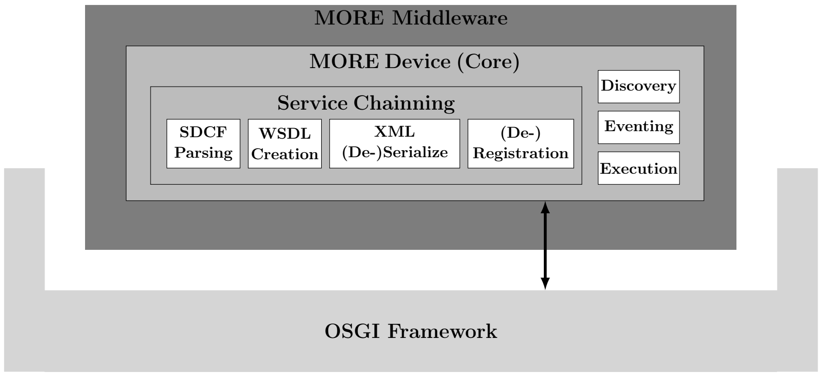


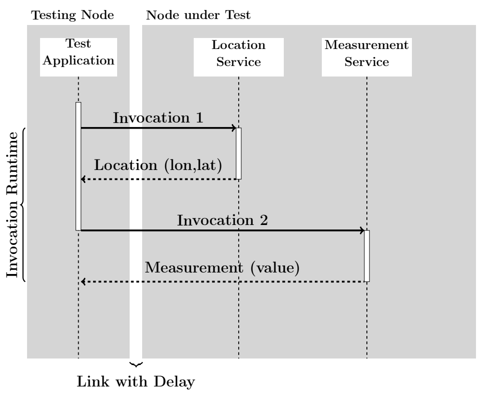
Figure 7.12: *MSC* in the *MORE* Architecture

chain and the execution sequence of the different service chain participants. The WSDL of the service chain is created on-the-fly when starting the service chain. The participants of a service chain can be local on the same *MORE Device* or remote on other *MORE Devices* in the network. This is indicated by the <local> element. When a certain service address should be utilized, the unique identifier of the *MORE Service* must be provided with the *<UID>* tag. Otherwise, an arbitrary *MORE Service* with a certain scope can be discovered with WS-Discovery and a random one is selected.

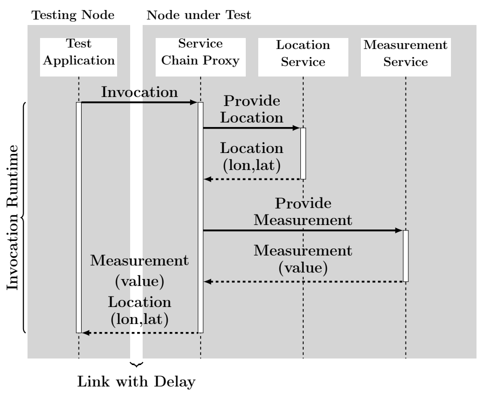
```
<tns:ServiceChainObject xmlns:tns="ServiceChaining">
<ServiceChainIdentifier>ServiceChainA</ServiceChainIdentifier>
<NameSpace>http://www.ist-more.org/SC</NameSpace>
<Operation>GM</Operation>
<DependentServices>
 <DependentService ServiceIdentifier="http://www.ist-more.org/LS">
  <ServiceOperation>GL</ServiceOperation>
  <NameSpace>http://www.ist-more.org/LS</NameSpace>
  <local>yes</local>
 </DependentService>
 <DependentService ServiceIdentifier="http://www.ist-more.org/MS">
  <ServiceOperation>GML</ServiceOperation>
  <NameSpace>http://www.ist-more.org/MS</NameSpace>
 </DependentService>
</DependentServices>
</tns:ServiceChainObject>
```

Listing 7.2: Service Chain Configuration File Example

Architectural Integration in *MORE*: The *MSC* module is directly integrated in the *MORE Core* as depicted in Figure 7.12. The module is capable of using all *MORE Core* services such as discovery or communication with other *MORE Services*. The life cycle of a service chain comprises, analogous to the *MORE Service* life cycle, four steps: service chain construction, service chain deployment, service



(a)



(b)

Figure 7.13: (a) Direct Remote Invocation of two Services and (b) Remote Invocation of a Service Chain. Both over a Link with Delay.

chain utilization and service chain un-deployment. Before creating the *MORE Service*, the *MSC* module parses the *SCDF*, discovers participating (remote) *MORE Services*, checks the availability of the participating *MORE Services*, registers the *MORE Services* of the service chain to the *MORE Core* and finally creates the WSDL of the service chain on-the-fly. If a member of a service chain fails or is not available, the service chain will return an error message.

As described in the former section, *MSC* is capable of creating a service chain from *MORE Services* available on the same *MORE Device* or are available on other *MORE Devices*. The invocation process for participating *MORE Services* inside the *MSC* module consists of three phases. In the first phase, the incoming message parts are parsed and transformed to a format which can be processed by the service. Secondly, the participating *MORE Service* is invoked. The response of a service is transformed and sent back to the invoker of the service chain or to the succeeding participating *MORE Services*.

7.3.4 Evaluation

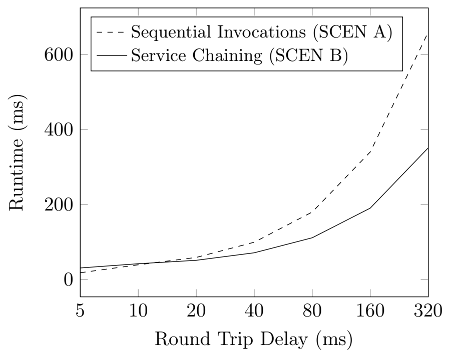
The evaluation is split into two parts. First of all, some resource considerations are made (see Section 7.3.4.1), followed by a runtime efficiency evaluation for an example scenario is given in Section 7.3.4.2.

7.3.4.1 Resource Considerations

MSC is based on the DWPS4J implementation [117] and it requires an external parser (Apache Crimson) for marshalling between the different data types of the participanting *MORE Services*. The total ROM footprint for *MSC* is 335 KB. With regard to commercial WS-BPEL engines having a size of at least 20 MB [19] in addition to the Java environment, this footprint is far smaller.

Module	Size (KB)
Service Chaining Module	134
Apache Crimson	201
Total	335

(a)



(b)

Figure 7.14: *MSC* Evaluation Results: (a) Memory Footprint *MSC* Module and (b) Invocation Times for Evaluation Scenarios

7.3.4.2 Use Case Evaluation

In this section, it will be evaluated if *MSC* on a L2 node is beneficial in terms of runtime. Therefore, two test scenarios are compared to each other which emulate the influence of the link delay of long-range wireless connections. Per link direction an additional delay was added to all outgoing packets with the help of the traffic control of Linux [3] in order to emulate long-range wireless connections. The scenarios were created as depicted in Figures 7.13(a) and 7.13(b). In Figure 7.13(a), both services on a DPWS device are sequentially invoked. In *Invocation 1* the location of the node is examined with the *Location Service* and in a second invocation the update-to-date sensor values are requested. In Figure 7.13(b) the same functionality is provided but the testing application requests the service chain proxy to acquire measurement data with position information.

The results for the evaluation are shown in Figure 7.14(b). *SCEN A* comprises the scenario depicted in Figure 7.13(a) and *SCEN B* the one illustrated in Figure 7.13(b). Both, the *Node under Test* and the *Testing Node* were executed in a separate virtual machine with traffic control equipped Linux. For each scenario, 300 service invocations were conducted. The aggregated link delay was chosen between 0 ms and 320 ms. As can be seen in Figure 7.14(b), *SCEN A* outperforms *SCEN B* only when the link delay is below 10 ms. When the link delay is higher the invocation of the service chain brings higher performance.

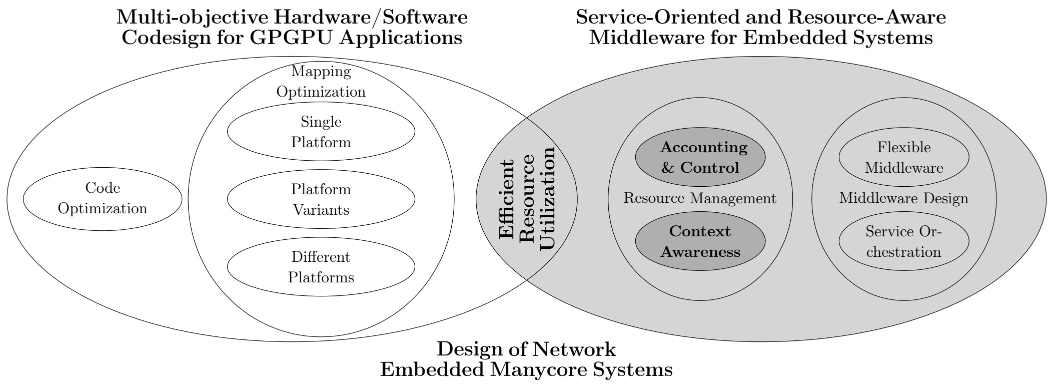

Multi-objective Hardware/Software Codesign for GPGPU Applications
Service-Oriented and Resource-Aware Middleware for Embedded Systems
Mapping Optimization
Single Platform
Platform Variants
Different Platforms
Code Optimization
Efficient Resource Utilization
Accounting & Control
Resource Management
Context Awareness
Flexible Middleware
Middleware Design
Service Orchestration
Design of Network Embedded Manycore Systems

Figure 7.15: Resource Management - Overview on the Overall Design Process

7.3.5 Summary

In this section, a service orchestration approach – called *MSC* (MORE Service Chaining) – based on *MORE* and tailored towards embedded gateway nodes (*MORE Gateways*) in hierarchical networks was presented. The *MSC* approach targets gateway nodes in 3-tier network topologies. The *MORE Service* representing the service chain is created on the *MORE Gateway* at runtime and allows L2 and L1 nodes to access the service chain as normal *MORE Services*. In an evaluation, it was shown that in networks with large latencies the processing overhead for *MSC* is acceptable and that *MSC* can provide a flexible service orchestration mechanism.

7.4 Resource Management

The resource management facilities implemented in *MORE* are described in this chapter. After an introduction to the research areas *resource management* and *non-functional requirement awareness* in Section 7.4.1, in Section 7.4.2, state-of-the-art technologies and related work to these areas will be presented. In the scope of the overall design process, depicted in Figure 7.15, in this chapter novel functionalities for the resource accounting/control and two resource management mechanisms, *NOFURSI* (NOn-Functional Requirements aware Service Invocation) and - *NOFURAS* (NOn-FUnctional Requirements Aware Subscription) will be introduced. These resource management mechanisms provide context awareness for *MORE Services* and *MORE Devices*. The chapter ends up by presenting results and summarizing the section.

7.4.1 Introduction

According to the authors of [103], *"non-functional requirements are ... referred to as constraints, softgoals, and the quality attributes of a system"*. This non-functional *information* should be modelled in a middleware architecture to handle resource utilization efficiently. As stated in [55], software design comprises three non-functional core concepts. First of all, there are non-functional attributes which comprise certain classes of attributes such as *time efficiency.* The second non-functional concept is that of a non-functional behaviour which is the assignment of a non-functional attribute to a software component. The last term in this context is a non-functional requirement which is the actual assignment of a concrete value to a non-functional behaviour. The handling of such non-functional *information* is the domain of the *MORE Resource Management* which is designed for providing context awareness for *MORE Devices* in terms of resource utilization.

The following resources should be considered in *MORE Resource Management* [4]:

- *Processing Resources*: In *MORE*, multi-task environments with priority based scheduler at operating system level will be considered. Hence, the processing capability of a service can be controlled e.g. by priority assignment, time allocation techniques [5] or adaptive sampling.

- *Energy Consumption*: The energy consumption is one of the most critical non-functional attribute for mobile devices. Therefore, it must be considered in resource management for middleware applications comprising mobile devices.

- *Bandwidth*: Especially in remotely deployed sensor networks, the bandwidth is a critical issue. High bandwidth and long distance communication are inducing a high energy consumption. Therefore, it can be beneficial in terms of energy consumption to process or compress data in-situ on the sensor nodes.

In order to handle resources efficiently, information from the middleware layer should be taken into account [124], e.g. information about processing demands. The design of the resource management should comprises the following requirements:

1. *Modular Resource Management*: Due to diverse hardware platforms it is mandatory that resource management can load different configurations and provide different hardware platform aware plug-ins.

2. *Tight Integration*: An efficient resource management approach must be tightly coupled to different services of the middleware layer, in particular with the *MORE Core.* Especially the execution services and the eventing services must interact with the resource management.

3. *Multi-Layered Approach*: Not all features of the resource management can be employed in the middleware layer as they directly target the execution platform. Therefore, there is a need to implement resource management functionality on operating system and hardware level.

7.4.2 Related Work

Several research fields build the basis for efficient resource utilization and resource management. At first, related work dealing with energy consumption in SANETs will be presented, followed by the description of adaptive sampling and resource management techniques for sensor/actuator networks.

In [120], the authors presented an approach to reduce the amount of data over time which is transferred in a wireless sensor network. The authors mainly focussed on efficient data packet routing in a network and on the aggregation of data inside a network to reduce energy consumption. The authors of [91] described routing and topology building methods for a wireless sensor/actuator network. Their method models the end-to-end delay and the energy consumption as a hard constraints which must be met. In addition to that, they restrict the topology of the network, in such a way that there are only connections between one sensor and one actuator. Saving energy was accomplished in the aforementioned works mostly by efficient routing which is not the objective of the resource management functionalities to be presented in this chapter.

The amount of work already accomplished in the field of resource management is enormous and therefore, only an excerpt from common literature can be provided. One resource management method to be described in this chapter adapts the triggering frequency of the sensor nodes in the network which can be seen as a type of *Adaptive Sampling*. In this research area, several approaches exist such as the work presented in [2]. Within that work, the authors proposed an adaptive sampling method that enables the deployer of a wireless sensor network to conserve energy by exploiting the fact that sometimes the energy consumption for processing is higher than for communication. The proposed method reduced the number of processing events by estimating the optimal frequency in which the sensor nodes must be triggered and thereby decrease the amount of processing on sensor node respectively the energy consumption.

Work in the field of resource management and adaptive applications was described in [5]. In [5], the authors created a time sharing protocol which allows to assign a fixed percentage of processing time to a task in a Linux operating system. This work was extended to cooperate with *MORE*, by allowing a *MORE Service* to occupy certain processing resources. In [40], the authors presented a framework for creating networks with different QoS levels. In that work a remote procedure call which can adapt to a QoS change was introduced. Important QoS values such as the utilization of network link are monitored. In [16], several techniques are introduced which can be used to reduce the energy consumption of embedded systems by dynamic power management. The so called *policy optimization problem* – outlined in [16] – is especially interesting for resource management functionalities to be presented in this chapter. The policy optimization problem describes – in a theoretical way – how to optimize a system with dynamic power management techniques. Another project targeted towards resource management in the field of embedded and cyber-physical system middleware is BETSY [111, 118]. In the BETSY project, formal models were developed which allow to build applications with different QoS levels. These models can then be utilized to adapt a system to environmental changes. For example, in a mobile streaming application, several video versions with different encoding qualities can be available. In the case that the bandwidth of an available wireless link decreases, the resource management can decide to switch to another version of the video which uses less bandwidth. The QoS will also be considered in the resource management functionalities to be presented in this chapter but from the sampling perspective. A holistic resource management approach was developed by the GRACE project [144]. The objective of the GRACE project was to provide multi-layer resource management for mobile devices. The operating system scheduler, the network links and the CPU are monitored for that purpose. On the controlling side, a per application controller and a global controller exist. The global controller is responsible for optimizing the system utilization while considering constraints for the processing resources and available network link bandwidth. The per application controller is responsible for optimizing the execution of a single application. Further resource management works and the challenges to an efficient embedded system design can be found in ARTIST Design roadmap [24].

7.4.3 Resource Management - Materials and Methods

In this section different methods for *MORE Resource Management* are introduced. At first, in Section 7.4.3.1, accounting and control methods are presented (see Section 7.4.3.1), followed by a description of two resource management functions used in *MORE* (see Section 7.4.3.2 and 7.4.3.3).

7.4.3.1 Resource Accounting and Control

The basic features of the *MORE Resource Management* are the control of resource accesses and the accounting of resources utilization. This is partly done at middleware level and operating system level and demands for a tight integration of these two levels. MORE resource management is the connecting element between the middleware level and operating system level. It provides the possibility to install so-called *resource units* at the operating system level as depicted in Figure 7.16. Examples for such *resource units* are a *DVFS unit* which enables the resource management to control the voltage and frequency of the CPU or memory, a *CPS unit* [5] which can be used to allocate a certain amount of processing time to services and a *Processing Interval Unit* which tracks for the publish-subscribe mechanism of *MORE* the interval between event notifications and runtime of single service invocations.

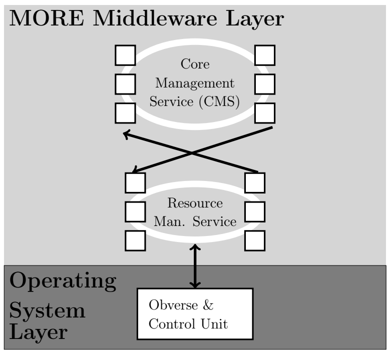


Figure 7.16: Resource Management - Accounting and Control

7.4.3.2 Non-Functional Requirements Aware Service Invocation (*NOFURSI*)

In traditional web service environments, standards such as WS-Policy [10] exist which allow a service consumer to request from a service provider a description on non-functional attributes such as security or quality. Projects like PLASTIC [9] use similar approaches to provide a resource optimized version of services based on so-called SLS (Service Level Specifications) and context information. However, in the field of embedded systems, a deeper access is needed, e.g. by controlling the scheduling policy or the processing power of a CPU. The *resource units* presented in the last section control the processing of a CPU and will be used in the context of the *NOFURSI* (Non-Functional Requirements aware Service Invocation) implemented in *MORE*.

The *NOFURSI* approach is designed as depicted in Figure 7.17. The non-functional attributes can be e.g. specified in a WSDL file analogue to the proposed specification of performance qualifiers for web services in [39]. A service consumer (Requesting Service) wants to invoke a service with the maximal available processing power of the service provider's (Providing Service) hardware platform. The single steps for *NOFURSI* are labelled with a natural number from (1) to (6). Firstly, the service provider requests the WSDL file containing available levels of a non-functional attribute: *processing power*. Secondly, this WSDL file is sent to the service consumer. Thirdly, the service consumer sends a service invocation message including a non-functional requirement towards a specific processing power level. The service provider then requests the resource at the resource management, the service is executed and the resource is released. In the last step the service consumer can retrieve the information that it has requested.

In the following, the functionality of the DVFS-Unit - already mentioned in Section 7.4.3.1 - will be explained in more detail. DVFS (Dynamic Voltage and Frequency Scaling) techniques estimate the most suitable frequency and voltage

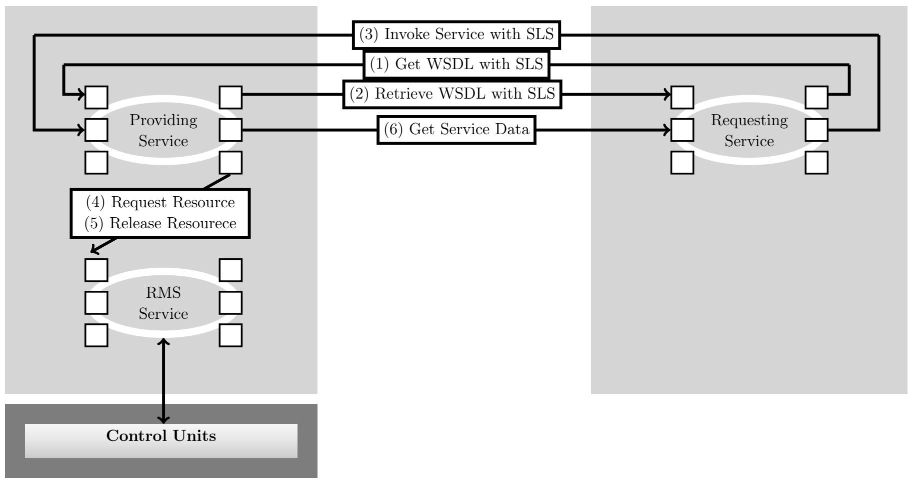


Figure 7.17: Resource Management - *NOFURSI*

configuration to reduce the dynamic power consumption. This can be e.g. done by a simple regulation rule: If the processor load is high, use a high processor frequency. If the processor load is low, use a low processor frequency. A dominant factor of the dynamic CMOS power consumption is the *transient power consumption* which describes the power consumption of transistor state changes. The *transient power consumption P* in watt (W) for CMOS circuits is defined as [32, 126]:

$$P = N \cdot c_f \cdot f \cdot V_{dd}^2, \tag{7.1}$$

where N is the switching activity of the transistors, c_f is the load capacity in ampere-seconds per volt ($\frac{\mathrm{As}}{\mathrm{V}}$), f the operating frequency in Hertz (Hz) and V_{dd} the supply voltage in volt (V). Parameters which can be scaled by the DVFS-Unit of the resource management are the frequency and the supply power. By decreasing them the power consumption will also decrease. As runtime will not increase with the same factor the power consumption will decrease, this will have a positive effect on the energy consumption [84]. The DVFS-Unit can be requested to change the frequency and voltage mode.

7.4.3.3 Non-Functional Requirements Aware Subscription (*NOFURAS*)

The capability to adapt the service provider's behavior to non-functional requirements and its integration in the eventing services of *MORE* will be presented in this section. The new functionality is summarized by the term: *NOFURAS* (NOn-FUnctional Requirements Aware Subscription).

A tight integration with the *MORE Core* and the operating system enables the resource management service (see Figure 7.17) to handle the non-functional information added to the subscription of a service as depicted in Figure 7.18 (described

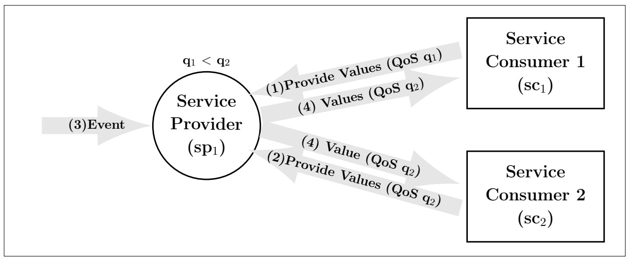


Figure 7.18: Resource Management - *NOFURAS*

later). This specification of the requirements of the subscriber enables the resource management to adapt the behaviour of the system towards this SLA (Service Level Agreement). *NOFURAS* is designed to track resources such as the average runtime of a service. The non-functional attributes can e.g. be specified in the WSDL description analogously to the proposed specification of performance qualifiers for web services in [39]. An example subscription message with non-functional requirements to the update interval of the service consumer is depicted in Listing 7.3. As one can see in that listing, the non-functional information is added to the header of the subscription message. The integration was accomplished in a way that these messages could also be interpreted by DPWS devices which do not need non-functional requirements. A new namespace *nonfunc* was created in which the non-functional requirements can be specified. The non-functional requirements are listed in a XML sequence called *nonfunc:Parameters* which includes one or more non-functional items. Each item has a *name*, a *value* and a *unit* as one can derive from Listing 7.3. The requirement which is listed in Listing 7.3 shows lower (490 ms) and upper bounds (510 ms) for the sampling interval of a service provider.

```
<SOAP–ENV:Envelope ... xmlns:nonfunc="...">
 <SOAP–ENV:Header>
   ...
   <nonfunc:Parameters>
    <nonfunc:item>
      <nonfunc:Name>notify_int_low</nonfunc:Name>
      <nonfunc:Value>490</nonfunc:Value>
      <nonfunc:Unit>ms</nonfunc:Unit>
    </nonfunc:item>
    <nonfunc:item>
      <nonfunc:Name>notify_int_high</nonfunc:Name>
      <nonfunc:Value>510</nonfunc:Value>
      <nonfunc:Unit>ms</nonfunc:Unit>
    </nonfunc:item>
   </nonfunc:Parameters>
 </SOAP–ENV:Header>
  ...
</SOAP–ENV:Envelope>
```

Listing 7.3: Integration of Non-Functional Requirements in Subscription Messages

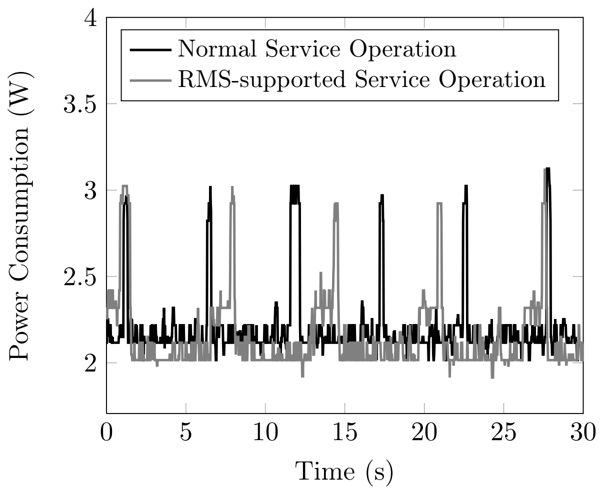


Figure 7.19: *NOFURSI* - Power Consumption for DVFS-Unit on a gumstix

The adaptation of the behavior of the service to SLA is controlled by *NOFURAS*. In particular, the resource management service handles all accesses of a service to the underlying operation system and libraries and controls the execution of the service. The control functionality can include features such as the (average and worst case) runtime of a service, the quality of the result of a service or service failures. This is needed in order to regulate the processing of a service towards a conformance to the specified non-functional requirements. An example of how an SLA or QoS Decision can be used with *NOFURAS* is depicted in Figure 7.18. For example, there could be service consumers sc_1 and sc_2 which are interested in a certain service provider sp_1 providing an image analysis with certain frame rates with respective QoS of $q_1 < q_2$. With *NOFURAS*, sc_1 and sc_2 can now inform sp_1 with which particular QoS the results of sp_1 are required (e.g. sc_1 needs q_1 and sc_2 needs q_2). For the QoS, several policies could be applied, e.g. provide a service with a QoS that satisfies all requirements. If the latter is applied, NOFURAS on sp_1 can choose q_2 in order to satisfy the non-functional requirements of sc_1 and sc_2.

7.4.4 Evaluation

In this evaluation section results and use cases will be provided for both resource management mechanisms: *NOFURAS* and *NOFURSI*. For *NOFURSI* , the general applicability will be evaluated in Section 7.4.4.1 and for *NOFURAS* the intra-logistic use case will be evaluated for the resource utilization in Section 7.4.4.2.

7.4.4.1 *NOFURSI*

In this section, the functionality of *NOFURSI* is evaluated. The DVFS-Unit allows the switching of the voltage and frequency of the main processor. The measurements were conducted on a *gumstix verdex pro X6LP* with a *Marvell PXA270* main processor. The gumstix was attached to a *netpro-vx expansion board.* In addition to that, the gumstix was connected via Ethernet to a standard PC which hosts the service consumer.

As test services a measurement service and a location service were used which were triggered periodically. Two configurations have been tested. In the first configuration (Label: *Normal Service Operation*) the services were invoked without using *NOFURSI* and the gumstix was running at full speed (600Mhz). In the second configuration (Label: *RMS-supported Service Operation*), the normal operation speed of the gumstix was 200 MHz and for service invocation the service requests a higher speed of 600Mhz. As can be seen from Figure 7.19, the scenarios work as they should. The power consumption was decreased for the scenario *RMS-supported Service Operation.* In order to calculate the energy consumption for the invocation of the services, a five seconds interval was chosen which includes exactly one invocation for both configurations. The energy consumption for the first configuration was 11.18 J and for the second configuration it was 11.11 J, i.e. the energy saving was around 1%. The effect which leads to this small savings is also shown in Figure 7.19. As can be seen in that figure, the change of the voltage and frequency level needs additional time and therefore additional energy. As one can see there is a drift in the two curves which is there due to the change of the voltage and frequency.

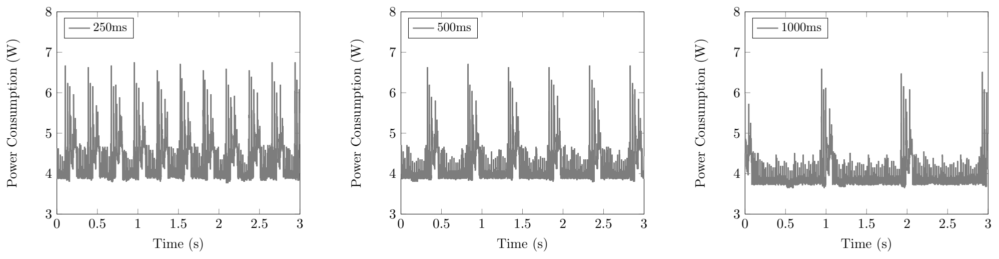
7.4.4.2 *NOFURAS*

This section provides the basic requirements to evaluate the functionality and efficiency of the proposed resource management in a logistics system architecture and the corresponding results. The evaluation was done to show how resources such as energy can be conserved by providing non-functional requirements to the subscription process.

Use Case - Intra Logistics Scenarios: As an exemplary system a SANET which controls a conveyor belt system is considered (see Sections 2.2 and 7.2.4.2).

In that particular SANET light-barriers and RFID-readers at the switches are replaced by low-cost cameras and an in-situ marker detection system (as depicted in Figure 7.8). The employed marker technology is called QR code [44]. The image processing as part of the marker detection system was accelerated by a parallel processing hardware based on OpenCL [61]. The topology of the SANET is as follows (depicted in Figure 2.3): The sensor nodes (camera systems) can control one or more switches and the belt is in front of them. The actuator nodes of the SANET are the switches which subscribe to the sensor nodes. All sensor nodes and actuator nodes are equipped with *MORE* and the new resource management *NOFURAS.*

One of the most critical parameters in terms of energy consumption of a service



(a) (b) (c)

Figure 7.20: Power Consumption for Different Subscription Intervals: (a) 250ms, (b) 500ms, (c) 1000ms

provider is the interval in which events take place/have to be processed and how often a service consumer needs that information. If, for instance, the information (QR code) which is provided to the service consumer is not updated in subsequent events, this information need not be transferred again. In terms of the conveyor belt, this could be a parcel which is still on the same trail towards a switch. The switch as a service consumer is only interested in the parcel's data if it represents new information and therefore, non-functional requirements with the necessary update interval are added to the sensor node's subscription. The sensor node can then adapt to this requirement. With the update interval, the minimal and maximal time interval in which a sensor node has to provide data if events occur is described. *NOFURAS* can actively restrain the events which are published by a sensor node. If more than one actuator node subscribes to a sensor node, the minimal update interval is chosen.

The following tests were conducted: The energy consumption and the processing time were measured for two image sizes: 320 × 240 pixels and 640 × 480 pixels. After these initial tests, several update intervals were tested in order to determine the update interval with optimal energy consumption for the considered conveyor belt system. The update intervals of the sensor nodes are 250±10 ms, 500±10 ms and 1000±10 ms. They are added as lower and upper bounds for the update interval to the subscription messages as depicted in Listing 7.3. The maximal speed of the considered conveyor belt system is approximately 1 m/s. Due to this and the architecture of the system, the largest possible update interval is 1000±10 ms, otherwise not all parcels can be identified properly. This is the standard speed of conveyor belt systems. The baseline system configuration is a publish/subscribe *MORE* environment without any restriction in terms of the sampling frequency of detected QR codes. Every detection is therefore transmitted. The evaluation was conducted with the energy consumption and performance testbed presented in Chapter 3. The device under test was a ZMS-8 from Ziilabs.

For evaluating the aforementioned energy consumption and runtime, the testbed

	One Detection		No Detection	
Image Size (Pixels)	r^{P_a,P_l}_{frame} (ms)	E^{P,P_l}_{frame} (J)	r^{P_a,P_l}_{frame} (ms)	E^{P_a,P_l}_{frame} (J)
320×240	88	0.377	36	0.152
640×480	291	1.271	132	0.574

Table 7.1: Energy Consumption for Analysing a Frame

provides the energy consumption

$$E^{P_a,P_l}_{frame} = energy(P, P_l, t^{P,P_l}_{start}, t^{P,P_l}_{end}), \tag{7.2}$$

where P_a is the QR code detection pipeline (shown in Figure 2.4), P_l the execution platform (ZMS-8 from Ziilabs) consuming power over time, t^{P_a,P_l}_{start} and t^{P_a,P_l}_{end} denoted the start and the end of processing one frame. They are all delivered by the testbed. The runtime of a program P_a on a platform P_l is therefore

$$r^{P_a,P_l}_{frame} = runtime(P_a, P_l) = t^{P_a,P_l}_{end} - t^{P_a,P_l}_{start}. \tag{7.3}$$

$t^{P_a,P_l}_{interstart}$ and $t^{P_a,P_l}_{interend}$ denoted the start and the end of the processing on P_l for 1 second. The corresponding energy consumption E^{P_a,P_l}_{inter} is

$$E^{P_a,P_l}_{inter} = energy(P_a, P_l, t^{P_a,P_l}_{interstart}, t^{P_a,P_l}_{interend}) \tag{7.4}$$

and the runtime r^{P_a,P_l}_{inter}:

$$r^{P_a,P_l}_{inter} = runtime(P_a, P_l) = 1\text{s}. \tag{7.5}$$

Results: The results in Figure 7.20 show the power consumption over time for different update intervals on a sensor node and thereby, indicate that theses nodes can adapt to the desired update intervals. While waiting for the next image, the power consumption ranges from 3.5 W up to 4.5 W. During image processing, up to 6.75 W are consumed by the sensor node. In this figure, only processing intervals are shown where a QR code was fully decoded. Therefore, the execution times for images with incomplete QR code detection are shorter. On the other hand, image processing methods and most of the detection-related algorithms are executed, regardless of the presence of a QR code in the image.

Table 7.1 shows the results for energy consumption and the processing time for different image sizes. When an QR code is detected in an image, analysis required 88ms for a 320×240 pixels image and 291ms for 640×480 pixels image (energy consumption: 0.377 J, respectively 1.271 J). The lower bound for image processing (no QR code in an image) within the marker detection process for a 320×240 pixels image amounts to 36 ms and to 132 ms for a 640×480 pixels image. The minimal

Image Size (Pixels)	Interval ($\pm$10 ms)	Minimal E_{inter} (J)	Maximal E_{inter} (J)
320×240	Without	196	212
320×240	250	188	202
320×240	500	185	191
320×240	1000	183	187
640×480	Without	211	225
640×480	250	-	-
640×480	500	194	218
640×480	1000	187	200

Table 7.2: Energy Consumption per Minute for Certain Intervals

energy consumption for these lower processing bounds amounts to 0.152 J, respectively 0.574 J.

The energy consumption results for the sensors nodes and the different update intervals are depicted in Table 7.2. The energy consumption values are measured for a one minute time frame. Minimal energy is consumed when no QR code was processed in the specified interval and maximal energy is consumed in the specified interval when for each processed image a QR code was recognized. When no update interval was specified in a subscription message then minimal energy consumption for an image of size 320×240 pixels is 196 J and 211 J for an image of size 640×480 pixels. The maximal energy consumption for the same scenario is 212 J(320×240) respectively 225 J (640×480). These energy consumption values also characterize the system without *NOFURAS* and therefore, they are used as the baseline for the evaluation. For an image size of 640×480 pixels and an update interval of 250$\pm$10ms, an evaluation is not possible since the average runtime of QR detection (291ms) exceeds this interval. For the largest update interval (1000$\pm$10 ms), the minimal energy consumption for an image of size 320×240 pixels is 183 J and 187 J for an image of size 640×480 pixels. The maximal energy consumption for the same scenario (1000$\pm$10 ms) is 187 J (320×240) respectively 200 J (640×480). This means that, in total, 12% (187 J/212 J) energy is saved in a system working with images of size 320×240 pixels and an update interval of 1000$\pm$10 ms, in comparison to a system without *NOFURAS*. For 640×480 pixels sized images, there is a reduction of the energy consumption of up to 8% (200 J/225 J). Overall, it can be summarized that by extending the update interval of the sensor nodes energy consumption can be reduced.

7.4.5 Summary

This section described an enhancement of *MORE* by the utilization of non-functional requirements for enabling a more efficient usage of resources in a sensor/actuator

network. Therefore, the two resource management approaches *NOFURAS* and *NOFURSI* were designed to control and track the behaviour of running services. *NOFURAS* is able to evaluate the non-functional requirements encapsulated in the subscriptions messages. The subscription process of the *MORE* was extended to specify non-functional requirements by the service consumer and to interpret them on the service provider side. *NOFURSI* is able to control the resource utilization of service invocation.
NOFURAS was then tested on an automated facility logistics system. The sensor/actuator network in this system under concern was provided with *NOFURAS* and it was tested towards it capability to save energy under the specification of a certain event notification interval. *NOFURSI* was evaluated with test scenarios showing the applicability of *NOFURSI* in *MORE*.

7.5 Conclusion

The use of SOA (Service-Oriented Architectures) and web services becomes more and more standard in the field of embedded and cyber-physical system software but the lack of certain features such as an efficient resource management reduces the applicability. In this chapter, therefore several approaches to the refinement of SOA-based system software were proposed, designed and evaluated.

Firstly, an efficient middleware approach, called *MORE* (Network-centric Middleware for GrOup communication and Resource Sharing across Heterogeneous Embedded Systems) was presented which supports an application designer in several steps of the web service development process. Especially, for the phases *design*, *deployment* and *maintenances*, new techniques were provided. For the latter two phases, a flexible deployment and maintenance approach was introduced in *MORE* and for the design phase a set of application-independent services, called added value services were provided. This approach enables *MORE* to integrate services on-the-fly in an ad-hoc network. Due to the use of DPWS as a base technology, it is possible to integrate *MORE* in standard web service environments. Additionally in this chapter a lightweight service orchestration mechanism – called *MSC* (MORE Service Chaining) – was presented, which enables application designers to compose *MORE Services* to a single service. It was shown that *MSC* suits well for systems at remote locations which have high communication latencies.

The SOCRADES technology roadmap [124] states that energy consumption must be a major optimization goal for systems using service-oriented architectures. Therefore, in this chapter, functionalities for resource accounting/control and two resource management mechanisms, *NOFURSI* (Non-Functional Requirements aware Service Invocation) and - *NOFURAS* (NOn-FUnctional Requirements Aware Subscription) have been presented. *NOFURAS* provides an application designer with the possibility to create sensor/actuator networks with *MORE* adding context-awareness to the subscription process. The other resource management service, *NOFURSI*, extends *MORE* with the capability to require and control the utilization of multiple resources at service execution. As an exemplary resource, the processing power of the CPU was considered and evaluated by applying DVFS-techniques.

Overall, it can be summarized that optimizing the energy consumption is an important objective for the design with service-oriented architectures, especially when using complex middleware libraries in the embedded system world, where mobile systems are omni-present.

Chapter 8

Conclusion and Future Work

This chapter is the conclusion chapter of this thesis and in it directions for future works are given.

Contents

Two major research areas towards pervasive computing devices exist: *Embedded and Cyber-Physical Systems* and *Communication Technologies*. In both areas resources can be scarce. Particularly processing resources and energy consumption should be considered while designing systems. Therefore, this thesis proposed methods and conducted evaluations towards an efficient resource utilization at runtime and methods taking into account resources at design time. The summary of the work accomplished in this thesis will be given in Section 8.1 and possible future work will be described in Section 8.2.

8.1 Summary

A special class of networked embedded systems are *Networked Embedded Many-Core Systems* which have been considered in this thesis. These systems are designed to be included in smart sensor/actuator networks jointly performing a control task. The aforementioned systems often have high computational requirements, e.g. for performing real-time image processing, and therefore should include many-core chips. The design and development process for this class of devices demands for dedicated methodologies towards an efficient resource usage, especially the combined optimization of energy consumption and runtime performance. For the research areas, a resource-aware design process was proposed. For the *Embedded and Cyber-Physical Systems* part, a *Multi-objective Hardware/Software-Codesign for GPGPU Applications* was developed and evaluated. In the scope of *Communication Technologies*, a *Service-Oriented and Resource-Aware Middleware for Embedded Systems* was proposed and also evaluated.

Multi-objective Hardware/Software-Codesign for GPGPU Applications: The major disadvantage of today's GPGPU programming is the dominance of manually performed code optimizations, as well as manual mapping optimizations. For both optimization types, advanced design methods have been proposed which include profiling-based optimizations of code and mappings. Additionally, they comprise multi-objective optimization techniques towards runtime and energy consumption efficiency.

The two code optimization techniques (*MOBLIS* and *FALIS*) are based on instruction scheduling in an optimizing compiler. The *MOBLIS* (Multi-OBjective Local Instruction Scheduling) approach assigns scheduling policies to single basic blocks in the PTX representation of a kernel. It was possible to decrease runtime and energy consumption. *FALIS* (Feedback-based and memory-Aware gLobal Instruction Scheduling) is based on global instruction scheduling with a focus on memory-related instructions. With *FALIS* reductions in energy consumption and in runtime can be observed. Both code optimization techniques are based on the use of profiling for runtime and energy consumption inside an optimizing compiler, which has not been done before in the scope of GPGPU application design. Additionally, the use of multi-objective genetic algorithms in combination with the optimizing compiler is a novelty in the GPGPU application design process.

The two mapping optimization techniques conducted design space explorations towards energy efficiency. The first DSE targets the decision, which platform is most suitable for a GPGPU application under the constraint of deadlines. The second DSE was targeted towards the decision, whether or not an integration of a GPGPU-capable graphics card for parallel application acceleration is beneficial to energy consumption. Both mapping optimizations are based on the use of profiling for runtime and energy consumption. In combination with selecting the most suitable graphics card for a GPGPU application, the conducted evaluations show novel results and directions towards a resource-aware GPGPU application design process.

Service-Oriented and Resource-Aware Middleware for Embedded Systems: Service-Oriented Architectures are used in a wide range of embedded and cyber-physical system environments. The vast variety of hardware platforms, communication links and application areas in these environments make an efficient use of SOA challenging. Especially the development of flexible configuration and deployment mechanisms, of efficient orchestration mechanisms and of elaborated context-awareness techniques are research targets.

With *MORE* (Network-centric Middleware for GrOup communication and Resource Sharing across Heterogeneous Embedded Systems), a service-oriented middleware for the embedded and cyber-physical system domain was developed. *MORE* provides a flexible configuration and deployment mechanism and enables integrating services on-the-fly in an ad-hoc network. Due to the use of web services as base technology, it is possible to integrate *MORE* in standard web service environments.

Additionally, a lightweight service orchestration mechanism called *MSC* (MORE Service Chaining) was developed, which enables application designers to compose several *MORE Services* into a single *MORE Service*.

In order to handle context-awareness and efficient resource utilization, two resource management mechanisms (*NOFURSI* and *NOFURAS*) have been developed for *MORE*. *NOFURSI* (Non-Functional Requirements aware Service Invocation) extends *MORE* with the capability to request resources at runtime and control their utilization at service execution. *NOFURAS* (NOn-FUnctional Requirements Aware Subscription) provides an application designer with the possibility to create context-aware sensor/actuator networks with *MORE* . This was achieved by extending the subscription process of *MORE* by the capability of adding non-functional requirements. Both resource management service techniques are based on the use of profiling for runtime and energy consumption on reference platforms.

8.2 Future Work

The methods and evaluations presented in this thesis cover a wide range of aspects in the design process of *Networked Embedded Many-Core Systems* and improved this design process in several points. The elaborated design process offers opportunities at several points which can be improved in the future.

Multi-objective Hardware/Software-Codesign for GPGPU Applications: Considering the optimization gain achieved by the code (see Sections 5.3 and 5.4) and mapping optimizations (see Sections 5.5 and 5.6) presented in this thesis, an even higher optimization can be assumed by taking into to account differently parallelized versions of the same algorithm. Automatic parallelizers such as Loopo [60] or Pluto [20] are heavily based on variable parameters which can influence the performance of a GPGPU application. These parameters should be optimized in a structured way by design space exploration techniques, in addition to the methods and evaluations presented in this thesis.

The code and mapping optimization techniques presented in this thesis targeted data parallelism only, i.e. only instances of one kernel run concurrently on a GPU. Recent GPUs are capable of executing different kernels concurrently. Therefore, in future work it should be explored, how task parallelism capabilities can be used to boost the performance of GPGPU applications, e.g. by executing kernels with many data accesses concurrently to kernels with few data accesses.

The mapping optimizations presented in this thesis assume that the quality of the result is not influenced by the optimizations. Especially in the fields of data mining, image processing and image analysis, the quality of the results often depends on the runtime of the algorithms. In future work, it should be evaluated how a trade-off between quality and performance can by efficiently integrated in the design process for *Networked Embedded Many-Core Systems*.

Service-Oriented and Resource-Aware Middleware for Embedded Systems: The middleware concept (*MORE*) presented in this thesis (see Section 7.2), provides the possibility to tailor the software stack for a larger number of different devices. However, this configuration process is still performed manually for each type of device. In future work it can be evaluated, how ontologies can be used to efficiently describe the features of the software stack and of the hardware devices. This can then be used to automatically tune the middleware software stack towards a specific device.

The resource management functions presented in this thesis (see Section 7.4) optimize single resources. A more holistic approach is optimizing several resources concurrently. This can be utilized to achieve an even higher optimization gain. Therefore, the one-level *Resource Unit* approach, should be extended to a multi-level approach where there are *Resource Units* which utilize other *Resource Units*. This approach can then be used to optimize several resources.

APPENDIX A

Appendix

A.1 Definitions

Definition 5 *(Basic Block)*
A basic block b_i is a certain structure of code. It comprises a sequence of instructions with exactly one entry point and exactly one exit point. $V = \{i_1, .., i_k\}$ denotes a set of instructions. k is the number of instructions in a basic block. For all two arbitrary instructions $i_x \in V$ and $i_y \in V$ $(1 \leq x \leq k$, $1 \leq x \leq k$, $x \neq y)$, $x < y$ means that i_x is before i_y in the sequence of instructions. n is the number of all basic blocks in a program and $B = \{b_i | 1 \leq i \leq n\}$ is the set of all basic blocks. An instruction is assigned to exactly one basic block b_i [90].

Definition 6 *(Control Flow Graph)*
A control flow graph is a graph $D = \langle V, E\rangle$ where the set of nodes V comprises all basic blocks $\{b_1, .., b_n\}$ and n is the number of all basic blocks in a program. The edge set $V \subseteq E \times E$ represents the control flow dependencies between basic blocks [90].

Definition 7 *(Data Dependency Graph)*
A data dependency graph is a graph $D = \langle V, E\rangle$, where $V = \{i_1, .., i_m\}$ denotes a set of instructions. m is the number of instructions. The edges $E \subseteq V \times V$ denote the data dependencies (RAW (Read After Write), WAR (Write After Read), WAW (Write After Write)) between instructions [90].

Definition 8 *(Program Dependency Graph)*
A program dependency graph $D = \langle V, E\rangle$ comprises data dependencies and control flow dependencies for all instructions of a program. The set of all instructions of one program is $V = \{i_1, .., i_n\}$ and $E \subseteq V \times V$ are the data dependencies or control flow dependencies between instructions [90].

Definition 9 *(Extended Basic Block)*
An EBB (Extended Basic Blocks) is a subset of all basic blocks $N \subseteq B$ and they this set of basic blocks is ordered according to the control flow D. N has exactly one entry point but multiple exit points. EBBs are the atomic scheduling domain for a global instruction scheduler [90], which means that instructions can only be moved – constrained by dependencies in the program dependency graph – inside a single EBB.

A.2 Mathematical Symbols & Style Sheet

Units

Physical units and digital information units are denoted in this thesis by the following unit symbols:

- Second: s
- Millisecond: ms
- Watt: W
- Joule: J
- Volt: V
- Ampere: A
- Byte: B.

In addition to that, standard unit prefixes are allowed.

Scalars & Vectors

Scalar values are written by Latin letters (a,b) or abbreviations (pos,$asap$). Vectors are marked by writing them boldface ($\mathbf{a}$,$\mathbf{b}$). Scalars having the same characteristic are extended by an index (a_i,a_j) or identifier (a_{id1}). Time stamps t_{id1}^{ido1}, energy consumption values E_{id1}^{ido1} and runtime values r_{id1}^{ido1} are denoted by a special format, because they can include further identifiers. In addition to that physical values such as P or R are denoted in capital letters.

Functions

Scalar functions are characterized by Latin letters f or also by abbreviations (max, min) having index (f_a,f_b) or identifier (f_{id},f_{ed}). Vector functions are marked boldface ($\mathbf{f_a}$,$\mathbf{f_b}$).

Accentuations

Proper names such as benchmarks and self created terms such as introduced methods are identified in *italics*.

List of Tables

List of Figures

List of Listings

Bibliography

[1] Advanced Micro Devices. AMD Embedded G-Series Platform. http://www.amd.com/us/Documents/49282_G-Series_platform_brief.pdf, 2012. [Cited on pages 3 and 5]

[2] Alippi, C., Anastasi, G., Galperti, C., Mancini, F., and Roveri, M. Adaptive Sampling for Energy Conservation in Wireless Sensor Networks for Snow Monitoring Applications. In *Proceedings of the Internatonal Conference on Mobile Adhoc and Sensor Systems (MASS)* (2007), IEEE, pp. 1–6. [Cited on page 117]

[3] Almesberger, W. Linux Network Traffic Control - Implementation Overview. http://www.almesberger.net/cv/papers/tcio8.pdf, 1999. [Cited on page 114]

[4] Alonso, A., Berjon, D., Conan, V., Foley, C., Hagen, M., Lavaux, D., Lévay, A., Marwedel, P., Michaelis, S., Schäfer, V., Schmutzler, J., Schneider, C., Timm, C., and Wietfeld, C. Deliverable D 2.1 - Architecture and Services. http://www.ist-more.org/images/stories/d2.1_architectureandservices.pdf, 2007. MORE Consortium. [Cited on pages 9, 10, 99, 100, 101, 103, and 116]

[5] Alonso, A., Salazar, E., and L´opez, J. Resource Management for Enhancing Predictability in Systems with Limited Processing Capabilities. In *Proceedings of the Conference on Emerging Technologies and Factory Automation (ETFA)* (2010), IEEE, pp. 1–7. [Cited on pages 10, 99, 103, 116, 117, and 118]

[6] Altera Corporation. Implementing FPGA Design with the OpenCL Standard. http://www.altera.com/literature/wp/wp-01173-opencl.pdf, 2011. [Cited on page 26]

[7] Amdahl, G. Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. In *Proceedings of the Spring Joint Computer Conference* (1967), AFIPS, pp. 483–485. [Cited on page 73]

[8] Arkin, A., Askary, S., Barreto, C., Bloch, B., Curbera, F., Ford, M., Goland, Y., Guízar, A., Kartha, N., et al. Web Services Business Process Execution Language Version (WS-BPEL). http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf, 2007. OASIS Standard. [Cited on page 95]

[9] Autili, M., Di Benedetto, P., and Inverardi, P. Context-Aware Adaptive Services: The PLASTIC Approach. In *Fundamental Approaches to*

Software Engineering (2009), Lecture Notes in Computer Science, Springer, pp. 124–139. [Cited on pages 4, 102, and 119]

[10] Bajaj, S., Box, D., Chappell, D., Curbera, F., et al. Web Services Policy (WS-Policy). `http://www.w3.org/TR/ws-policy/`, 2006. W3c Standard. [Cited on page 119]

[11] Bakhkhat, S., Böde, F., and Brucke, M. Eingebettete Systeme - Ein strategisches Wachstumgsfeld für Deutschland. `http://www.bitkom.org/files/documents/EingebetteteSysteme_web.pdf`, 2012. BITKOM Technology Roadmap. [Cited on page 2]

[12] Banerjia, S., Havanki, W. A., and Conte, T. M. Treegion Scheduling for Highly Parallel Processors. In *Proceedings of the European Conference on Parallel Processing (Euro-Par)* (1997), IEEE, pp. 1074–1078. [Cited on page 62]

[13] Bellwood, T., Capell, S., et al. Universal Description Discovery & Integration (UDDI). `http://uddi.org/pubs/uddi_v3.htm`, 2004. OASIS Standard. [Cited on page 91]

[14] Belussi, L., and Hirata, N. Fast QR Code Detection in Arbitrarily Acquired Images. In *Proceedings of the Conference on Graphics, Patterns and Images (SIBGRAPI)* (2011), IEEE, pp. 281–288. [Cited on pages 14 and 15]

[15] Benini, L., Lombardi, M., Milano, M., and Ruggiero, M. A Constraint Programming Approach for Allocation and Scheduling on the CELL Broadband Engine. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP)* (2008), Springer, pp. 21–35. [Cited on page 45]

[16] Benini, L., and Micheli, G. D. *Dynamic Power Management - Design Techniques and CAD Tools.* Kluwer, 1998. [Cited on page 117]

[17] Blum, C., and Merkle, D., Eds. *Swarm Intelligence: Introduction and Applications.* Natural Computing Series. Springer, 2008. [Cited on page 32]

[18] Bohn, H., Bobek, A., and Golatowski, F. WS-BPEL Process Compiler for Resource-Constrained Embedded Systems. In *Proceedings of the International Conference on Advanced Information Networking and Applications (AINAW)* (2008), IEEE, pp. 1387–1392. [Cited on pages 110 and 111]

[19] Bohn, H., Golatowski, F., and Timmermann, D. Dynamic Device and Service Discovery Extensions for WS-BPEL. In *Proceedings of International Conference on Service Systems and Service Management* (2008), IEEE, pp. 1–6. [Cited on pages 110, 111, and 113]

[20] Bondhugula, U., Baskaran, M., Krishnamoorthy, S., Ramanujam, J., Rountev, A., and Sadayappan, P. Automatic Transformations for Communication-Minimized Parallelization and Locality Optimization in the Polyhedral Model. In *Proceedings of the Joint European Conferences on Theory and Practice of Software (ETAPS) and International conference on Compiler Construction (CC)* (2008), Lecture Notes in Computer Science, Springer, pp. 132–146. [Cited on pages 42 and 131]

[21] Bosschere, K., Luk, W., Martorell, X., Navarro, N., O'Boyle, M., Pnevmatikatos, D., Ramirez, A., Sainrat, P., Seznec, A., Stenström, P., and Temam, O. High-Performance Embedded Architecture and Compilation Roadmap. In *Transactions on High-Performance Embedded Architectures and Compilers*, Lecture Notes in Computer Science. Springer, 2007, pp. 5–29. [Cited on pages 3, 19, 38, 42, and 58]

[22] Bottaro, A. RFP 86 - DPWS Discovery Base Driver. `http://andre.bottaro.pagesperso-orange.fr/papers/rfp-86-DPWSDiscoveryBaseDriver.pdf`, 2007. OSGi Alliance Standard. [Cited on page 102]

[23] Bottaro, A., Simon, E., Seyvoz, S., and Gerodolle, A. Dynamic Web Services on a Home Service Platform. In *Proceedings of the International Conference on Advanced Information Networking and Applications (AINA)* (2008), IEEE, pp. 378–385. [Cited on page 102]

[24] Bouyssounouse, B., and Sifakis, J. *Embedded Systems Design: The ARTIST Roadmap for Research and Development.* Lecture Notes in Computer Science. Springer, 2005. [Cited on pages 3, 4, 19, and 118]

[25] Box, D., Cabrera, L. F., Critchley, C., et al. Web Services Eventing (WS-Eventing). `http://www.w3.org/Submission/WS-Eventing/`, 2006. W3C Standard. [Cited on page 94]

[26] Box, D., Christensen, E., Curbera, F., et al. Web Services Addressing (WS-Addressing). `http://www.w3.org/Submission/ws-addressing/`, 2004. W3C Standard. [Cited on page 93]

[27] Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H. F., Thatte, S., and Winer, D. Simple Object Access Protocol (SOAP). `http://www.w3.org/TR/soap/`, 2000. W3C Standard. [Cited on pages 91 and 92]

[28] Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., and Yergeau, F. Extensible Markup Language (XML). `http://www.w3.org/TR/2008/REC-xml-20081126/`, 2008. W3C Standard. [Cited on page 91]

[29] Buck, I., Foley, T., Horn, D., Sugerman, J., Fatahalian, K., Houston, M., and Hanrahan, P. Brook for GPUs: Stream Computing on Graphics Hardware. In *Proceedings of the International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)* (2004), IEEE, pp. 777–786. [Cited on page 26]

[30] Bui, P., and Brockman, J. Performance Analysis of Accelerated Image Registration using GPGPU. In *Proceedings of the Workshop on General Purpose Processing on Graphics Processing Units (GPGPU)* (2009), ACM, pp. 38–45. [Cited on page 44]

[31] Chan, S., Conti, D., Kaler, C., Kuehnel, T., Regnier, A., Roe, B., Sather, D., Schlimmer, J., Sekine, H., Thelin, J., et al. Device Profile for Web Services (DPWS). `http://schemas.xmlsoap.org/ws/2006/02/devprof/`, February 2006. OASIS Standard. [Cited on pages 91, 92, and 103]

[32] Chandrakasan, A., Sheng, S., and Brodersen, R. Low-Power CMOS Digital Design. *IEEE Journal of Solid-State Circuits* (1992), 473–484. [Cited on pages 21 and 120]

[33] Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J. W., Lee, S.-H., and Skadron, K. Rodinia: A Benchmark Suite for Heterogeneous Computing. In *Proceedings of the International Symposium on Workload Characterization (IISWC)* (2009), IEEE, pp. 44–54. [Cited on pages 17, 52, and 67]

[34] Chen, D., Wang, L., Wang, S., Xiong, M., von Laszewski, G., and Li, X. Enabling Energy-Efficient Analysis of Massive Neural Signals Using GPGPU. In *Proceedings of the International Conference on Cyber, Physical and Social Computing (CPSCom)* (2010), IEEE/ACM, pp. 147 –154. [Cited on page 44]

[35] Chen, T., Raghavan, R., and Dale, J. Cell Broadband Engine Architecture and Its First Implementation. *IBM Journal of Research and Development* (2005), 559–572. [Cited on page 45]

[36] Cho, S., and Melhem, R. Corollaries to Amdahl's Law for Energy. *IEEE Computer Architecture Letters* (2008), 25–28. [Cited on page 44]

[37] Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S. Web Services Description Language (WSDL). `http://www.w3.org/TR/wsdl`, 2001. W3C Standard. [Cited on pages 91 and 92]

[38] Culler, D. E., Singh, J. P., and Gupta, A. *Parallel Computer Architecture - A Hardware/Software Approach.* Morgan Kaufmann, 1999. [Cited on page 27]

[39] D'Ambrogio, A. A WSDL Extension for Performance-Enabled Description of Web Services. In *Proceedings of the International Conference on Computer and Information Sciences (ISCIS)*, Lecture Notes in Computer Science. Springer, 2005, pp. 371–381. [Cited on pages 93, 119, and 121]

[40] Davies, N., Friday, A., Blair, G. S., and Cheverst, K. Distributed Systems Support for Adaptive Mobile Applications. *Mobile Network Applications* (1996), 399–408. [Cited on page 117]

[41] Davis, D., Malhotra, A., Warr, K., and Chou, W. Web Services Metadata Exchange (WS-MetadataExchange). `http://www.w3.org/TR/2009/WD-ws-metadata-exchange-20090317/`, 200. W3C Standard. [Cited on page 94]

[42] de Souza, L. M. S., Spiess, P., Guinard, D., Köhler, M., Karnouskos, S., and Savio, D. SOCRADES: A Web Service Based Shop Floor Integration Infrastructure. In *Proceedings of the International Conference on The Internet of Things (IOT)* (2008), C. Floerkemeier, M. Langheinrich, E. Fleisch, F. Mattern, and S. E. Sarma, Eds., vol. 4952 of *Lecture Notes in Computer Science*, IEEE, pp. 50–67. [Cited on page 98]

[43] Deb, K., Agrawal, S., Pratap, A., and Meyarivan, T. A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II. In *Proceedings of the International Conference on Parallel Problem Solving From Nature (PPSN)* (2000), Lecture Notes in Computer Science, Springer, pp. 849–858. [Cited on pages 35 and 36]

[44] Denso Wave Incorporated. QR Code. `http://www.denso-wave.com/qrcode/index-e.html`, 2010. [Cited on pages 14, 15, 106, and 123]

[45] Diamos, G. F., Kerr, A. R., Yalamanchili, S., and Clark, N. Ocelot: A Dynamic Optimization Framework for Bulk-Synchronous Applications in Heterogeneous Systems. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)* (2010), ACM, pp. 353–364. [Cited on page 45]

[46] Domínguez, R., and Kaeli, D. R. Improving the Open64 Backend for GPUs. NVIDIA GPU Technology Conference - `http://www.roddomi.com/pubs/NVIDIASummitPoster.pdf`, 2009. [Cited on pages 44 and 50]

[47] Dorigo, M., and Di Caro, G. The Ant Colony Optimization Meta-Heuristic. In *New Ideas in Optimization*. McGraw-Hill, 1999, pp. 11–32. [Cited on page 32]

[48] Erickson, D., Mandal, S., Yang, A., and Cordovez, B. Nanobiosensors: Optofluidic, Electrical and Mechanical Approaches to Biomolecular De-

tection at the Nanoscale. *Journal of Microfluidics and Nanofluidics* (2008), 33–52. [Cited on page 11]

[49] Falk, H., and Marwedel, P. *Source Code Optimization Techniques for Data Flow Dominated Embedded Software.* Kluwer Academic Publishers, 2004. [Cited on page 33]

[50] Fiehe, C., Litvina, A., Luck, I., Dohndorf, O., Kattwinkel, J., Stewing, F.-J., Kruger, J., and Krumm, H. Location-Transparent Integration of Distributed OSGi Frameworks and Web Services. In *Proceedings of the International Conference on Advanced Information Networking and Applications Workshops (WAINA)* (2009), IEEE, pp. 464–469. [Cited on page 102]

[51] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. RFC 2616 - Hypertext Transfer Protocol (HTTP), 1999. [Cited on page 91]

[52] Fisher, J. Trace Scheduling: A Technique for Global Microcode Compaction. *IEEE Transactions on Computers* (1981), 478 –490. [Cited on page 63]

[53] Foley, C., Balasubramaniam, S., Botvich, D., Donnell, W., Schmutzler, J., Michaelis, S., and Stair, T. Distributed Pervasive Services using Group Service communication supporting Body Area Networks. In *Third International Conference on Body Area Network* (Tempe, Arizona, USA., 2008). [Cited on pages 10, 99, and 103]

[54] Foley, C., Power, G., Griffin, L., Chen, C., Donnelly, N., and de Leastar, E. Service Group Management Facilitated by DSL Driven Policies in Embedded Middleware. In *Proceedings of the Symposium on Computers and Communications (ISCC)* (2010), IEEE, pp. 483–488. [Cited on page 103]

[55] Franch, X., and Botella, P. Putting Non-Functional Requirements into Software Architecture. In *Proceedings of the International Workshop on Software Specification and Design (IWSSD)* (1998), IEEE, pp. 60–67. [Cited on page 116]

[56] Frigo, M., and Johnson, S. G. The Design and Implementation of FFTW3. *IEEE Special Issue on "Program Generation, Optimization, and Platform Adaptation" 93*, 2 (2005), 216–231. [Cited on page 17]

[57] Fürst, C., Lorz, C., and Makeschin, F. Challenges for Monitoring and the Use of Monitoring Data for Landscape Management from Point of View of the End-User. *Forests in a Changing Environment. Results of 20 years ICP Forests Monitoring* (2007), 54–59. [Cited on pages 16 and 105]

[58] Georgia Tech Research. Vector Signal Image Processing Library. `http://www.vsipl.org/software/`, 2010. [Cited on page 17]

[59] Görlich, M. Untersuchung und Verbesserung der Speicherzugriffsverteilung in GPGPU-Programmen unter Nutzung von lokalen Schedulingmethoden. Master's thesis, Embedded System Group, Faculty of Computer Science, TU Dortmund, 2011. [Cited on pages 47, 53, and 62]

[60] Griebl, M., and Lengauer, C. The Loop Parallelizer LooPo. In *Proceedings of the Workshop on Compilers for Parallel Computers* (1996), Austrian Center for Parallel Computation, pp. 311–320. [Cited on pages 42 and 131]

[61] Group, K. OpenCL Specification. `http://www.khronos.org/opencl/`, Juli 2010. Khronos Standard. [Cited on pages 29, 43, and 123]

[62] Hackmann, G., Gill, C., and Roman, G.-C. Extending BPEL for Interoperable Pervasive Computing. In *Proceedings of the International Conference on Pervasive Services* (2007), IEEE, pp. 204–213. [Cited on pages 4 and 111]

[63] Hagen, M., Waligora, S., and othe. Deliverable D 2.1 -User Requirements Analysis. `http://www.ist-more.org/images/stories/d1.1_user_req_analysis_v1_0_prodv.pdf`, 2006. MORE Consortium. [Cited on pages 16 and 106]

[64] Haines, M., et al. Web Service Implementation Methodology. `http://www.oasis-open.org/committees/documents.php?wg_abbrev=fwsi/`, 2005. OASIS Standard. [Cited on pages 94 and 95]

[65] Han, T. D., and Abdelrahman, T. S. Reducing Branch Divergence in GPU Programs. In *Proceedings of the Workshop on General Purpose Processing on Graphics Processing Units (GPGPU)* (2011), ACM, pp. 1–8. [Cited on pages 62 and 63]

[66] Hansmann, U., Stober, T., Merk, L., and Nicklous, M. *Pervasive Computing.* Springer, 2003. [Cited on page 1]

[67] Hennessy, J. L., and Patterson, D. A. *Computer Architecture - A Quantitative Approach.* Morgan Kaufmann, 2012. [Cited on page 28]

[68] Hong, S., and Kim, H. An Analytical Model for a GPU Architecture with Memory-level and Thread-Level Parallelism Awareness. In *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)* (2009), ACM/IEEE, pp. 152–163. [Cited on pages 46, 59, and 61]

[69] Intel Coporation. Introduction to Intel® OpenCL Tools. `http://software.intel.com/en-us/articles/introduction-to-intel-opencl-tools/`, 2011. [Cited on page 26]

[70] Jammes, F., Mensch, A., and Smit, H. Service-Oriented Device Communications Using the Devices Profile for Web services. In *Proceedings of the International Conference on Advanced Information Networking and Applications Workshops (AINAW)* (2007), IEE, pp. 947–955. [Cited on page 104]

[71] Keinert, J., Streubühr, M., Schlichter, T., Falk, J., Gladigau, J., Haubelt, C., Teich, J., and Meredith, M. SystemCoDesigner: An Automatic ESL Synthesis Approach by Design Space Exploration and Behavioral Synthesis for Streaming Applications. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* (2009), 1–23. [Cited on page 45]

[72] Kerns, D. R., and Eggers, S. J. Balanced Scheduling: Instruction Scheduling when Memory Latency is Uncertain. In *Proceedings of the Conference on Programming Language Design and Implementation (PLDI)* (2004), ACM/SIGPLAN, pp. 515–527. [Cited on page 44]

[73] Kung, S. Y., Kailath, T., and Whitehouse, H. J. *VLSI and Modern Signal Processing.* Prentice Hall, 1984. [Cited on page 63]

[74] Laumanns, M., Zitzler, E., and Thiele, L. A Unified Model for Multi-objective Evolutionary Algorithms with Elitism. In *Proceedings of the Congress on Evolutionary Computation* (2000), IEEE, pp. 46–53. [Cited on page 25]

[75] Lavaux, D., Schmutzler, J., Timm, C., and Michaelis, S. MORE Middleware and Services - User Guide. Tech. rep., EU Project MORE, 2009. [Cited on pages 10, 99, and 103]

[76] Lee, E. A. Computing Foundations and Practice for Cyber-Physical Systems: A Preliminary Report. Tech. rep., EECS Department, University of California, Berkeley, 2007. [Cited on page 1]

[77] Lee, S. P., Chan, L. P., and Lee, E. W. Web Service Implementation Methodology for SOA Application. In *Proceedings of the International Conference on Industrial Informatics (IINDIN)* (2006), IEEE, pp. 335–340. [Cited on page 94]

[78] Leupers, R. Instruction Scheduling for Clustered VLIW DSPs. In *Proceedings of the International Conference on Parallel Architecture and Compilation Techniques (PACT)* (2000), IEEE, pp. 291–300. [Cited on page 44]

[79] Libuschewski, P., Weichert, F., and Timm, C. Parameteroptimierte und GPGPU-basierte Detektion viraler Strukturen innerhalb Plasmonenunterstützter Mikroskopiedaten. In *Proccedings of the Workshop Bildverar-*

beitung für die Medizin (BVM) (2012), Lectures Notes on Computer Science, Springer, pp. 237–242. [Cited on page 9]

[80] Liu, S.-M. Open64 Release 4.0: High Performance Compiler for Itanium and x86 Linux. In *Proceedings of the Conference on Programming Language Design and Implementation (PLDI)* (2007), ACM/SIGPLAN. [Cited on pages 50 and 64]

[81] Luke, S. *Essentials of Metaheuristics.* Lulu, 2009. [Cited on pages 32, 33, and 49]

[82] Ma, X., Dong, M., Zhong, L., and Deng, Z. Statistical Power Consumption Analysis and Modeling for GPU-based Computing. In *Proceedings of the Workshop on Power Aware Computing and Systems (HotPower)* (2009), ACM. [Cited on pages 21 and 43]

[83] Machanick, P. Approaches to Addressing the Memory Wall. Tech. rep., School of IT and Electrical Engineering, University of Queensland, 2002. `http://www.itee.uq.edu.au/~philip/Publications/Techreports/2002/Reports/memory-wall-survey.pdf`. [Cited on page 58]

[84] Marwedel, P. *Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems.* Springer, 2011. [Cited on pages 1, 2, 20, 35, 41, and 120]

[85] Marwedel, P., and Engel, M. Plea for a Holistic Analysis of the Relationship between Information Technology and Carbon-Dioxide Emissions. In *Proceedings of the Workshop on Energy-Aware Systems and Methods (GI-ITG)* (2010). [Cited on page 19]

[86] Melzer, I. *Service-orientierte Architekturen mit Web Services - Konzepte, Standards, Praxis.* Spektrum Akademischer Verlag, 2007. [Cited on pages 89, 90, and 91]

[87] Michaelis, S., Wolff, A., Schmutzler, J., and Timm, C. MORE - Architecture and Services, Public Deliverable 2.1. Tech. rep., EU Project MORE, Dortmund, Germany, 2007. [Cited on pages 10, 99, and 103]

[88] Modi, V., and Kemp, D. Web Services Dynamic Discovery (WS-Discovery). `http://docs.oasis-open.org/ws-dd/discovery/1.1/wsdd-discovery-1.1-spec.html`, 2009. OASIS Standard. [Cited on page 93]

[89] Molnar, F., Szakaly, T., Meszaros, R., and Lagzi, I. Air Pollution Modelling using a Graphics Processing Unit with CUDA. *Computer Physics Communications* (2010), 105 –112. [Cited on page 17]

[90] Muchnick, S. S. *Advanced Compiler Design Implementation.* Morgan Kaufmann, 1997. [Cited on pages 6, 31, and 133]

[91] Munir, M. F., and Filali, F. Maximizing Network-Lifetime in Large Scale Heterogeneous Wireless Sensor-Actuator Networks: A Near-Optimal Solution. In *Proceedings of the Workshop on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks (PE-WASUN)* (2007), ACM, pp. 62–69. [Cited on page 117]

[92] Nadalin, A., Kaler, C., Monzillo, R., and Hallam-Baker, P. WS-Security Core Specification. `http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf`, 2009. OASIS Standard. [Cited on page 94]

[93] Nere, A., and Lipasti, M. Cortical Architectures on a GPGPU. In *Proceedings of the Workshop on General-Purpose Computation on Graphics Processing Units (GPGPU)* (2010), ACM. [Cited on page 44]

[94] Nikolov, H., Thompson, M., Stefanov, T., Pimentel, A., Polstra, S., Bose, R., Zissulescu, C., and Deprettere, E. Daedalus: Toward Composable Multimedia MP-SoC Design. In *Proceedings of the Annual Design Automation Conference (DAC)* (2008), ACM, pp. 574–579. [Cited on page 45]

[95] Nugteren, C. Improving CUDA's Compiler through the Visualization of Decoded GPU Binaries. Master's thesis, Electronic Systems Group, Faculty of Electrical Engineering, Eindhoven University of Technology, 2009. [Cited on page 44]

[96] Nvidia Corporation. NVIDIA and Audi Marry Silicon Valley Technology with German Engineering. `http://www.nvidia.com/object/io_1262839759949.html`, 2010. [Cited on pages 26, 29, and 30]

[97] Nvidia Corporation. CUDA Architecture - Introduction & Overview. `http://developer.download.nvidia.com/compute/cuda/docs/CUDA_Architecture_Overview.pdf`, 2012. [Cited on pages 17, 30, 38, 52, 53, and 67]

[98] Nvidia Corporation. CUDA Programming Guide. `http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf`, 2012. [Cited on pages 29, 47, and 59]

[99] Nvidia Corporation. CUDA Toolkit. `http://developer.nvidia.com/cuda-downloads`, 2012. [Cited on page 17]

[100] Nvidia Corporation. NVIDIA GeForce. `http://www.nvidia.de/object/geforce_family_de.html`, 2012. [Cited on page 24]

[101] Nvidia Corporation. The CUDA Compiler Driver NVCC. `http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/nvcc.pdf`, 2012. [Cited on pages 30 and 31]

[102] Ohbuchi, E., Hanaizumi, H., and Hock, L. Barcode Readers using the Camera Device in Mobile Phones. In *Proceedings of the International Conference on Cyberworlds* (2004), IEEE, pp. 260–265. [Cited on pages 14 and 15]

[103] Pavlovski, C. J., and Zou, J. Non-Functional Requirements in Business Process Modeling. In *Proceedings of the Asia-Pacific Conference on Conceptual Modelling (APCCM)* (2008), Australian Computer Society, pp. 103–112. [Cited on page 115]

[104] Phivakumar, P., and Jouppi, N. P. CACTI 3.0: An Integrated Cache Timing, Power, and Area Model. Tech. rep., HP Labs, 2001. `http://www.hpl.hp.com/techreports/Compaq-DEC/WRL-2001-2.pdf`. [Cited on page 21]

[105] Postel, J. RFC 768 - User Datagram Protocol (UDP), 1980. [Cited on page 91]

[106] Postel, J. RFC 793 - Transmission Control Protocol (TCP), 1981. [Cited on page 91]

[107] PROFIBUS Nutzerorganisation e.V. Interbus. `http://www.interbusclub.com/`, 2012. [Cited on page 98]

[108] Prosyst Software GmbH. mBS Telematics - OSGi for Automotive Sector, 2012. [Cited on page 91]

[109] Pulli, K. OpenCL in Handheld Devices. In *Proceedings of Annual Hot Chips Conference* (2009), IEEE. [Cited on page 43]

[110] Ren, D., and Suda, R. Power Efficient Large Matrices Multiplication by Load Scheduling on Multi-core and GPU Platform with CUDA. *Proceedings of the IEEE International Conference on Computational Science and Engineering* (2009), 424–429. [Cited on page 44]

[111] Restrepo-Zea, A., Seneclauze, M., Decotignie, J.-D., Oliver, R. S., Fohler, G., Steffens, L., Geilen, M., Chinta, A., Weffers-Albu, A., Koulamas, C., et al. Deliverable D7 - Model Composition and End-to-end Prediction. `http://www.hitech-projects.com/euprojects/betsy/deliverables/betsy_deliverable_d7_final_v3.0.pdf`, 2007. BETSY Consortium. [Cited on page 118]

[112] Retail Banking Research. Global ATM Market and Forecasts to 2016. `http://www.rbrlondon.com/reports/G2016_Brochure.pdf`, 2012. [Cited on page 19]

[113] Risco-Martín, J. Java Evolutionary COmputation Library (JECO). https://sourceforge.net/projects/jeco, 2010. [Cited on pages 52 and 66]

[114] Rofouei, M., Stathopoulos, T., Ryffel, S., Kaiser, W., and Sarrafzadeh, M. Energy-Aware High Performance Computing with Graphic Processing Units. In *Proceedings of the Workshop on Power Aware Computing and Systems (HotPower)* (2008), IEEE. [Cited on page 44]

[115] Rong, L., Fredj, M., Issarny, V., and Georgantas, N. Mobility management in B3G networks: a middleware-based approach. In *Proceedings of the International Workshop on Engineering of Software Services for Pervasive Environments (ESSPE)* (2007), ACM, pp. 41–45. [Cited on page 102]

[116] Schmutzler, J., Bieker, U., and Wietfeld, C. Network-centric Middleware supporting dynamic Web Service Deployment on heterogeneous Embedded Systems. In *Proceedings of the International Conference on Concurrent Enterprising* (2008), ICE. [Cited on pages 10, 99, 102, 103, 104, and 105]

[117] Schneider Electric. DPWS4J Toolkit. `https://forge.soa4d.org/projects/dpws4j/`, 2011. [Cited on page 113]

[118] Seneclauze, M., Decotignie, J.-D., van der Stok, P., de Groot, H., van Hartskamp, M., van Doren, G., van Heesch, D., Perez, C. O., Joosten, M., Blanch, C., et al. The BETSY Project on Timeliness and Energy Aspects of Video Streaming. In *Proceedings of the International Workshop on Wireless Ad-hoc Networks (IWWAN)* (June 2005). [Cited on page 118]

[119] Shalf, J., Bashor, J., Patterson, D., Asanovic, K., Yelick, K., Keutzer, K., and Mattson, T. The MANYCORE Revolution: Will HPC LEAD or FOLLOW? *SciDAC Review* (2009), 40–49. [Cited on page 3]

[120] Sharaf, A., Beaver, J., Labrinidis, A., and Chrysanthis, K. Balancing Energy Efficiency and Quality of Aggregate Data in Sensor Networks. *VLDB Journal* (2004), 384–403. [Cited on page 117]

[121] Siedhoff, D., Weichert, F., Libuschewski, P., and Timm, C. Detection and Classification of Nano-Objects in Biosensor Data. In *Proceedings of the International Workshop on Microscopic Image Analysis with Applications in Biology (MIAAB)* (2011). [Cited on page 9]

[122] Simunic, T., Benini, L., and De Micheli, G. Cycle-Accurate Simulation of Energy Consumption in Embedded Systems. In *Proceedings of the Annual Design Automation Conference (DAC)* (1999), ACM, pp. 867–872. [Cited on page 21]

[123] Steinke, S., Knauer, M., Wehmeyer, L., and Marwedel, P. An Accurate and Fine Grain Instruction-Level Energy Model Supporting Software Optimizations. In *Proceedings of the International Workshop on Power And Timing Modeling, Optimization and Simulation (PAMOS)* (2001), Lecture Notes in Computer Science, Springer. [Cited on page 21]

[124] Taisch, M., Colombo, A. W., Karnouskos, S., and Cannata, A. SOCRADES Technology Roadmap, 2008. SOCRADES Consortium. [Cited on pages 3, 4, 98, 100, 116, and 127]

[125] Texas Advanced Computing Center. GotoBLAS2. `http://www.tacc.utexas.edu/tacc-projects/gotoblas2`, 2010. [Cited on page 17]

[126] Texas Instruments. *CMOS Power Consumption and C_{pd} Calculation*, 1997. `www.ti.com/lit/an/scaa035b/scaa035b.pdf`. [Cited on pages 21 and 120]

[127] The OSGi Alliance. OSGi Service Platform Core Specification. `http://www.osgi.org/Specifications`, 2007. [Cited on pages 90 and 91]

[128] Thiele, L. Design Space Exploration of Embedded Systems. Artist Network of Excellence on Embedded System Design - Spring School Embedded System in Xi'an, 2006. [Cited on page 41]

[129] Timm, C., Gelenberg, A., Marwedel, P., and Weichert, F. Energy Considerations within the Integration of General Purpose GPUs in Embedded Systems. In *Proceedings of the Annual International Conference on Advances in Distributed and Parallel Computing (ADPC)* (2010), GSTF. [Cited on page 9]

[130] Timm, C., Gelenberg, A., Weichert, F., and Marwedel, P. Reducing the Energy Consumption of Embedded Systems by Integrating General Purpose GPUs. Tech. Rep. 829, Embedded System Group, Faculty of Computer Science, TU Dortmund, 2010. [Cited on page 9]

[131] Timm, C., Görlich, M., Weichert, F., Marwedel, P., and Müller, H. Feedback-Based Global Instruction Scheduling for GPGPU Applications. In *Proceedings of the ICCSA Workshop on Advances in High Performance Algorithms and Applications (AHPAA)* (2012), Lecture Notes on Computer Science, Springer. [Cited on page 9]

[132] Timm, C., Libuschewski, P., Siedhoff, D., Weichert, F., Müller, H., and Marwedel, P. Improving Nanoobject Detection in Optical Biosensor Data. In *Proceedings of the International Symposium on Bio- and Medical Information and Cybernetics (BMIC)* (2011), pp. 236–240. [Cited on page 9]

[133] Timm, C., Michaelis, S., Marwedel, P., Schmutzler, J., Seger, J., Wietfeld, C., and Wolff, A. Deliverable D 2.3 - Performance and System Model, 2007. [Cited on pages 9, 10, 99, and 103]

[134] Timm, C., Schmutzler, J., Marwedel, P., and Wietfeld, C. Dynamic Web Service Orchestration applied to the Device Profile for Web Services in Hierarchical Networks. In *Proceedings of the International ICST Conference on COMmunication System softWAre and middlewaRE (COMSWARE)* (2009), ACM, pp. 1–6. [Cited on pages 9, 10, and 105]

[135] Timm, C., Weichert, F., Fiedler, D., Prasse, C., Müller, H., ten Hompel, M., and Marwedel, P. Decentralized Control of a Material Flow System enabled by an Embedded Computer Vision System. In *Proceedings of IEEE ICC Workshop on Embedding the Real World into the Future Internet (RWFI)* (2011), IEE, pp. 1–5. [Cited on pages 9 and 13]

[136] Timm, C., Weichert, F., Marwedel, P., and Müller, H. Design Space Exploration Towards a Realtime and Energy-Aware GPGPU-based Analysis of Biosensor Data. In *Special Issue "International Conference on Energy-Aware High Performance Computing (ENA-HPC)"*, Computer Science - Research and Development. Springer, 2011, pp. 1–9. [Cited on page 9]

[137] Timm, C., Weichert, F., Marwedel, P., and Müller, H. Multi-Objective Local Instruction Scheduling for GPGPU Applications. In *Proceedings of the International Conference on Parallel and Distributed Computing Systems (PDCS)* (2011), IASTED/ACTA Press. [Cited on page 9]

[138] Timm, C., Weichert, F., Prasse, C., Müller, H., ten Hompel, M., and Marwedel, P. Efficient Resource Management in Sensor/Actuator Networks based on Non-Functional Requirement Specifications. In *Proceedings of the International Network Conference (INC)* (2012), Centre for Security, Communications and Network Research, Plymouth University. [Cited on page 9]

[139] Tiwari, V., Malik, S., and Wolfe, A. Power Analysis of Embedded Software: A First Step towards Software Power Minimization. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)* (1994), IEEE/ACM, pp. 384–390. [Cited on page 21]

[140] Toshiba. TC90413XBG Single-chip SoC for ATSC LCD TV. `http://www.toshiba.com/taec/components/docs/ProdBrief/07K04__TC90413XBG.pdf`, 2007. [Cited on page 26]

[141] Tseng, C.-J., and Siewiorek, D. Automated Synthesis of Data Paths in Digital Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (1986), 379–395. [Cited on page 63]

[142] Valliiee, M., Ramparany, F., and Vercouter, L. Flexible Composition of Smart Device Services. In *Proceeding of the International Conference on Pervasive Systems and Computing (PSC)* (2005), CSREA Press, pp. 27–30. [Cited on pages 4 and 102]

[143] Valluri, M., and John, L. Is Compiling for Performance == Compiling for Power? In *Proceedings of the Workshop on Interaction between Compilers and Computer Architectures (INTERACT)* (2001), Kluwer Academic Publishers, pp. 101–117. [Cited on page 44]

[144] Vardhan, V., Sachs, D., Yuan, W., III, A. F. H., Adve, S., Jones, D., Kravets, R., and Nahrstedt, K. GRACE-2: Integrating Fine-Grained Application Adaptations with Global Adaptation for Saving Energy. *International Journal of Embedded Systems (IJES)* (2005). [Cited on page 118]

[145] Voorneveld, M. Characterization of Pareto Dominance. *Operations Research Letters* (2003), 7–11. [Cited on page 34]

[146] Wang, Z., and Hu, X. S. Energy-Aware Variable Partitioning and Instruction Scheduling for Multibank Memory Architectures. *ACM Transactions on Design Automation of Electronic Systems* (April 2005), 369–388. [Cited on page 44]

[147] Weichert, F., Fiedler, D., Hegenberg, J., Müller, H., Prasse, C., Roidl, M., and ten Hompel, M. Marker-Based Tracking in Support of RFID Controlled Material Flow Systems. *Journal of Logistics Research* (2010), 13–21. [Cited on page 13]

[148] Weichert, F., Gaspar, M., Timm, C., Zybin, A., Gurevich, E., Engel, M., Müller, H., and Marwedel, P. Signal Analysis and Classification for Plasmon Assisted Microscopy of Nanoobjects. Tech. Rep. 830, Graphical Systems Group and Embedded System Group and , Faculty of Computer Science, TU Dortmund, 2010. [Cited on page 9]

[149] Weichert, F., Gaspar, M., Timm, C., Zybin, A., Gurevich, E., Engel, M., Müller, H., and Marwedel, P. Signal Analysis and Classification for Surface Plasmon Assisted Microscopy of Nanoobjects. *Sensors and Actuators B: Chemical* (2010), 281–290. [Cited on pages 9, 11, 12, 13, 14, and 73]

[150] Weichert, F., Gaspar, M., Zybin, A., Gurevich, E., Görtz, A., Timm, C., Müller, H., and P., M. Plasmonen-unterstützte Mikroskopie zur Detektion von Viren. In *Proceedings of the Workshop Bildverarbeitung für die Medizin (BVM)* (2010), Lecture Notes on Computer Science, Springer, pp. 76–80. [Cited on page 9]

[151] Weichert, F., Timm, C., Gaspar, M., Zybin, A., Gurevich, E. L., Müller, H., and Marwedel, P. GPGPU-basierte Echtzeitdetektion von Nanoobjekten mittels Plasmonen-unterstützter Mikroskopie. In *Proceedings of the Workshop Bildverarbeitung für die Medizin (BVM)* (2011), Lecture Notes on Computer Science, Springer, pp. 39–43. [Cited on pages 9 and 13]

[152] Weiser, M. The Computer for the 21st Century. In *Human-Computer Interaction.* Morgan Kaufmann, 1995, pp. 933–940. [Cited on page 1]

[153] Wolff, A., Michaelis, S., Schmutzler, J., and Wietfeld, C. Network-Centric Middleware for Service Oriented Architectures across Heterogeneous Embedded Systems. In *Proceedings of the International Enterprise Distributed Object Computing Conference (EDOC)* (2007), IEEE, pp. 105–108. [Cited on pages 10, 99, 100, 101, and 103]

[154] Woo, D., and Lee, H.-H. Extending Amdahl's Law for Energy-Efficient Computing in the Many-Core Era. *Computer* (2008), 24–31. [Cited on pages 3 and 44]

[155] Yi, W., Tang, Y., Wang, G., and Fang, X. A Case Study of SWIM: Optimization of Memory Intensive Application on GPGPU. In *Proceedings of the International Symposium on Parallel Architectures, Algorithms and Programming (PAAP)* (2010), IEEE, pp. 123–129. [Cited on page 44]

[156] Zeeb, E., Bobek, A., Bohn, H., Prüter, S., Pohl, A., and Krumm, H. WS4D: SOA-Toolkits making Embedded Systems ready for Web Services. In *Proceedings on the International Workshop on Open Source Software and Productlines (SPLC-OSSPL)* (2007). [Cited on pages 92 and 93]

[157] ZiiLABS. ZMS-08: Media Rich Applications Processor. `http://www.ziilabs.com/downloads/PB_ZiiLABS_ZMS-08.pdf`, 2011. [Cited on pages 3, 5, 41, and 43]

[158] Zitzler, E., Giannakoglou, K., Tsahalis, D., Periaux, J., Papailiou, K., Fogarty, T., Ler, E. Z., Laumanns, M., and Thiele, L. SPEA2: Improving the Strength Pareto Evolutionary Algorithm For Multiobjective Optimization. In *Proceedings of the International Conference on Evolutionary and Deterministic Methods for Design, Optimization and Control with Applications to Industrial and Societal Problems (EUROGEN)* (2001), International Center for Numerical Methods in Engineering, pp. 95–100. [Cited on pages 36, 45, and 49]

[159] Zybin, A., Kuritsyn, Y., Gurevich, E., Temchura, V., Überla, K., and Niemax, K. Real-time Detection of Single Immobilized Nanoparticles by Surface Plasmon Resonance Imaging. *Plasmonics* (2010), 31–35. [Cited on page 11]

Resource-Efficient Processing and Communication in Sensor/Actuator Environments

Abstract: The future of computer systems will not be dominated by personal computer like hardware platforms but by embedded and cyber-physical systems assisting humans in a hidden but omnipresent manner. These pervasive computing devices can, for example, be utilized in the home automation sector to create sensor/actuator networks supporting the inhabitants of a house in everyday life.

The efficient usage of resources is an important topic at design time and operation time of mobile embedded and cyber-physical systems. Therefore, this thesis presents methods which allow an efficient use of energy and processing resources in sensor/actuator networks. These networks comprise different nodes cooperating for a "smart" joint control function. Sensor/actuator nodes are typical cyber-physical systems comprising sensors/actuators and processing and communication components. Processing components of today's sensor nodes can comprise many-core chips.

This thesis introduces new methods for optimizing the code and the application mapping of the aforementioned systems and presents novel results with regard to design space explorations for energy-efficient and embedded many-core systems. The considered many-core systems are graphics processing units. The application code for these graphics processing units is optimized for a particular platform variant with the objectives of minimal energy consumption and/or of minimal runtime. These two objectives are targeted with the utilization of multi-objective optimization techniques. The mapping optimizations are realized by means of multi-objective design space explorations. Furthermore, this thesis introduces new techniques and functions for a resource-efficient middleware design employing service-oriented architectures. Therefore, a service-oriented architecture based middleware framework is presented which comprises a lightweight service orchestration. In addition to that, a flexible resource management mechanism will be introduced. This resource management adapts resource utilization and services to an environmental context and provides methods to reduce the energy consumption of sensor nodes.

Keywords: Design Space Exploration, Multi-objective Optimization, Optimizing Compiler, Service-Oriented Architectures, Device Profile for Web Services, Resource Management, Context-Awareness, Energy-Efficiency