POLITECNICO DI MILANO
DEPT. OF ELECTRONICS, INFORMATION AND BIOENGINEERING
DOCTORAL PROGRAM IN INFORMATION TECHNOLOGY

DESIGN SPACE EXPLORATION OF OPENCL APPLICATIONS ON HETEROGENEOUS PARALLEL PLATFORMS

Doctoral Dissertation of:
Edoardo Paone

Supervisor:
Prof. Cristina Silvano

Co-supervisors:
Prof. Gianluca Palermo
Prof. Vittorio Zaccaria

Tutor:
Prof. Andrea Bonarini

The Chair of the Doctoral Program:
Prof. Carlo Fiorini

2014 – XXVII

Abstract

PARALLEL programming is a skill which software engineers no longer can do without, since multi- and many-core architectures have been widely adopted for general-purpose computing platforms. In 2006 Intel introduced the first multi-core processor on the consumer market and, at the same time, NVIDIA unveiled CUDA, a programming paradigm to exploit Graphics Processing Units (GPUs) for general purpose computing.

Some years later (2008) the Khronos Consortium released the first specification of OpenCL, an open cross-platform API, inspired by CUDA, to efficiently exploit data-level parallelism while enabling application portability across different computing platforms. This API ensures functional portability of applications, so the same code can be compiled and executed on multi-core CPUs as well as GPGPUs, but also synthesized and deployed on FPGAs. However, additional fine-tuning of the application code might be needed in order to take the most performance out of a specific architecture.

Another important aspect for application optimization is the exploitation of architectural heterogeneity on modern computing platforms. The convergence to *globally heterogeneous locally homogeneous* parallel architectures, in both the embedded and High Performance Computing (HPC) domains, leads to a rapid overlapping of the challenges related to the efficient exploitation of the available computing devices. In particular, mapping of application tasks cannot be considered independently from specific optimization of each task, besides accounting for the overhead of data transfers and synchronization between different devices.

Code customization and task mapping represent the main challenge for

the design and optimization of OpenCL applications targeted to heterogeneous parallel platforms. At this aim, the Design Space Exploration (DSE) methodology presented in this thesis allows to efficiently explore the customization options of a parametric OpenCL application design. On the one hand, the proposed techniques reduce the exploration time on simulation platforms while providing close-to-optimal solutions; on the other hand, they exploit platform-specific constraints to prune out unfeasible solutions from the design space.

Task-level parallelism could be combined with request-level parallelism, in order to deploy and run multiple independent OpenCL applications on the same platform. This application scenario is enabled by the increasing number of cores integrated in the same chip, however modern platforms still lack run-time management to support applications with resource sharing. Thus, multi-application use cases often suffer from resource contention and performance degradation, especially under dynamic workload variations.

The contribution of this thesis consists of the DSE support to generate Pareto-optimal application configurations, with different trade-offs between performance and QoS. A run-time management technique is presented, which exploits the knowledge-base gathered at design-time to implement effective application auto-tuning. By including platform metrics and resource utilization in the optimization phase, this methodology also supports performance-aware scheduling on multi-core platforms and improves the overall system performance with respect to soft real-time constraints.

The proposed design methodology and runtime software layer have been implemented and demonstrated on a real case-study – an OpenCL stereo-matching application – targeting different industrial platforms.

Acknowledgments

Being admitted to the PhD program at Politecnico di Milano was a great opportunity for my academic and professional development. I am grateful to Cristina for having taken me under her supervision. She did a great job as a supervisor, by putting a lot of effort to guide me in the scientific research and contributing to my work with precious advice. In her group I also found two brilliant young professors, Gianluca and Vittorio. I worked very close to both of them, first in the 2PARMA European project and later in my PhD research, and this collaboration has brought the main results of this thesis. Thus, I would like to thank Gianluca and Vittorio for their great effort as co-supervisors of my PhD work.

During my PhD I met very talented and skilled people. In particular, my thoughts go to Patrick and Giuseppe, who did a great engineering work developing the *Barbeque* open source project. I really enjoyed contributing to their project and this collaboration produced the best of my outcomes, in terms of prototypes and demos. Moreover, I learned many new software engineering skills from Patrick, so I would like to thank him for his professional advice.

Another fruitful collaboration was the one with Prof. Ingo Sander from KTH. In particular, I would like to thank Ingo for his scientific support in one of the works presented in this thesis.

Finally, all my colleagues at Politecnico di Milano deserve my thanks for the nice work environment: many thanks to all of them.

Contents

CHAPTER 1

Introduction

The increasing computational parallelism provided by multi- and many-core architectures allowed to overcome the *power wall* in silicon technology, which limited the maximum operating frequency of single-core processors. However, the progress of software compilers and EDA tools was not fast enough to keep pace with such an increase of architectural complexity, more specifically they still provide limited support to the software developer to cope with parallelism. As a result, the computational capabilities of modern platforms are usually not fully utilized.

At the same time, the possibility to integrate special-purpose accelerators on the same chip enabled better energy efficiency, which in turn contributed to the diffusion of mobile computing devices in the electronics market. Thus, modern platforms also expose heterogeneity to the software developer, who has to decide the functionality mapping, and require *ad hoc* programming APIs to off-load data processing to specialized accelerators used as co-processors. This, together with the increasing number of cores, represents a burden for the software developer at the programming level – the so-called *programmability wall*. Moreover, the rapid change of the commercial platforms requires continuous porting of applications and software libraries to new hardware, which increases the development cost and represents a risk

for the time-to-market of the final product.

Therefore, a hot research topic aims at providing tools to ease application development for heterogeneous parallel platforms. It is a common view in the academic and industrial research [40] that the programmer should only express the concurrency of the application, and leave to the tools the mapping of this concurrency into the parallelism of the hardware. This thesis makes a step forward in this direction, targeting OpenCL [67] as a cross-platform programming paradigm for heterogeneous parallel platforms.

1.1 Motivation

The availability of mobile computing platforms on the market is continuously growing, leveraging a wide range of *smart* hand-hold devices (e.g. smartphones and tablets) and more recently even wearable ones, such as smartwatches and smart-glasses. The market numbers show that such devices are becoming more popular than desktop and laptop PCs, leading to a *post-PC* era [40]. This new era is characterized by pervasive connectivity, which enables to just "plug a device" into the Internet cloud wherever the user is, at home, at the office or even moving from home to the office.

Pervasive connectivity enables download of information and fruition of multimedia contents, data sharing, video and phone conferences, as well as collection of data from sensors and upload on the cloud. These features make a difference with respect to the previous generation of hand-hold devices, which were mainly designed for a single functionality. On the contrary, modern devices are *smart*, in the sense that they are multi-functional and provide higher computational capability, but still have to be regarded as embedded systems, because they are subject to a limited power budget. Thus, software applications should be optimized to get the maximum performance out of a specific platform.

Embedded computing devices represent the building blocks of the "Internet of Things" and cyber-physical systems [40], because they are used to interface the physical world with the digital world. Being connected to the cloud, the new generation of mobile computing platforms generates a huge amount of unstructured data, in the form of audio, video and physical measurements. Sending this data to a data-center to extract useful information is not a viable solution, because it requires high communication bandwidth not available today along with very expensive, power-hungry data-centers.

In this context, robust video analysis is likely to become one of the next killer applications of computing systems [40]. Currently, millions of surveillance cameras are producing video shootings that are never analyzed

because human observers are too expensive. Therefore, there is a need for surveillance systems with enough computational power to automatically analyze the input video and to request human intervention only when intrusion or abnormal behaviors are detected. Similar advancements are needed in other domains such as speech recognition, customer analytics and sensor data processing.

The progress in the manufacturing process of semiconductor devices enabled the integration of multiple processors on the same chip – the so-called Multi-Processor System-on-Chip (MPSoC). This platform allows the application developer to accelerate compute-intensive software tasks on specialized processors (e.g. Digital Signal Processors (DSPs) for audio and video encoding/decoding), while improving energy-efficiency. At the same time, MPSoCs can be customized by designing application-specific hardware and selecting which components to integrate (e.g., processors, memory, analog, wireless, sensors, MEMS), for more optimized devices.

The fabrication costs of customized MPSoCs are high, so this approach can be adopted only for large production volumes. For small volumes, the application developer has to target standard commercial platforms and to optimize the software, rather than designing a custom MPSoC. Thus, recent MPSoCs integrate multi-core processors to provide enough computational power for a variety of applications, for example Nvidia Tegra 4 includes a multi-core ARM-based CPU and a multi-core embedded GPU. This platform architecture exposes parallelism at two levels: i) in the form of task-mapping on multi-processors, which requires dealing with platform heterogeneity since each processor can be a specialized accelerator; and ii) as computational parallelism provided by each processor, being it a multi-core CPU or a GPU for general-purpose computing, which typically includes several homogeneous cores.

One main problem is the rapid change of the computing platforms available on the market, because new-generation products require porting of application code to different accelerators. At the same time, it might be difficult to exploit, at application-level, the increasing computational parallelism since there is still limited support for automatic optimization of parallel code. These two factors represent a *programmability wall*, which is the main source of complexity for the design of embedded systems today. New design tools are needed to provide better support for software optimization and cross-platform portability, instead of relying on hardware-software co-design of application-specific MPSoCs.

Another aspect related to application design is that the constantly growing computational power provided by multi- and many-core processors

can exceed the requirements of a single application. Therefore, in order to reduce design costs, multiple applications could be deployed and executed on the same platform. This solution requires run-time techniques to allocate system resources to applications, taking into account workload variability and dynamic requirements. Similar techniques are also needed to improve the system power consumption, as recently proposed by ARM with the "big.LITTLE" architecture or previously by Nvidia with the Optimus technology for automatic switching between integrated and discrete GPUs.

In conclusion, the last decade has witnessed a shift of the complexity in embedded systems design from the *hardware* to the *software*. The design of embedded software represents today the real added value of *smart* consumer electronics and *smart* sensors, while the present hardware is not yet fully exploited. Thus, after overcoming the *power wall* by adopting multi-core processors, academic research and industry have to address the so-called *programmability wall*.

1.2 Technological background

Today's multi-core CPUs implement a Symmetric Multi-Processing (SMP) architecture, where two or more identical processors – usually in a number power of two – are connected to a shared memory and controlled by a single Operating System (OS) instance. By means of OS support, it is possible to schedule different processes on parallel cores – a type of parallelism known as *request-level parallelism* [51]. Conversely, *parallel processing* consists of one process running on parallel cores, but this requires the process to spawn a pool of threads collaborating on a single task. Nowadays, there is no yet compiler that can automatically generate a multi-thread program from sequential application code (except for some specific patterns, e.g. loops), thus the application developer has to express parallelism in the source code.

Several programming models have been proposed for the development of multi-thread applications for parallel processing. For example, OpenMP [89] exploits code annotation (by means of C *pragmas*) to give hints to the compiler for parallelization; while Threading Building Blocks (TBB) [95], a C++ template library developed by Intel, uses compile-time constructs to express parallel tasks. Both OpenMP and TBB libraries abstract access to the multiple processors by allowing the operations to be treated as *tasks*, which are allocated to individual cores dynamically by the library run-time engine. This approach aims at decoupling the programming from the peculiarities of the SMP architecture, e.g. the number of cores, but it is limited to CPUs[1].

[1] The OpenMP support for heterogeneous platforms, introduced in version 4.0 [90] with the `target` directive,

Graphics Processing Units (GPUs) are specialized processors for graphics applications. This type of applications is compute-intensive and throughput-oriented, because the processing of large blocks of data (such as image rendering) can be done in parallel. To exploit data parallelism, GPUs have a highly parallel architecture, with a high number of stream processors and high bandwidth to the video memory. In order to use GPUs for general purpose computing, Nvidia introduced the Compute Unified Device Architecture (CUDA) programming language [88]. CUDA is a C API that allows a host application to access the GPUs as co-processors and offload compute-intensive tasks. It is widely used to accelerate not only image processing but also any application that exposes data-parallelism such as physics simulation, base-band processing in Software Defined Radio (SDR) [103] and cryptographic applications [50].

So far, the increasing number of processing cores integrated in one accelerator has been exploited to provide homogeneous computational parallelism. Nevertheless, in recent years we have witnessed to an ever increasing co-existence of different types of accelerators within the same platform, being it embedded, desktop or High Performance Computing (HPC). This results in a mix of *globally heterogeneous locally homogeneous* computational parallelism. From the very beginning, platform heterogeneity imposed a specialized programming API for each accelerator type (and eventually vendor, in case of proprietary API), such as CUDA, Intel TBB, or the IBM Cell BE SDK. This fragmentation has represented one of the major hurdles towards the development of portable, parallel applications. Recently this problem has been addressed by the Khronos Consortium with the proposal of a cross-platform programming model – the Open Computing Language (OpenCL) [67] – designed around the computational paradigm named *single program multiple data* (SPMD), a subcategory of *multiple instruction multiple data* (MIMD) [43].

OpenCL enables functional portability between different accelerators: multi-core CPUs, many-core NoC-based accelerators[2], DSPs[3] and FPGAs[4]. However, the OpenCL API does not ensure robust, cross-platform performance portability, e.g. in terms of throughput for streaming applications. On the one hand, OpenCL programs should be optimized for the target architecture by tuning platform-related parameters; on the other hand, task mapping over the available processing elements must be specific for each platform, which increases the development effort.

is still an abstraction layer over multiple offloading programming paradigms such as CUDA or OpenCL.

[2] See Adapteva Epiphany IP: http://www.adapteva.com/epiphany-multicore-intellectual-property/

[3] See Texas Instruments OpenCL support: http://processors.wiki.ti.com/index.php/OpenCL

[4] See Altera OpenCL SDK: http://www.altera.com/products/software/opencl/opencl-index.html

Another problem is that OpenCL does not provide an easy way to orchestrate task execution on multiple heterogeneous devices. This requires either design-time or run-time techniques to find the optimal mapping of application tasks to platform accelerators, while satisfying data-dependency constraints. Moreover, OpenCL provides an application runtime but does not support allocation of resources to separate applications (request-level parallelism [51]). Therefore, additional platform support for Run-Time Resource Management (RTRM) is needed in order to deploy multiple concurrent OpenCL applications on the same platform.

1.3 Thesis contributions

This thesis work has two main objectives:

1. **OpenCL application customization and optimization**
   This objective addresses one limitation of the OpenCL programming paradigm, namely the tight dependence of application performance on platform architectural details. Although the OpenCL API is cross-platform and generic enough to enable programming of different types of accelerators, customization of software parameters is necessary to achieve good performance when porting applications to a target platform. This problem is exacerbated by the intrinsic heterogeneity of modern computing platforms, which usually include two or more accelerators, eventually of different type, or can be extended by replacing plug-in modules (e.g. on the PCI bus of a general-purpose workstation). Thus, efficient task mapping on the available platform devices is important to maximize the throughput of an OpenCL application and to target the same application to different platforms.
   To address this problem, an optimization methodology is proposed, based on customization of a parametric application design. The methodology exploits Design Space Exploration (DSE) to identify the optimal solutions, with respect to multiple design objectives such as performance or Quality of Service (QoS).
   In this context, this thesis brings the following main contributions:

   - A customizable OpenCL application design providing a parameterized software pipeline, combined with a suitable exploration strategy for multi- and many-core platforms.
   - A technique to configure an ensemble model based on Artificial Neural Networks (ANNs), which allows to reduce the exploration time on simulation platforms for many-core architectures.

- A technique, supported by a *constraint solver*, to deal with platform constraints, which allows for fast pruning of the design space and efficient task mapping on heterogeneous parallel platforms.

2. **Application auto-tuning and run-time management**
   This second objective concerns the problem of resource sharing when multiple applications are deployed on the same computing platform. This type of parallelism – referred to as *request-level parallelism* – can be combined with *task-level parallelism*. However, platform resources should be properly assigned depending on application requirements in order to optimize the overall average system performance.
   The solution for performance-aware scheduling proposed in this thesis exploits offline application profiling to identify a set of optimal configurations, with different trade-offs between design objectives. This knowledge-base, obtained at *design-time*, is exploited at *run-time* for application auto-tuning and efficient resource allocation.
   The contributions of this thesis, with respect to run-time management, are listed below:

   - A Design Space Exploration (DSE) methodology to generate a set of Pareto-optimal application configurations, used at run-time for application dynamic auto-tuning and resource allocation.
   - A light-weight run-time management technique to optimize resource sharing for computationally intensive OpenCL applications.

The techniques listed above have been validated on different industrial platforms, described in Section 2.4, in order to demonstrate the applicability of the proposed methodology to real case studies. The implementation of the techniques exploits two external tools, MOST for the *design-time* phase and *Barbeque* for the *run-time* phase, developed at PoliMI, which will be introduced in Section 2.5 and Section 2.6, respectively. Conversely, a Stereo-Matching OpenCL application, used in several experiments, has been developed in this thesis and will be introduced in Section 2.3.

1.4 Thesis organization

The thesis is structured in two parts: Part I describes the contributions at **design-time**, concerning application optimization and customization, while Part II is about the contributions for **run-time** application auto-tuning and resource management. This organization should help the reader to follow the information flow from the *design-time* phase (Chapters 3-5) to the *run-time*

phase (Chapters 6-7): in the proposed application design methodology, the design knowledge gathered at design-time by means of DSE is exploited to support run-time decisions for optimal resource allocation and efficient application auto-tuning.

Part I. OpenCL Application Optimization and Characterization

DSE consists of exploring the configuration space given by all possible combinations of tunable design parameters, to identify the optimal solutions with respect to some design objectives, e.g. throughput, power consumption, or Quality of Service (QoS). The techniques proposed in this thesis mainly exploit profile-based DSE, in which an executable or simulation model of the application and target platform is used to capture the performance metrics [108]. The executable model refers to a profiling configuration where the application is executed on the real platform, while the simulation model should be used when the platform is not available, to enable application prototyping in an early design stage in order to reduce the time-to-market.

OpenCL is emerging as a standard for parallel programming of heterogeneous hardware accelerators. With respect to device specific languages, OpenCL enables application portability but does not guarantee performance portability [76], eventually requiring additional tuning of the implementation to a specific platform or to unpredictable dynamic workloads. Not only the number of cores changes from one platform to another, but also the way computational parallelism is exploited in the compilation or synthesis flow, as well as the memory access mechanisms (e.g. DMA) and the size of local memories. Thus, this thesis addresses the problem of OpenCL performance portability by adopting a parametric application design and customizing application parameters for the specific target platform.

Automated exploration and optimization of application parameters represents a novel and interesting research domain, particularly useful in the field of approximate software computing [99]. For these applications, the quality and accuracy of the output can be traded off in return for lower computational complexity and higher throughput. However, the size of a design space grows exponentially with the number of parameters. The main contributions of Part I consist of three DSE techniques for the design-time optimization phase, described below. These techniques allow to efficiently explore the design space, avoiding profiling of suboptimal or unfeasible solutions, and thus to reduce the exploration time. The output is a set of application *operating points*, which are optimal for the target platform and expose different trade-offs between performance metrics and QoS.

Chapter 3. Customization of parametric OpenCL applications

The first contribution of this thesis is a methodology to analyze the customization space of an OpenCL application in order to improve performance portability and to support dynamic adaptation. This chapter introduces a parametric design for an OpenCL stereo-matching application, then an optimization case study by targeting this application to the STMicroelectronics STHORM (P2012) many-core computing fabric [83] is presented. In this approach, DSE techniques are used to generate a set of *operating points* that represent specific configurations of the parameters allowing different trade-offs between performance and accuracy of the algorithm itself. These points give detailed knowledge about the interaction between the application parameters, the underlying architecture and the performance of the system; they could also be used by a run-time manager software layer to meet dynamic performance and QoS constraints, as shown later in Part II.
Besides the parametric OpenCL design, another contribution of this chapter consists of an exploration heuristic to reduce the overall number of simulations. This heuristic makes the DSE-based optimization approach feasible also when the simulation platform is slow. This is typically the case when a platform is in the prototyping phase and thus it can only be simulated. The experiments of this chapter use cycle-accurate simulations for the STHORM platform to analyze the customization space of the OpenCL stereo-matching application.

Chapter 4. Ensemble models for simulation of many-core platforms

The availability and the quality of software applications could determine the survival of a computing platform – as well as the success of a new platform – on the market. For this reason and also to reduce the time-to-market, during the prototyping phase it is necessary to start developing applications for the target platform on a simulation model.
This chapter presents a modeling technique to reduce the time associated with cycle-accurate simulations of parallel applications deployed on many-core embedded platforms. It uses an ensemble model based on artificial neural networks that exploits (in the training phase) multiple levels of simulation abstraction, from cycle-accurate to cycle-approximate, in order to predict the cycle-accurate results for unknown application configurations.
The experimental results show that high-level modeling can be used to significantly reduce the number of low-level model evaluations provided that a suitable artificial neural network is used to aggregate the results. The methodology for the design and optimization of such an ensemble model

is assessed on the simulation platform for STMicroelectronics STHORM (P2012) many-core computing fabric.

Chapter 5. Task mapping under heterogeneous platform constraints

While the previous DSE techniques considered a single target accelerator, in this chapter the methodology is extended to target a generic platform with multiple OpenCL accelerators, eventually heterogeneous. The OpenCL API already supports a cooperative multi-device usage model with event-based synchronization, but there is not yet any support for automatic load balancing among heterogeneous devices.
To cope with this limitation, the chapter presents a design flow for performance optimization of OpenCL applications consisting of two phases: i) a *tuning phase* that optimizes application kernels, considering device-specific constraints, and generalizes the technique presented in Chapter 3; and ii) a *task-mapping phase* that maximizes the whole application throughput on the target heterogeneous platform.
The main contribution consists of a novel analytical technique, supported by a *constraint solver* [96] for fast pruning of the design space. In the tuning phase, this technique efficiently identifies an initial set of feasible task configurations that are compliant with the platform device constraints. In the mapping phase, it improves the task-level parallelism accounting for the overhead of host-to-device and device-to-host memory transfers (overheads implied by multiple OpenCL contexts for different device vendors).
The result is a software pipeline optimized for the target heterogeneous parallel platform to maximize the average application throughput.

Part II. Application Auto-Tuning and Run-Time Management

The ever increasing number of processing elements integrated on the same many-core chip delivers computational power that can exceed the performance requirements of a single application. The number of chips (as well as the related power consumption and production cost) can thus be reduced by deploying multiple applications on the same chip – a practice which is called *resource consolidation*. This practice is already common in the HPC domain, in which *virtualization* [18] allows multiple users to acquire quotas of the same cluster to run different applications. In the embedded domain, the interest for virtualization techniques is more related to security and reliability aspects [69], in order to isolate critical functionality from best-effort applications on mixed-criticality systems. On the contrary, partitioning and allocation of platform resources to improve the overall system performance

is still based on fair allocation of the user time among the active tasks, as done by the Linux scheduler. Moreover, this type of Run-Time Management (RTM) is limited to multi-core CPUs, because other heterogeneous accelerators are usually not managed by the Operating System (OS).

Part II presents a run-time resource management framework, based on design-time optimization and characterization of the workload by means of DSE. At the same time, the design-time techniques presented in Part I provide fine-grained trade-offs between performance and QoS that will be exploited by an Application-Specific Run-Time Manager (AS-RTM) to improve the average use-case performance. In this context, the term *auto-tuning* refers to the capability of an application to trade-off application-specific metrics, such as the accuracy of result, with performance metrics.

Thus, the contribution is twofold: an auto-tuning software layer for fine-grained application adaptivity and light-weight autonomous resource management; and a more general two-level Run-Time Management (RTM) framework, to decouple resource allocation from application adaptivity.

Chapter 6. Application auto-tuning with autonomous RTRM

To satisfy the performance requirements in the presence of resource contention, the approach presented in this chapter exploits application auto-tuning, based on design-time analysis, of both application-specific *dynamic knobs* [56] and computational parallelism. Such features are implemented in a software library, linked to each application, which is used to demonstrate the main contribution: a light-weight Run-Time Resource Management (RTRM) technique to improve resource sharing for computationally intensive OpenCL applications.
The experiments are aimed at evaluating how much the interaction between RTRM and application auto-tuning can become synergistic yet orthogonal. In the proposed approach, run-time adaptation decisions are taken by each application, autonomously. This has two main advantages: i) a non-invasive application design, in terms of source code, and ii) a very low run-time overhead, since it does not require any central coordination by a supervisor nor communication between the applications.
An experimental campaign was carried out by using a video processing application – the OpenCL stereo-matching implementation presented in Section 2.3 – and stressing out resource usage. The results show that, while RTRM is necessary to provide lower variance of the application performance, the application auto-tuning layer is fundamental to trade it off with respect to the computation accuracy.

Chapter 7. Combining application adaptivity and system-wide RTRM

To better exploit the computational capabilities offered by multi-core high-end embedded systems, new parallel programming paradigms, such as OpenCL, should be combined with effective resource management. However, dealing with mixed-priority workloads and time-varying scenarios still represents an open problem.
The contribution presented in this chapter addresses such challenges by exploiting the synergy between the MOST DSE framework and the Barbeque RTRM – presented in Section 2.5 and Section 2.6, respectively – to achieve effective and flexible system-wide application adaptivity. In particular, design space exploration techniques are used to generate a set of *operating points*, which are selected at run-time by an Application-Specific Run-Time Manager (AS-RTM), the same presented in Chapter 6. These points are also clustered with respect to performance and resource requirements to identify a set of Application Working Modes (AWMs), to be used by the Barbeque RTRM for efficient resource allocation.
The technique presented in this chapter was developed in the context of the European FP7 project 2PARMA[5] and validated on an embedded many-core platform as well as on a NUMA server x86 machine.

[5] PARallel PAradigms and Run-time MAnagement techniques for Many-core Architectures, project website: http://www.2parma.eu/

CHAPTER 2

Background, Terminology and Toolchain

The goal of this chapter is to provide the reader with the necessary background before describing the proposed techniques and to introduce applications and tools used in the following of this thesis. More in details, Section 2.1 describes the research context, introducing some related works and underlining the motivations of the proposed approach.

This thesis work addresses the OpenCL programming paradigm to target heterogeneous parallel computing platforms. OpenCL is being adopted by many platform vendors, in the embedded and High Performance Computing (HPC) domains, mainly because it allows to effectively exploit both data parallelism and concurrency in an application task graph, while providing a cross-platform API. Thus, Section 2.2 gives a brief presentation of OpenCL, while the reader can refer to the specification [67] for a detailed API description.

Augmented reality and advanced image processing are just two examples of computationally intensive multimedia applications which are now required on high-end mobile devices, while exhibiting classical HPC traits. Stereo-Matching, described in Section 2.3, belongs to a class of applications that allow to dynamically tune at run-time the trade-off between performance and accuracy metrics, by means of some tunable parameters. Therefore,

the experiments consider different case studies based on a Stereo-Matching application, implemented in this thesis with OpenCL APIs.

OpenCL enables application portability but does not guarantee performance portability, eventually requiring additional tuning of the implementation to a specific platform. Therefore, the experimental setup includes different types of OpenCL devices used as software programmable accelerators, described in Section 2.4, in order to analyze the problem of application optimization for heterogeneous parallel platforms.

Finally, Section 2.5 presents the MOST framework and Section 2.6 the Barbeque Run-Time Resource Manager. The former provides a Design Space Exploration (DSE) tool, used in Part I to automate the process of customizing and optimizing an application for the target platform. The latter is used in Part II to implement a Run-Time Resource Management (RTRM) framework, for run-time optimization of resource allocation on multi- and many-core platforms.

2.1 Background

In the embedded domain, platform-based design [66] represents a dominant paradigm to design optimized architectures and to meet time-to-market constraints. In this approach, a template Multi-Processor System-on-Chip (MPSoC) platform is provided by the silicon vendors as reference design. Then, the application developer can use conventional Design Space Exploration (DSE) to tune the hardware/software parameters of the platform in order to achieve the desired trade-off between area, power consumption and performance. DSE generally consists of a multi-objective optimization problem whose domain is composed of parameters of a microprocessor-based platform (and all integrated IPs) and/or application parameters while the objective functions are, typically, the overall performance and power consumption [66]. The solution of the optimization problem is a set of non-dominated points termed *Pareto set* [30].

The emphasis on the efficiency of the DSE algorithms is due to the fact that in computer architecture research and development, simulation still represents the main tool to predict performance of alternative architectural design points. If we consider a full cycle-accurate system-level simulation, even the most simple workload requires a significant time to be analyzed, thus a comprehensive exploration of the design alternatives could exceed practical limits [92]. The overall goal is thus to minimize the number of simulations to be executed during the exploration phase. A common approach to solve this problem is to use Design of Experiments (DoEs) and

Response Surface Model (RSM) techniques [101] combined with suitable multi-objective minimization or maximization techniques [30]. An alternative approach is based on meta-heuristics (such as Simulated Annealing [110] and Genetic Algorithms [35]) that provide more accurate results but in the long term considering simulation time.

The possibility to customize the platform and to implement optimized software tasks for each available processor led to a deep research into hardware-software co-design. However, this approach is complex, because of the size of the design space, and expensive at the same time, thus it can be applied only for large production volumes. More recently, general purpose parallel computing platforms have been proposed for both the embedded and HPC domains, consisting of a host processor and one or more multi- and many-core accelerators (e.g. STMicroelectronics STHORM [83], Adapteva's Parallela platform[1]). Such platforms provide sufficient computational power to support a wide range of applications and can be easily programmed with software, e.g. by means of the OpenCL API [67], reducing the per-unit design cost for small production volumes.

The development of complex software for parallel, heterogeneous platforms – the so called *programmability wall* – has been addressed with a wide range of software engineering methodologies. Among the most prominent approaches in the area of OpenCL development we can find *template libraries* for imperative languages [33, 36, 115] and *functional meta-programming* [28, 29, 72].

Template libraries for imperative languages provide optimized kernel implementations for operations on vectors and matrices. Besides, they provide an easy interface to the low level primitives of the OpenCL toolchain, by enabling straightforward composition of kernels. This property enables the programmer to shift the focus towards the top-level view of the application, improving productivity. However, these libraries are limited to regular vector or matrix operations, thus they are most suitable for arithmetic kernels.

Functional meta-programming is a technology widely used in the field of functional languages. It allows to specify programmatically how and when a certain part of the code should be generated and executed. Code generation can thus be done not only at design-time but also later (either at *walk* or *run-time*). This opens opportunities for automatic application customization and optimization which could greatly improve the performance on heterogeneous platforms. The state of the art in this case is very advanced, as both Obsidian [29] and Barracuda [72] are able to generate CUDA code that makes automatic use of local memory on GPU devices.

[1] http://www.adapteva.com/introduction/

Domain Specific Languages (DSLs) can be used as well to provide a re-targetable dynamic compilation framework [77, 26, 37]. This approach is application-domain specific and usually limited to a class of kernels (e.g. stencil operations), thus it cannot address *kernel heterogeneity* within complex applications. The generated code outperforms hand-optimized programs, because the high-level description in a DSL enables a set of intermediate source-to-source code transformations before compilation for the target platform. However, due to the nature of DSLs, it is very difficult to agree on a common programming framework.

This thesis presents an innovative approach to optimize OpenCL application design, focused on software-programmable multi- and many-core accelerators for embedded computing platforms. It is based on the DSE techniques already studied for MPSoC design, applied here to a parametric application design, such as the OpenCL Stereo-Matching application presented in Section 2.3. The performance portability problem, in this case, is casted into a multi-objective optimization problem, which allows to consider, by means of profile-based DSE, several design objectives, e.g. power consumption, throughput and Quality of Service (QoS) [122].

The OpenCL introduction given in Section 2.2 will show that there are some parameters to customize the behavior of an OpenCL runtime for a target platform. Besides these OpenCL parameters, application-specific parameters must be also considered by the design-time phase. They can be found in a class of applications that allow to dynamically tune at run-time the trade-off between performance and Quality-of-Result (QoR) metrics, by means of *dynamic knobs* [56]. An example of such knobs consists of those parameters that change the number of loop cycles, by skipping some iterations with a given step (a technique known as *loop perforation* [106]). This generates a more approximate result, which we also refer to as *QoR loss*, in return for a faster execution.

Thus, the domain of the multi-objective optimization problem is composed of both platform-related and application-specific parameters. However, the number and the range of customizable parameters define a large design space. Such complexity makes it impossible to take fast run-time decisions without any knowledge about the interaction between parameters and metrics. On-line RSMs have been used to improve run-time management by either using piecewise linear regression [17], Artificial Neural Networks (ANNs) and Q-learning [81] or other predictive/adaptive models [31]. Design-time exploration is increasingly used as an alternative technique to provide simple guidelines for run-time management of small eco-systems[121, 79] or to support more complex and versatile *composite*

approaches [105]. Following this second technique, the design-time phase proposed by this thesis is aimed at building a *performance model* of an application on the target platform, to be used by the run-time framework presented in Part II.

Part II will also deal with multi-application scenarios deployed on multi-core platforms. These scenarios expose both *task-level parallelism*, at the application-level, and *request-level parallelism* [51], at the system-level in the form of multiple independent tasks. The OpenCL API is designed to make efficient use – at application-level – of the massive computational parallelism provided by modern accelerators. On the contrary, there is not yet support for efficient deployment of multiple OpenCL applications on the same platform. Several frameworks (e.g. OmpSs [39], SOCL [52], SnuCL [68]) have been proposed to optimize execution of an OpenCL application on a heterogeneous platform, but they are limited to a single application. The reason for this limitation is that the runtime support is usually provided as a library to be linked to the application. Conversely, run-time management of multi-application scenarios and dynamic workloads requires some form of coordination among the applications – either in a centralized [55, 19] or distributed approach [71, 119] – supported by an inter-process communication infrastructure.

2.2 OpenCL programming model

Open Computing Language (OpenCL) is an open standard for parallel programming of heterogeneous systems [67]. Its standard API enables cross-platform portability of applications between different types of accelerators: multi-core CPUs, General Purpose GPUs (GPGPUs), many-core accelerators (such as STMicroelectronics P2012 [83] or Adapteva's Epiphany) and – more recently – also FPGAs [109]. These accelerators are off-the-shelf components that can be used to speed up compute-intensive applications, while reducing the design cost with respect to application-specific platform solutions. OpenCL allows to use such accelerators as programmable parallel processors, thus enabling platform re-usability and addressing a wide spectrum of application fields, from graphics and multimedia to scientific and medical software.

OpenCL has a strong CUDA [88] heritage, but it provides a common hardware abstraction layer across different multi-core architectures [76]. It follows an off-loading programming paradigm, where the application code running on a host CPU is responsible for offloading computationally intensive kernels on the available accelerators (also called *OpenCL devices*) and

takes care of transferring data from host to device, and *viceversa*, because of different address spaces. From a programming perspective, OpenCL provides a standard host API to access different types of co-processors to ease porting of application code across different platforms.

In order to write portable code, the application designer should only express the potential parallelism and let the compile or synthesis toolchain optimize the code for a specific target architecture: thus, the way parallelism is exploited on a GPU architecture might be very different from the way it is done on a FPGA. The OpenCL host API allows to express application parallelism at two levels:

- At task-level, by means of an **iteration space** (called *NDRange* in the OpenCL specification), to express potential data-parallelism in compute-intensive application hotspots (called *kernels*).
- At application-level, in the form of **task graph**, by expressing data dependencies among tasks deployed within the same OpenCL context but executed on different *command queues*, each command queue being attached to a computing device.

These two ways of expressing parallelism are analyzed more deeply in the following sections, since both will be exploited by the parametric application design proposed in Part I. Finally, the last paragraph will introduce the *device fission* API, extensively used in Part II to control resource assignment to OpenCL applications.

2.2.1 OpenCL NDRange: intra-task parallelism

OpenCL is a language and runtime specification born to address intra-task parallelism, which mainly exploits data parallelism. On the one hand, it allows to create highly optimized kernels because commercial OpenCL compilers can exploit vector units, Fused Multiply-Add (FMA) blocks, floating-point units and other DSP blocks available on an OpenCL device. On the other hand, to express intra-task parallelism, one has to specify the iteration space with a composition of two grids: a *global* and a *local* one. The global grid scales linearly with the problem size and provides a baseline for partitioning work across the processing units of the device. It allows to express fine-grained data parallelism, which results in the parallel execution of the same program – the OpenCL kernel – for each point in the iteration space – the work-items. The local grid is used to assign iterations to each kernel work-group, which consists of work-items sharing data through a *local memory*.

OpenCL uses a relaxed consistency memory model [47], i.e. work-items could be executed in any order and the state of memory visible to a work-item is not guaranteed to be consistent across the collection of work-items at all times. However, OpenCL provides some synchronization mechanisms at the work-group level in the form of barriers, to enable consistency of shared buffers in local memory.

Sizing of the local grid is critical since:

1. too large local grids might cause register spill on OpenCL devices such as GPUs or CPUs [46],
2. local memory can be too small to accommodate all the work-items,
3. too small grids can lead to inefficient use of local memory, increasing the number of accesses to the slower global memory.

It is therefore extremely important to consider grid sizing in application design [98, 111]. Both global and local grids are indeed exploited differently by each platform compiler and/or runtime, to make efficient use of the underlying architecture. For example, on modern CPUs the OpenCL compiler takes advantage of advanced vector instruction sets [60], while on Altera FPGAs the synthesis tool transforms the iteration space into a deep hardware pipeline [109]. Thus, in the design methodology presented in Chapter 3, grid sizing is exposed as a set of application parameters in order to enable automated optimization by means of DSE.

As already mentioned, the size of the local grid determines also the amount of local memory required by a work-group for the shared buffers. Thus, the amount of local memory available on a device represents a platform constraint, which can be used to filter out from the design space the unfeasible solutions (Chapters 3, 5). At the same time, the work-group size impacts on the memory bandwidth for reading and writing data to the global memory. On platforms that use Direct Memory Access (DMA) to transfer data to the local memory, software-pipelined kernel implementations could benefit from overlapping data transfer with processing [77]. Thus, the application design proposed in Chapter 3, which targets P2012 (see Section 2.4) and uses DMA at the work-group level, will consider grid sizing in order to balance the pipeline stages.

2.2.2 OpenCL task graph: inter-task parallelism

In the previous section, we introduced the concept of OpenCL *NDRange*, which allows to express and exploit data parallelism on multi-core accelerators. OpenCL applications have to explicitly offload a data-parallel task

(kernel *NDRange*) on an accelerator device, by *enqueuing* it on the *command queue* attached to that device; then, the OpenCL runtime will execute the task as soon as the device becomes available.

However, when it comes to multiple devices OpenCL falls short. While it allows device partitioning for running multiple tasks on the same device, it does not provide an easy way to orchestrate task execution on multiple devices. This is due in part to the inability of using a single command queue for all the devices. Still, it is possible to exploit multi-device execution using separate command queues for each device and event-based synchronization, but this requires that:

- the application developer should write *ad hoc* host code to orchestrate the *command queues*
- devices should belong to the same vendor platform.

However, if two devices do not belong to the same vendor platform, OpenCL does not allow to create a context containing both devices. If two devices belong to two separate contexts, then it is not possible [52]:

- to synchronize commands executed on one device with commands executed on the other;
- to share buffers between the devices;
- to copy data from a buffer on one device to a buffer on the other device.

Therefore, the developer is forced to instantiate separate OpenCL contexts for different devices, taking care of copying data across contexts and using very inefficient synchronization methods, such as waiting until a command queue gets empty. Thus, designing and hand-coding an OpenCL application that maintains load balance between different heterogeneous devices, while efficiently managing memory transfers, is a complex task. Moreover, it is difficult to write portable code, since the layout in terms of available accelerators could be very different from one platform to another.

To address these limitations of the OpenCL API and enable scheduling on heterogeneous platforms, a number of techniques and frameworks have been proposed in literature. The authors in [49] propose a static approach based on compiler analysis of the kernel code to predict the optimal partitioning on a heterogeneous platform for an OpenCL data-parallel task. However, this approach introduces memory transfer overhead for data dependencies in complex task graphs. On the contrary, SOCL [52] is a unified OpenCL platform which uses dynamic and adaptive load balancing of the application task graph. Thus, OpenCL is extended in order to support command

queues attached not only to one particular device, but to a group of devices – eventually from different vendors. More oriented to HPC, the SnuCL framework [68] allows an application to exploit heterogeneous CPU/GPU clusters, where the nodes are on a network, by exposing OpenCL devices in a compute node as if they were in the host node.

2.2.3 OpenCL device fission

In the OpenCL platform model, each accelerator contains one or more compute units. For example, for a multi-core CPU a compute unit is a *worker thread* executing on a core, while for a GPU a compute unit is a stream processor. By default, the OpenCL runtime distributes the work-groups of one kernel NDRange among all compute units of the device attached to the command queue. Thus, an NDRange kernel executed on a multi-core CPU, by default, uses all the available cores.

However, since the number of cores is rapidly increasing, it is beneficial to let the OpenCL programmer take more control over which resources should be used by the runtime, rather than treating a device as a single homogeneous computing resource[2]. For this reason, the OpenCL device fission API was first introduced as an OpenCL 1.1 extension, then included in the OpenCL 1.2 specification [67]. Fundamentally, device fission allows sub-dividing a device into one or more sub-devices, thus enabling more advanced inter-task parallelism across the command queues.

There are several partitioning options to apply the device fission [67]. Once created, a sub-device can be used as a generic OpenCL device, with standard APIs. These APIs will be used in Part II to assign resources of a homogeneous multi-core platform to multiple applications, within a framework for RTRM.

2.3 Target application: OpenCL Stereo-Matching

The stereo-matching algorithm described in [125] has been implemented in an OpenCL application, to be used in the experimental part of this thesis. Stereo-Matching is relevant for multimedia applications and, at the same time, provides a parameterizable and adaptable design. The reason for using OpenCL, as programming paradigm, is to enable cross-platform portability and evaluate the proposed techniques for application optimization on heterogeneous parallel platforms.

In the computer vision domain, one application of interest is 3D scene

[2] https://software.intel.com/en-us/articles/opencl-device-fission-for-cpu-performance

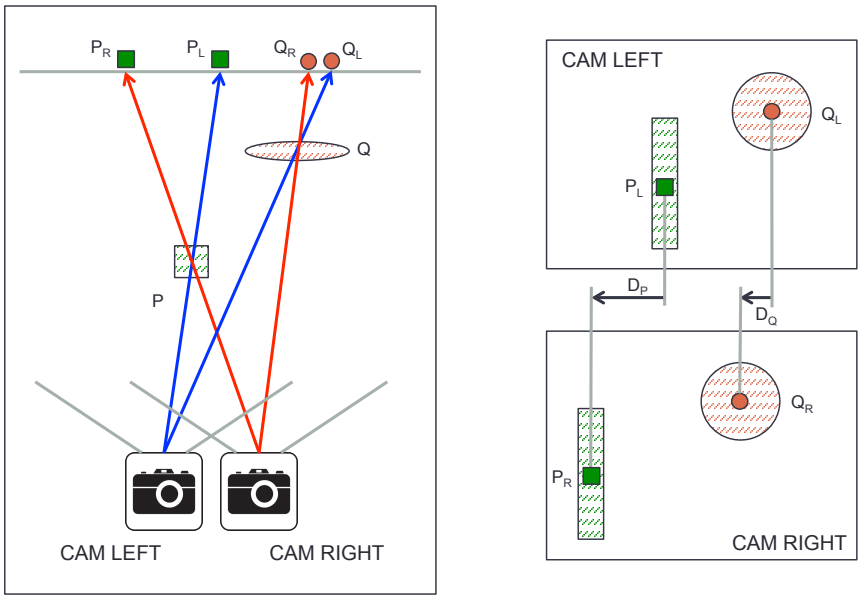


Figure 2.1: *Example of input stereo images (on the right), obtained shooting a scene which contains two objects: a green parallelepiped and an orange spheroid.* D_P *and* D_Q *represent the* binocular disparity *of pixels P and Q, where* $D_P > D_Q$ *since P belongs to the object closest to the cameras.*

reconstruction, which builds on top of stereo-matching algorithms to compute the depth map for a scene captured with stereo cameras. One example application is the *Lens Blur* filter in the Google Camera App[3], which allows to take out-of-focus background from a photo by blurring pixels by different amounts depending on the pixel depth. In this way, a software application provides on smart-phones and tablets the same optical effect that traditionally required professional cameras with advanced optics.

Given a stereo-image pair as input, the stereo-matching algorithm assumes that each pixel in the right image ('anchor') has a corresponding pixel in the left image, but eventually in a different position. Images are assumed to have been captured by cameras with the same vertical position, thus the search can be limited to the same row. The difference between the horizontal position in the left image and the position in the right image is called *binocular disparity* and it is inversely proportional to the distance of the point with respect to the cameras. Since it represents a distance between two points in a bitmap image, the disparity is measured in pixels.

In Figure 2.1, a 3D scene (on the left) containing two objects – a green

[3] http://googleresearch.blogspot.it/2014/04/lens-blur-in-new-google-camera-app.html

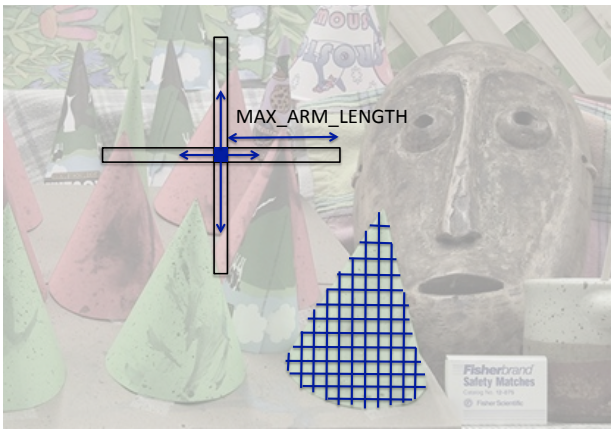


Figure 2.2: *Construction of shape-adaptive local support regions with cross-based method.*

parallelepiped and an orange spheroid – is captured by a pair of stereo cameras. The blue and red lines show the projection of pixels P and Q – belonging to the surfaces of the parallelepiped and the spheroid, respectively – on the focal plane. The projections are different for the left and right cameras, because of the binocular disparity proportional to the distance between the cameras – which is fixed and known. However, the disparity also depends on the distance of the objects from the cameras, thus the disparity of pixel P, belonging to the parallelepiped, is higher than the disparity of Q, which belongs to the spheroid and is farther than P.

The peculiarity of the selected algorithm [125] consists of adaptive-shape local support windows, based on color similarity, for each pixel: pixels with similar color are, indeed, likely to belong to the same object surface. Support windows are represented by means of a *cross*-like structure for each pixel, whose arms extend to cover any irregular shape in the image (see Figure 2.2). This technique allows aggregating matching costs in adaptive-shape support regions, thus enhancing the accuracy of the disparity result.

To compute the disparity for each pixel, the algorithm proceeds by formulating *disparity hypotheses* and combining, for each hypothesis, the support regions of the 'anchor' pixel and that of the corresponding pixel in the left image. Then, it applies an orthogonal integral image technique for fast cost aggregation of the raw matching costs on the combined window. The final result, for each pixel, is the disparity which minimizes the matching cost over the combined window, between the 'anchor' pixel in the right image and its projection in the left image. This cost is normalized over the size of the combined support window.

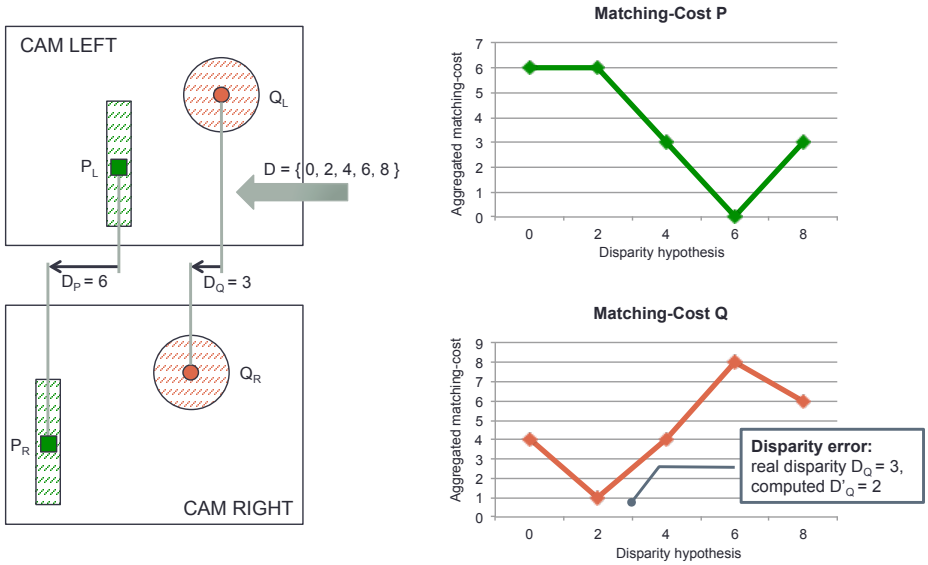


Figure 2.3: *Matching cost aggregation, considering disparity hypotheses 0, 2, 4, 6, 8.*

In the example of Figure 2.3, the exact disparities are $D_P = 6$ and $D_Q = 3$. Disparity hypotheses are considered from 0 to 8, with a fixed step equal to 2. For each hypothesis, the algorithm considers the support regions of pixels P' and Q', which represent the projection of the anchor pixels for the selected disparity. By means of image orthogonal integral, it computes the aggregated matching cost over the combined support regions, as shown in the two plots. For pixel P, the disparity hypothesis which minimizes the matching cost is 6, while for Q it is 2. Thus, by using non contiguous hypotheses, the result for pixel Q is not exact: the exact disparity is 3, while the approximate one computed by the algorithm is 2.

The difference between approximate and exact disparity is the *disparity error*, an application-specific metric already proposed in [125] to evaluate the trade-off between performance and accuracy of the computation. In particular, the QoS for the stereo-matching application, used in the experiments of this thesis, is quantified as the inverse of the disparity error. By increasing the step between consecutive disparity hypotheses, the application QoS is likely to decrease but the computation becomes faster, because a smaller number of hypotheses is evaluated by the algorithm.

The disparity information is typically stored as a gray-level bitmap file, where higher-intensity pixels represent higher disparity values (see Figure 2.4). The three disparity maps on the right of Figure 2.4 show the result

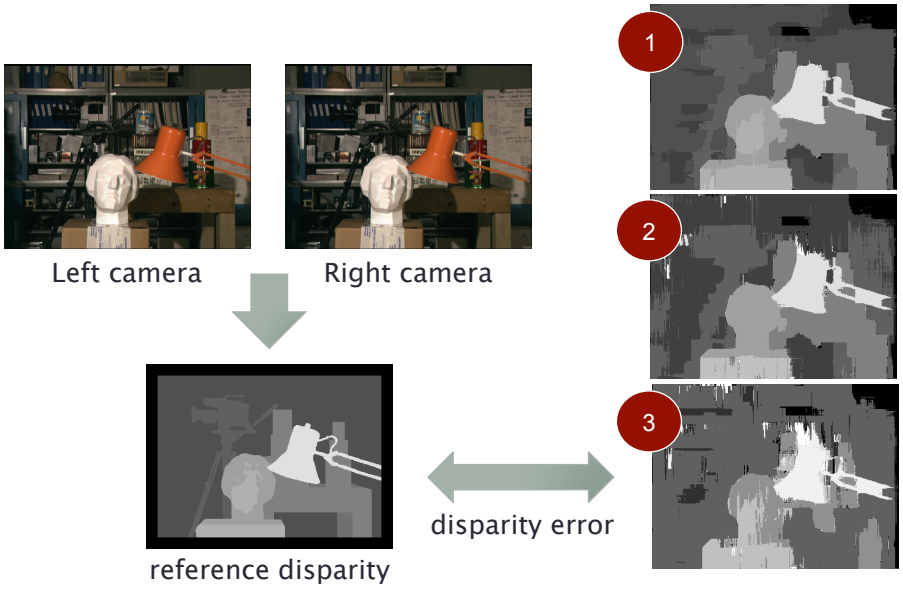


Figure 2.4: *Measurement of the stereo-matching QoS, by comparing the output for a given parameter configuration with respect to a reference disparity map.*

on the Tzukuba dataset by varying the hypothesis step. As the step increases, the QoS decreases. However, this is not the only parameter, there are other parameters that allow to control the accuracy – and the performance – of the stereo-matching algorithm. These parameters will be referred to as *application parameters* in the rest of this thesis and they are listed below.

color_threshold: defines a threshold for color similarity with respect to the anchor pixel when building a shape-adaptive support region.

max_arm_length: the support region of each pixel is encoded in a cross structure and all pixels covered by the cross arms (left, right, up and down) are similar in color to the anchor pixel: this parameter represents the maximum arm length and thus limits the size of support windows (see Figure 2.2).

max_hypo_value and **hypo_step**: these parameters define the set of integer disparities that the algorithm should test for each pair of pixels. Each disparity value is a 'hypothesis' that is actually evaluated by the algorithm. The disparity hypotheses (in pixels) will range from 0 to *max_hypo_value*, with a given *hypo_step* step. The latter parameter acts as the 'resolution' of the algorithm, by setting the step between consecutive disparity hypotheses.

matchcost_limit: controls the truncation limit of matching cost.

2.4 Target platforms

As a cross-platform open standard, OpenCL is used today for programming heterogeneous parallel platforms. Although its API provides functional portability over a wide range of computing devices, OpenCL applications then require additional design-time and run-time tuning for achieving best performance on each target platform. To address this problem and to validate the proposed techniques, different types of accelerators have been used in the experimental setup of this thesis work.

One platform is the STMicroelectronics P2012 (STHORM) [83], a research many-core computing fabric developed in the context of the 2PARMA FP7 European project [107]. This platform was designed to improve the performance/energy efficiency in order to enable parallel processing for embedded applications, but it exposes a degree of complexity in the on-chip and off-chip memory hierarchy. Thus, it requires platform-specific customization of the OpenCL applications. The other platforms are desktop workstations with CPU/GPU heterogeneity, which were selected in order to generalize the techniques proposed in this thesis. Since this information represents a common background for the following chapters, the platforms are introduced in this section.

2.4.1 STMicroelectronics Platform 2012 (STHORM)

The target platform for the applications used in Chapter 3 and Chapter 4 is STMicroelectronics STHORM, a low-power many-core computing fabric also known as Platform 2012 [21, 83, 52]. STHORM consists of a Globally Asynchronous Locally Synchronous (GALS) fabric of clusters (4 in our configuration), connected through an asynchronous global Network-on-Chip (GANOC). The STHORM cluster (see Figure 2.5) is composed of a multi-core computing engine, called ENCore, and a cluster controller. The ENCore cluster hosts 16 Processing Elements (PEs); the base processing element of the ENCore engine is a STxP70-V4 processor, a dual-issue customizable 32-bit RISC core of STMicroelectronics with a 32-bit floating-point unit.

The processing elements do not have private data caches to avoid maintaining memory coherence across the cores. Nevertheless, they share a L1, 256 KB multi-banked Tightly Coupled Data Memory (TCDM): this memory is mapped as *local memory* by the STHORM's OpenCL toolchain. A DMA engine allows for efficient asynchronous memory transfers between global memory (*off-chip*) and local memory (*on-chip*): this mechanism – exploited by the OpenCL asynchronous copies – allows hiding the latency of accesses to the external memory, which can be several hundreds of clock cycles.

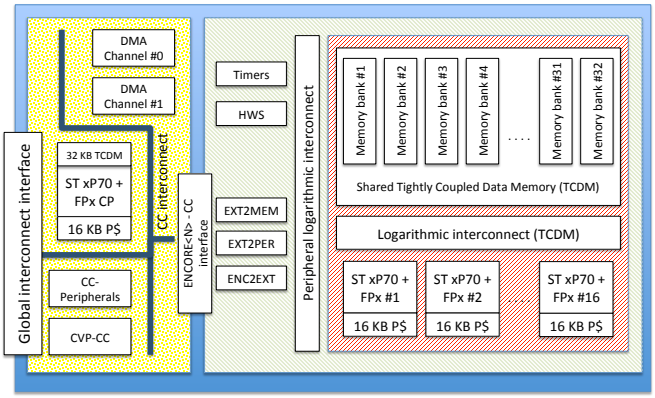


Figure 2.5: *STMicroelectronics P2012 (STHORM) cluster architecture overview.*

Two aspects must be taken into account when designing an OpenCL application targeted to STHORM:

- The latency of accesses to external memory represents a bottleneck for OpenCL applications on STHORM. Thus, some design effort needs to be spent in order to reduce the number of read/write operations and to exploit asynchronous transfers. In particular, *software pipelining* should be used to write the core application kernel, in order to allow for asynchronously fetching next data with DMA while elaborating current data.

- One workgroup is executed on one cluster and each one of its work-items is mapped to a processing element, to exploit the computational power at the cluster level. Work-items have access to the 256 KB local memory and to the synchronization module available on ENCore that provides a hardware-supported accelerated synchronization. Parallelism between workgroups can only be exploited on a multi-cluster configuration and it is anyway limited by the number of clusters.

STHORM SDK includes a simulation platform – *GePop* – that can be configured in terms of simulation speed and of accuracy with regard to the hardware architecture. The two configurations used in this thesis are *posix-posix* and *posix-xp70*:

- In *posix-posix*, the host processor and each STHORM processing element are modeled as POSIX threads and the STHORM memory hierarchy is also modeled. Application and OpenCL runtime code run natively on the workstation for both host and STHORM side.

- In *posix-xp70*, the host code (application and runtime) is executed natively on the workstation in a POSIX thread. The STHORM device is modeled as an architecture accurate platform based on xp70 Instruction Set Simulator (ISS), on which the OpenCL runtime and the application kernel execution are simulated.

2.4.2 CPU/GPU platforms

Two additional platforms were used in this work, as follows.

PLT1: Workstation with Intel Xeon Quad-Core CPU E5-1607 at 3.0 GHz and 8 GB RAM, running a Linux distribution based on kernel 3.5. OpenCL 1.2 runtime for the CPU, provided by Intel OpenCL SDK 2013. Discrete GPU NVIDIA Quadro NVS 300, with OpenCL 1.1 runtime provided by CUDA SDK version 5.5.

PLT2: Non-Uniform Memory Access (NUMA) machine with four nodes, each a Quad-Core AMD Opteron Processor 8378 at 2.4 GHz, with 8 GB of RAM per node, running a Linux distribution based on kernel 3.9. OpenCL 1.2 runtime provided by AMD Accelerated Parallel Processing (APP) SDK v2.8.1. No discrete GPU available.

As explained in Section 2.2, multi-core CPUs already support the OpenCL *device fission* API, thus the Intel and the AMD OpenCL platforms were used in Part II to validate the RTRM techniques.

2.5 MOST: Multi-Objective System Tuner

For the design-time exploration and optimization phases, in this thesis we have used the Multi-Objective System Tuner (MOST). MOST is a DSE tool, built on top of the open-source project Multicube Explorer developed at PoliMI [124]. It provides a re-targetable framework to drive design multi-objective optimization towards the Pareto-set in the objective space [30], thus skipping sub-optimal solutions. Since the design-time optimization techniques described in Part I have been implemented and validated using this tool, an overview of MOST is given in this section.

In order to evaluate a design configuration, MOST requires a simulation model. The key feature supporting re-targetability is that MOST does not include any model of the target problem but provides a standard XML interface to enable easy integration of it. This model enables profiling of the metrics to be optimized for each design configuration, but it is used as a *black box*: MOST internally represents the design space as a Cartesian space and uses the simulation model to find a projection of the design points into the objective space.

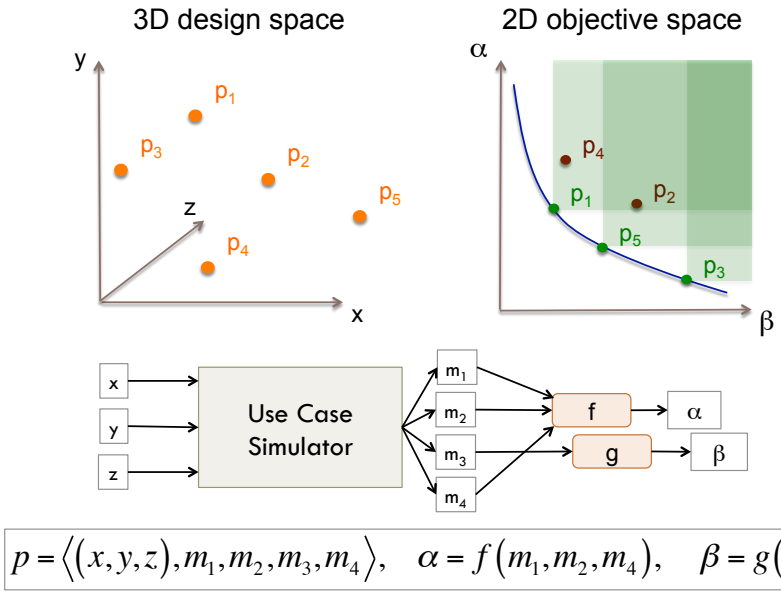


Figure 2.6: *Projection of points from the design space (left) to the objective space (right). The design space represents the Cartesian space given by the design parameters (x, y and z in this example). The same points appear in the objective space, which shows the objective functions on the axes.*

The internal representation used by MOST can be better understood by looking at the example illustrated in Figure 2.6. It shows an optimization problem with three decision parameters (x, y and z) and two objectives, the functions α and β to be minimized. The plot on the left is the MOST internal representation of the design space, where the coordinates of each point are given by the configuration of parameters. The same points appear in the plot on the right, which represents the projection of design points in the objective space. The position of the points in the second plot depends on the values of the objective functions α and β. Each design point is represented with a tuple, containing the vector of parameter values and associated metrics. The metrics are not computed by MOST, but can be retrieved either by the simulation model provided by the user or by a proper RSM trained on a subset of the design space. The multi-objective optimization problem consists of a set of minimization functions, computed on the design metrics: in this example, α is a function of metrics m_1, m_2 and m_4, while β depends only on metric m_3.

To configure this optimization problem, the designer has to provide two input files, the two dashed-line modules in Figure 2.7:

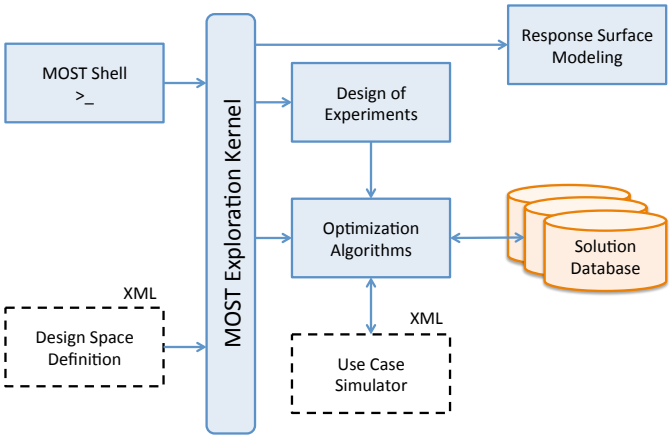


Figure 2.7: *Overview of the MOST architecture. The filled modules are internal components, while the dashed-line modules are XML files that need to be input for a particular optimization problem.*

- The XML design space description, which defines the parameters and metrics of the design and, for each parameter, the range of values;
- An XML interface to the use case simulator, which allows MOST to invoke the simulator and to retrieve the metrics for a given design point.

Then, by means of scripting (the *MOST shell* in Figure 2.7) the designer can define the objective functions and start the exploration in batch mode.

In general, MOST is a simulation- or profile-based DSE framework. Although it was initially conceived for optimization and customization of MPSoC platforms, MOST has been recently also applied to compiler optimization [15] and – in this thesis – to application customization. In the context of this thesis, the simulation model consists of an OpenCL application deployed on a parallel platform, but the platform itself can be either the real hardware one or just a simulation model. In the first case the performance metrics, such as execution speed, are more accurate, while in the second case it is possible to exploit power and thermal models to analyze more architectural design issues. However, in both cases MOST allows 1) to profile the behavior of an application running on the target platform and 2) to identify a set of optimal configurations.

MOST has a modular structure regarding its internal components: DoEs, RSMs and optimization algorithms. DoEs are sampling techniques used to select an initial set of experiments, out of the design space, to start analyzing the system behavior. RSMs are techniques used to describe the analytical relation between design parameters and one or more metrics, to build a

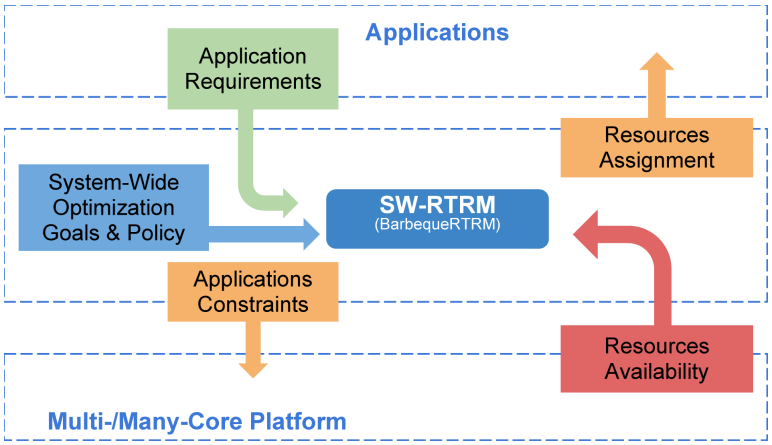


Figure 2.8: *Overall view of BarbequeRTRM interactions with applications and the underlying platform to support system-wide run-time resource management.*

meta-model of the system behavior. The optimization algorithms are the heuristics used to dynamically select the next configurations to evaluate in the DSE loop, trying to reduce the overall number of simulations while achieving close-to-optimal Pareto solutions. By combining DoEs, RSMs and optimization algorithms it is possible to define heuristics tailored to a class of problems or specific platform architectures [92].

2.6 Barbeque Run-Time Resource Manager

The System-Wide Run-Time Resource Manager (SW-RTRM) represents the core component of the resource management approach presented in Chapter 7. The framework implementing this module is *BarbequeRTRM*[4], developed at PoliMI [19].

An overview is provided in Figure 2.8. From a high abstraction level, the SW-RTRM is in charge of collecting two types of information:

1. Variable resource requirements from the running applications: this information is generated whenever a new application is started, so resources must be assigned, or an application completes its execution and releases the resources; moreover, it includes dynamic resource requests during application run-time.

2. Changes in system resource availability: this information comes from platform monitoring of run-time thermal and reliability issues.

[4]Project website at http://bosp.dei.polimi.it

These two kinds of information can trigger an *optimization step*, where a system-wide optimization policy is executed to identify a new *resource partitioning* among demanding applications, on a event-driven basis.

Static application requirements are defined by a predefined set of possible configurations, i.e. the Application Working Modes (AWMs) identified at design-time through DSE, each one corresponding to a certain amount of platform resources. Besides, *dynamic* application requirements can be adjusted at run-time by calling the *setGoalGap* API, in case an application cannot meet its performance goal with the assigned AWM.

A detailed description of the optimization policy has been provided in [19]. Briefly, the resource management problem can be formulated as a Multi-choice Multi-dimension Multiple Knapsack Problem (MMMKP) [63], which is known to be NP-hard. The Barbeque scheduler implements an optimization policy as a resource manager module, based on state-of-the-art heuristics [121] that enable near-optimal solutions and are fast enough for effective run-time exploitation. More in detail, the AWM selection is performed by considering system-wide optimization objectives, spanning from application QoS to priority, from reconfiguration overhead to fairness, as well as power consumption minimization and stability enforcement.

Once a new resource assignment has been identified, the SW-RTRM leverages on the control mechanisms implemented by the Platform Integration Layer (PIL) module to setup resource constraints for the scheduled applications. For instance, for multi-core Linux-based systems, the PIL relies on the Control Groups framework [20]. This allows to setup a set of isolated execution contexts, one for each scheduled application, matching the optimal resource requirements identified at design time.

Finally, resource assignment (AWM selection) is notified to each application by exploiting the Abstract Execution Model (AEM) API provided by the application runtime library (*RTLib*). This library handles communication with the SW-RTRM and implements the application control-loop [19].

2.7 Conclusions

This chapter introduced a programming paradigm (OpenCL), a parallel application (Stereo-Matching), different heterogeneous platforms (STHORM, CPU/GPU workstations) and some external tools (MOST, *Barbeque*) that will be used in the experimental parts. The experiments will exploit these tools to validate the novel techniques of this thesis for customization and optimization of OpenCL applications targeted to heterogeneous parallel platforms, in Part I, and for efficient run-time management, in Part II.

Part I

OpenCL Application Customization and Optimization

Overview

The first part of this dissertation is focused on design-time customization and optimization of an OpenCL application targeting a heterogeneous parallel platform. Chapter 3 presents more in detail the parametric application design and Design Space Exploration (DSE) optimization for the STHORM (P2012) platform, which was introduced in Section 2.4. The contribution of Chapter 4 is an ensemble model based on Artificial Neural Networks (ANNs) that exploits (in the training phase) multiple levels of simulation abstraction, from cycle-accurate to cycle-approximate, to predict the cycle-accurate results for unknown application configurations. Finally, the work presented in Chapter 5 completes the design-time phase with a novel analytical technique, supported by a *constraint solver*, for fast pruning of the design space and efficient task mapping on heterogeneous parallel platforms.

CHAPTER 3

Automated Optimization of Parametric OpenCL Applications

In this chapter, a methodology is proposed to analyze the customization space of an OpenCL application in order to improve performance portability and to support dynamic adaptation. This methodology was published in [7] and represents the first outcome of this thesis work as well as a relevant contribution to the 2PARMA FP7 European project.

The methodology exploits Design Space Exploration techniques to generate a set of Operating Points (OPs) that represent specific configurations of the parameters allowing different trade-offs between performance and accuracy of the algorithm itself. These points give detailed knowledge about the interaction between the application parameters, the underlying architecture and the performance of the system; they could also be used by a run-time manager software layer to meet dynamic performance and Quality of Service (QoS) requirements, as demonstrated later in Chapter 6.

The problem is that the exploration time for a full search is proportional to the size of the design space, which grows exponentially with the number of parameters. This problem is exacerbated by the long simulation time required by cycle-accurate simulation platforms, especially for highly parallel

architectures such as STHORM, targeted in this chapter. The *posix-xp70* simulation model of the STHORM platform, described in Section 2.4, enables accurate profiling of OpenCL kernel execution time; however, the simulation is relatively slow, since it takes around 30 minutes for simulating one application configuration on one input pair of stereo images.

To address this problem and to support application customization, the exploration methodology presented in Section 3.3 allows for a reduction of the overall number of simulations by exploiting relations between parameters as they are discovered and unfolded during the exploration. Finally, this chapter also presents a technique exploiting a fast high-level simulation model to further reduce the exploration time, enabling in the experiments (Section 3.4) an overall speed-up of 16x with respect to an exhaustive exploration of the entire design space.

This type of approach, where the application parameters are customized for a target platform or a specific use case, is typically referred to in literature as *auto-tuning*. Section 3.1 presents several auto-tuning techniques to optimize performance of kernels for different operations, either based on static analysis or on real profiling. When dealing with multiple optimization objectives, the solution consists of a Pareto-set and the application configuration could be chosen offline by the designer among the Pareto-optimal solutions. However, a more interesting approach is where the configuration is dynamically selected at run-time, in order to adapt the application behavior. This is also called application *auto-tuning*, but it is based on dynamic requirements and is typically used to deal with varying execution contexts.

In order to distinguish between them, in this thesis the term *auto-tuning* is reserved to the run-time phase (Part II), while offline auto-tuning is referred to as application customization and represents the main topic of Part I. Nevertheless, in the next section we keep the term auto-tuning to refer to customization techniques presented by previous works, in order not to alter their original nomenclature.

3.1 Related Work

Application auto-tuning[1] on conventional CPUs typically takes place at compile-time and has been successfully applied in several works, e.g. ATLAS [118], FFTW [45] and SPIRAL [93]. Compile-time optimization considers common CPU architectural properties and therefore focuses primarily on loop unrolling, register spills and data reuse in L1/L2 caches,

[1] Here *auto-tuning* means offline optimization and customization of application parameters for a target platform, as explained in the introduction.

while CPU performance is not much sensitive to memory bandwidth issues.

In the OpenCL programming paradigm, as well as CUDA on GPGPUs, it is difficult to take into account the architectural peculiarities of the target platform. The main reason is that the OpenCL API provides an abstraction layer, meant to ease application porting across different architectures, ranging from multi-core CPUs to FPGAs. Besides, some metrics, in particular those related to memory bandwidth, are difficult to analyze without the profiling support. Thus, most works that target parallel processing by means of CUDA or OpenCL also exploit run-time auto-tuning, but they are limited to scientific kernels for dense and sparse linear algebra.

Optimization of dense matrix multiplication. This kernel is probably the most widely used by OpenCL tutorials (e.g. [59]), because it is possible to apply several types of optimizations: sizing of the local and global *NDRange*, tiling to exploit shared buffers in local memory, coalescing of memory accesses. From a computational point of view, the peculiarity of this kernel is that each element of the result matrix is computed with a sequence of multiply-and-add operations on one row of the first input matrix and one column of the second input matrix. On the one hand, some OpenCL platforms support a fast implementation of the Fused Multiply-Add (FMA) OpenCL built-in function. On the other hand, it might be not possible to buffer an entire row or column of the input matrices in local memory, depending on the matrix size. However, dense matrix multiplication exposes a lot of data reuse across work-items, thus this kernel can benefit from a tiled implementation. Choosing by hand the optimal values for all the parameters is not trivial, because it requires a good knowledge of the underlying platform architecture [116]. Moreover, whenever the OpenCL application is deployed on a different platform, both host code and OpenCL kernels need to be customized again. Thus, several auto-tuning frameworks have been proposed, e.g. in [82].

Optimization of 1D, 2D and 3D stencils. In a stencil operation, each point in a multidimensional grid is updated with weighted contributions from a subset of its neighbors. Thus, the possibility for data reuse is limited to a search span around each work-item. This results in smaller shared buffers than for dense matrix multiplication and a relative simple kernel implementation, since tiling is not required in most cases. Nevertheless, optimization techniques for stencil kernels have been investigated in several works, e.g. [34, 62, 112], including NUMA-aware allocation, array padding, multi-level blocking, loop unrolling and reordering, as well as prefetching for cache-based architectures and DMA for local-store based architectures.

Optimization of Fast Fourier Transform (FFT). This kernel is another application often used to benchmark OpenCL platforms, because of its high memory bandwidth. An auto-tuning technique for the FFT kernel is presented in [38], while a more complex 3D-FFT implementation is proposed in [87].

Run-time auto-tuning is based on profiling, during the application setup, of the execution time for different kernel configurations. These configurations are custom kernel implementations, obtained from a generic code template by customizing some parameters during the code generation. The parameters, in turn, define a design space that is typically too large to be explored extensively. Thus, the works cited above exploit application-specific heuristics to reduce the number of tests from a full combinatorial design space. However, these heuristics are eventually also architecture-specific (e.g. only GPU [87] or Intel CPUs [58]), thus often being limited to a single platform vendor and not enabling cross-platform portability.

Overall, three are the main limitations of the previous works:

- They all provide application auto-tuning for specific kernels, in most cases just for dense and sparse linear algebra.
- An application typically consists of more kernels, of different type, in the form of a task graph; thus, combining different auto-tuning techniques, embedded in the single kernels, might be not easy.
- The optimization techniques consider a single objective, namely the execution time, while a more generic approach should take into account also power consumption, resource utilization, QoS, etc. in a multi-objective optimization problem.

The auto-tuning approach proposed in this chapter differentiates from the previous works because it decouples the application parametric design from the exploration strategy. This separation of concerns allows to exploit an external Design Space Exploration (DSE) framework and to define advanced, eventually application-specific, exploration strategies. Thus, the DSE support to application customization is the strong contribution of this chapter.

3.2 The OpenCL customizable Stereo-Matching application

The stereo-matching application developed in this thesis has been implemented primarily targeting STMicroelectronics STHORM (P2012) [83], thus some design choices address specific features of a multi-cluster parallel architecture. For example, the usage of asynchronous memory transfers

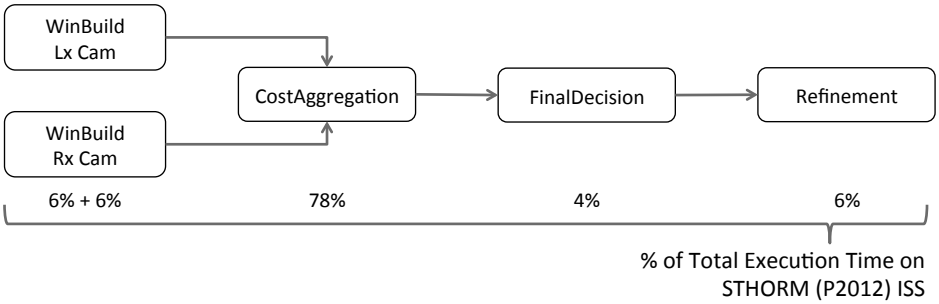


Figure 3.1: *Execution flow of the OpenCL kernels with the corresponding percentages of execution time, profiled on STHORM Instruction Set Simulator simulator.*

by means of the `async_work_group_copy` OpenCL API and the mapping of workgroups to the platform computing clusters enables optimal exploitation of STHORM. The kernel grids have been sized so to match the number of cores available on one *ENCore* cluster (see Section 2.4), but the parametric design still leaves the possibility, by means of a *resource parameter*, to change the number of workgroups and the size of a workgroup to target a different platform.

3.2.1 Structure of the OpenCL kernels

The structure of the application is shown in Figure 3.1. For each pair of stereo images, the following five OpenCL kernels are executed.

WinBuild (left and right frames): two instances of the same OpenCL kernel (one for the left and one for the right frame) can be executed in parallel to build the local support regions for each pixel in the image.

CostAggregation: this kernel evaluates all the disparity hypotheses by computing the matching-cost associated with support regions of pixels on the same line in the two reference images. A Winner-Takes-All (WTA) decision is then taken for selection of the 'cheapest' disparity hypothesis in terms of matching-cost.

FinalDecision: this kernel simply considers the results of all the workgroups involved in the previous step and decides the global disparity result. It can be seen as a reduction step.

Refinement: it performs a regularization (*smoothing*) of disparity results over the support regions.

The profiling percentages reported in Figure 3.1 show that *CostAggregation* is the most computationally intensive kernel. As proposed in [125], for

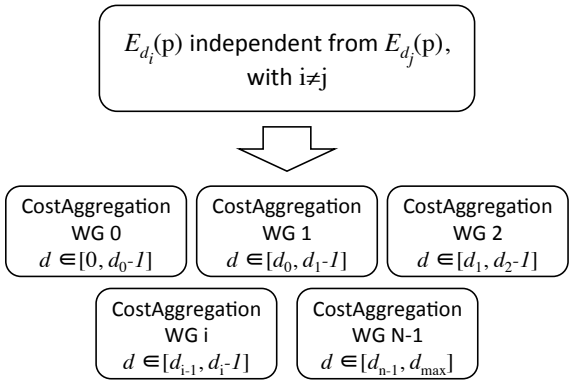


Figure 3.2: *Parallelization of matching-cost aggregation kernel over several workgroups.*

each disparity hypothesis d this kernel performs two orthogonal integration steps over the raw matching-costs on the combined support region. As expressed by Equation 3.1, the final result is the disparity hypothesis d_p^0 that minimizes the aggregation cost $E_d(p)$:

$$d_p^0 = \arg \min_d E_d(p), d \in [0, d_{max}] \tag{3.1}$$

Since different disparity hypotheses can be considered independently, it is possible to distribute the workload over several workgroups, each workgroup working on a different range of disparities (see Figure 3.2). The WTA decision is first taken at the workgroup level, on the local range of disparity hypotheses; then, *FinalDecision* kernel selects the final disparity that minimizes the aggregation cost over all the workgroups. This solution has been designed for multi-cluster architectures with a shared-memory model such as STHORM.

3.2.2 Resource parameters

In order to explore different platform configurations, the following *resource parameters* have been introduced.

Number of workgroups (*nb_workgroups*) for matching-cost aggregation. In our implementation, data-parallelism is mainly exploited by work-items within a workgroup; however different workgroups can work in parallel if more clusters are available. This parameter allows setting the number of workgroups for *CostAggregation* kernel, enabling concurrent execution on several clusters. Each workgroup will process a different range of disparity hypotheses (see Figure 3.2). The maximum allowed number

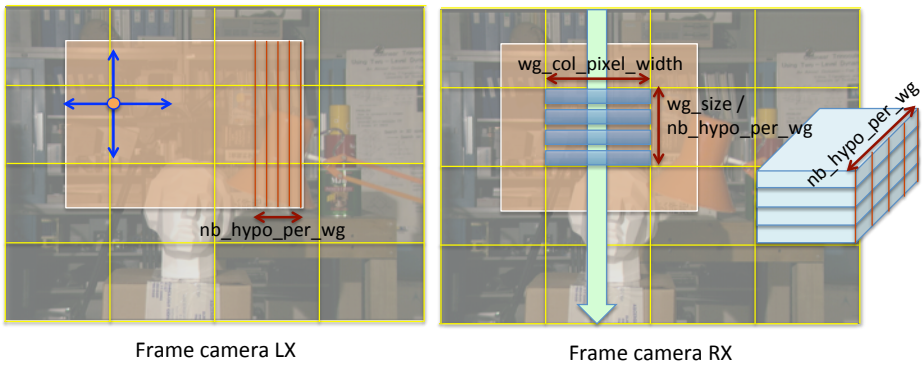


Figure 3.3: *Matching-cost Aggregation OpenCL kernel.*

of workgroups is the total number of disparity hypotheses divided by the number of hypotheses evaluated in one cycle by each workgroup; in turn, the number of hypotheses evaluated in one cycle is defined by parameter *nb_hypo_per_wg*, described later:

$$num\_disparities_{tot} = \left\lfloor \frac{max\_hypo\_value}{hypo\_step} \right\rfloor + 1 \tag{3.2}$$

$$nb\_workgroups_{max} = \left\lceil \frac{num\_disparities_{tot}}{nb\_hypo\_per\_wg} \right\rceil \tag{3.3}$$

Workgroup size (*nb_wi_per_wg*). This parameter specifies the workgroup size: by reducing the workgroup size, it becomes possible to schedule several workgroups on the same cluster. We have set *nb_wi_per_wg* to 16 (the number of PEs available on one cluster) given the features of the target platform. However, this parameter must be tuned according to the accelerator architecture: other platforms might provide a lower parallelism within the workgroup, in this case it is important to reduce the workgroup size and to exploit the parallelism between workgroups.

In our application, the work-items work on different image rows, each one processing a number of pixels specified by *wg_col_pixel_width*. Figure 3.3 shows the execution model for *CostAggregation* kernel: this kernel is slightly different from the other ones, since more than one PE can work on the same image row but with a different disparity value (see *nb_hypo_per_wg*).

Number of pixels processed per work-item (*wg_col_pixel_width*). Our OpenCL kernels never access external memory directly, data is first copied to local memory (L1, 256 KB on cluster) by means of asynchronous DMA transfers at workgroup level. In order to compensate for the DMA latency,

we have implemented a software pipeline (Figure 3.4) so that asynchronous memory transfers – global-to-local (read) and local-to-global (write) – are executed in parallel to data processing [77]. The application also exploits FIFO buffers allocated in local memory to pipeline different stages of data processing, for example horizontal and vertical matching-cost integration.

The parameter *wg_col_pixel_width* allows to indirectly control the size of DMA transfers. As shown in Figure 3.3, *wg_col_pixel_width* specifies the length of the row of pixels to be processed by one work-item: thus, tuning of this parameter makes it possible to compensate the DMA latency with the processing time in the pipeline stage.

Thus, the configuration of application parameters changes the pipeline properties: not only the stage duration, as explained before, but also the latency (*max_arm_blocks*, as shown in Figure 3.4). The relation between *max_arm_blocks* and the application parameters is described below:

$$block\_size = \frac{nb\_wi\_per\_wg}{nb\_hypo\_per\_wg} \tag{3.4}$$

$$max\_arm\_blocks = \left\lceil \frac{max\_arm\_length}{block\_size} \right\rceil \tag{3.5}$$

Number of disparity hypotheses for each workgroup (*nb_hypo_per_wg*). This parameter introduces a third dimension in the *NDRange* grid of *CostAggregation* kernel (see Figure 3.3). In Equation 3.4, *block_size* is the number of pixel rows processed within one pipeline stage; however, there might be more than one work-item processing the same line but evaluating different disparity hypotheses (as many as specified by *nb_hypo_per_wg*). For some configurations of the parameters (as we will show in Section 3.4), increasing *nb_hypo_per_wg* might give better results in terms of performance and memory utilization.

3.3 The DSE methodology for application customization

The goal of *Design Space Exploration* (DSE) is to analyze and tune application-specific parameters and resource parameters to minimize the *disparity error* and maximize the application *throughput*, also taking into account the variability on input data and run-time workload.

The *disparity error* is defined as the error of the algorithm in computing the actual disparity of the pixels in the stereo frames. To evaluate the *disparity error*, we used a set of seven bitmap image-pairs (three color components per pixel, encoded on 24bit) chosen from the Middlebury stereo

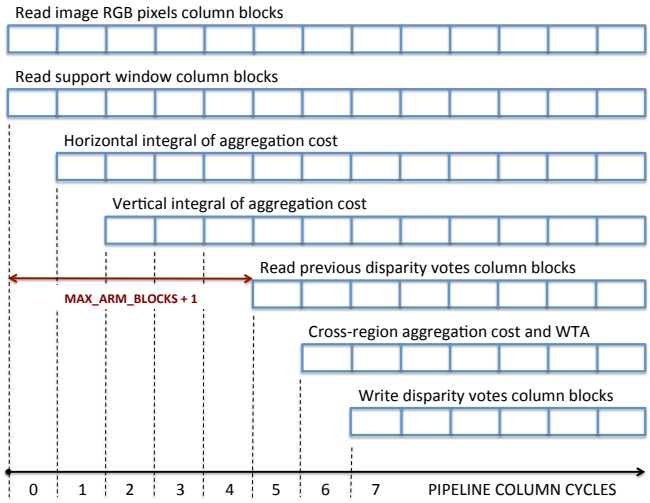


Figure 3.4: *Software pipeline for CostAggregation kernel, with max_arm_blocks=4 (e.g.* nb_wi_per_wg=*16,* nb_hypo_per_wg=*4 and* max_arm_length=*16).*

datasets [102]. Each reference image-pair has a corresponding *truth* disparity map which is used to compute the *disparity error* as the *average intensity difference per pixel* in the seven disparity maps.

Since we have to deal with a minimization problem, the other objective was set to *delay per pixel* (execution time divided by the number of pixels processed by the application), that is the inverse of *throughput*. This metric is computed as the geometric average value of *delay per pixel* over the seven images in the set.

The output of the methodology is a set of Operating Points (OPs), i.e. tuples composed of application-specific parameter and resource parameter values providing optimal *disparity error* and *delay per pixel*. Since there are two target objectives, the definition of optimality is not unique [57]; thus the final output of our methodology is a Pareto set of configurations.

Finally, the Pareto set of operating points can be used by either the application designer, at deployment time, or by a suitable run-time management layer to trade off *Quality of Service* (QoS) with performance [79]. In fact, the original parameter space is too wide to make accurate decisions about the optimal configurations, both for the developer or the run-time layer. Moreover, workloads or QoS requirements (such as the *disparity error*) can change dynamically, requiring dynamic adaptation of the parameter configuration. Our methodology prunes the parameter space to identify only those important to take this decision.

An OP for the stereo-matching application is formally defined by a tuple:

$$c = \langle \boldsymbol{\alpha}, \boldsymbol{\rho}, \delta, \epsilon \rangle \tag{3.6}$$

where:

- $\boldsymbol{\alpha}$ and $\boldsymbol{\rho}$ are two arrays of values, the first one for application parameters and the second one for resource parameters.

$$\boldsymbol{\alpha} = \begin{bmatrix} color\_threshold \\ max\_arm\_length \\ hypo\_step \\ max\_hypo\_value \\ matchcost\_limit \end{bmatrix} \in A \tag{3.7}$$

$$\boldsymbol{\rho} = \begin{bmatrix} nb\_workgroups \\ nb\_wi\_per\_wg \\ wg\_col\_pixel\_width \\ nb\_hypo\_per\_wg \end{bmatrix} \in R \tag{3.8}$$

  where both A and R are finite, discrete domains: $A \subset N_0^5$, $R \subset N_0^4$ (the possible values are reported in Table 3.1).

- δ is the *delay per pixel*, which depends on both $\boldsymbol{\alpha}$ and $\boldsymbol{\rho}$:

$$\delta = f_1(\alpha, \rho) \tag{3.9}$$

- ϵ is the *disparity error*, which only depends on $\boldsymbol{\alpha}$:

$$\epsilon = f_2(\alpha) \tag{3.10}$$

The multi-objective optimization problem is thus defined as a set of two objective functions to be minimized:

$$\min_{\boldsymbol{\alpha} \in A, \rho \in R} \boldsymbol{f}(\boldsymbol{\alpha}, \boldsymbol{\rho}) = \begin{bmatrix} f_1(\boldsymbol{\alpha}, \boldsymbol{\rho}) \\ f_2(\boldsymbol{\alpha}) \end{bmatrix} \in R^2 \tag{3.11}$$

However, each configuration of parameters $c = \langle \boldsymbol{\alpha}, \boldsymbol{\rho} \rangle$ requires the allocation of a certain amount of buffers in local memory when deployed on the target device. Thus, the choice of $\boldsymbol{\alpha}$ and $\boldsymbol{\rho}$ is also subject to the following constraint:

$$m(\boldsymbol{\alpha}', \boldsymbol{\rho}') \leq M \; with \; \boldsymbol{\alpha}' \subseteq \boldsymbol{\alpha}, \boldsymbol{\rho}' \subseteq \boldsymbol{\rho} \tag{3.12}$$

where M is the size of local memory available on the device and the function m quantifies the memory usage for a given configuration of parameters.

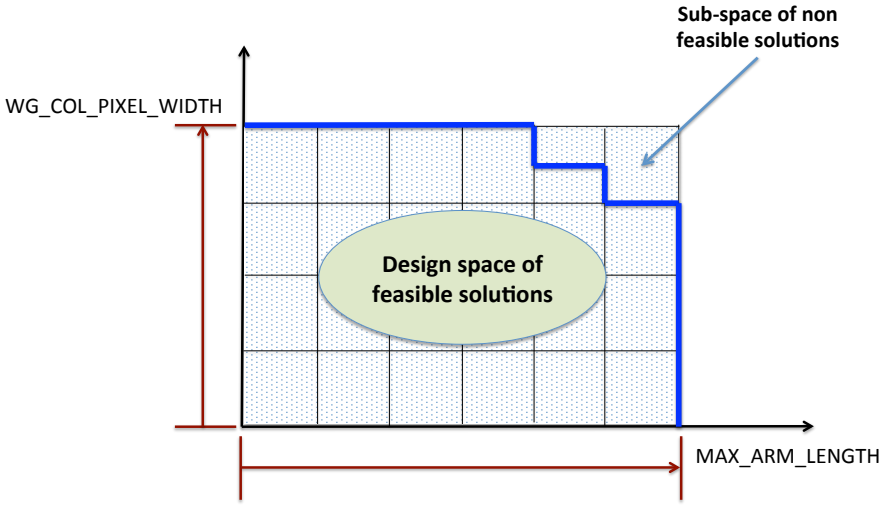


Figure 3.5: *Irregular space of feasible solutions due to the constraint of local memory size.*

The DSE methodology proposed in this thesis, based on a parameter inter-dependency model [48], aims at reducing the space of application configurations by exploiting relations between parameters. Equation 3.12 points out that **the size of local memory poses a constraint on the feasible solutions** for the stereo-matching application, but such constraint can be applied to any OpenCL application with shared buffers. In particular, it creates inter-dependencies among application and resource parameters that should be taken into account. This also means that the multi-dimensional subspace of feasible solutions has an irregular shape (e.g., Figure 3.5).

The proposed exploration strategy consists of three main phases, presented below.

Phase 1: Random sampling of the design space. We first apply a random sampling over all parameters on the complete reference dataset. In this phase we apply a twofold preliminary analysis of the results:

- **Analysis of correlation:** we calculate the correlation between input parameters and measured metrics. A high value of correlation often means that there is some kind of dependency: this analysis provides an estimate of the impact of parameters on the metrics.

- **Analysis of standard deviation:** the standard deviation is a measure of variability. We are mostly interested in the variability of metrics *disparity error* and *delay per pixel* with respect to the input data. This

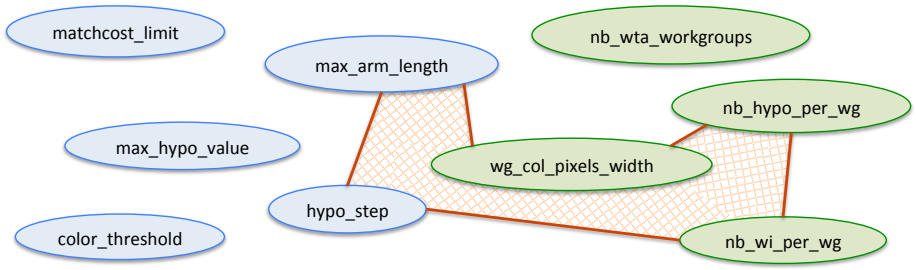


Figure 3.6: *Inter-dependency graph of application (blue) and resource (green) parameters.*

information is used to determine if the image pairs in the dataset can be pruned by using only one representative image pair.

Phase 2: Exploration of resource parameters and inter-dependent parameters. According to Equation 3.10, the *disparity error* is independent of the resource parameters, i.e. the quality of the result does not depend, in our implementation, on the configuration of the *NDRange* grid. By exploiting this property, the goal of this phase is to identify the configuration of *resource parameters* that minimizes the *delay per pixel* metric.

We manually identify the dependencies between parameters and draw a dependency graph (Figure 3.6) where nodes represent the input parameters and edges the inter-dependencies: "A path from node A to a node B and back to A, which forms a cycle, indicates that the Pareto-optimal configurations of all the parameters on the cycle need to be calculated simultaneously" [48]. As expressed by Equation 3.12, not all parameters necessarily impact on the memory usage: thus, the set of inter-dependent parameters consists of the subset $\boldsymbol{\alpha}' \cup \boldsymbol{\rho}'$. For all the inter-dependent parameters a full combinatorial search is done. The other parameters can be considered separately in the next phase since their choice is not constrained by the memory size. Besides, we explore the configurations of inter-dependent parameters on the single representative image pair identified in phase 1. Eventually, this phase allows for identification of the best configuration of *resource parameters* for each configuration of inter-dependent *application parameters*.

Phase 3: Refinement of application parameters on a high-level simulation model. We refine the Pareto-optimal solutions found at step 2 by considering all the image-pairs in the reference data-set and by exploring the values of the remaining independent application parameters. Since they impact mainly on the *disparity error*, we show that it is possible to do this analysis by using a fast simulation model instead of the Instruction Set

Table 3.1: *Design space of the Stereo-Matching OpenCL application on STHORM. Step* exp2 *for* nb_hypo_per_wg *means that this parameter takes power-of-2 values (1, 2, 4).*

parameter	min	max	step
hypo_step	1	3	1
color_threshold	14	64	10
max_arm_length	1	18	1
matchcost_limit	60	60	–
max_hypo_value	200	200	–
nb_wi_per_wg	16	16	–
nb_hypo_per_wg	1	4	exp2
nb_workgroups	1	1	–
wg_col_pixel_width	64	96	16

Simulator (ISS) – the *posix-xp70* configuration of the STHORM simulation model. Although the fast simulation model introduces some approximation, this optimization technique provides a significant speedup that will be quantified in Section 3.4.

The resulting operating points are Pareto-filtered leaving a set of optimal trade-offs in terms of *throughput* and *disparity error*.

3.4 Experimental results

In this section we apply the proposed methodology to the OpenCL stereo-matching application: the goal is to optimize the application for STHORM (P2012) by customizing application parameters. In particular, we follow the three phases described in Section 3.3 to efficiently derive a set of Pareto-optimal operating points in terms of *disparity error* and *delay per pixel*.

Table 3.1 describes the design space of our stereo-matching application: an important preliminary task, actually, is to define the design space. We have assigned a constant value to the parameters *matchcost_limit* and *max_hypo_value*, based on the characteristics of input data (seven images from the Middlebury stereo dataset). Indeed, the experimental results have been generated by using a single-cluster configuration for the STHORM simulator (multi-cluster is not supported by the SDK version we have used). For this reason, parameter *nb_workgroups* has not been explored (fixed to 1). Since the provided SDK does not allow multiple workgroups to be executed concurrently on the same cluster, also parameter *nb_wi_per_wg* is constant, fixed to the number of PEs available on the cluster.

Considering all possible values for each parameter in our design space, the total number of potential configurations is 3888. Since these configura-

tions must be profiled on a dataset consisting of 7 images, 27216 ISS-based simulations are required for an exhaustive exploration of the application parameters. This is a very large number considering that it would have required approximately 1.5 years by using the cycle-accurate simulator on a modern workstation (around 30 minutes for each simulation). We will show that our methodology allows to reduce the time associated with design exploration to derive a Pareto-approximate set of configurations.

3.4.1 DSE Results of Phase 1

We first consider a Design-of-Experiments (DoE) consisting of a random sampling of the entire design space; these configurations are explored on the cycle-accurate STHORM simulator considering all images in the dataset. The sample was sized in order for the exploration to satisfy a given temporal constraint (100 design points, so that this task did not require longer than two weeks on a modern workstation). Then, we perform the following analyses.

Analysis of correlation. A first analysis of the correlation values in Table 3.2 points out the parameters that mostly affect the metric *delay per pixel*: *hypo_step* and *wg_col_pixel_width*. We can also see that *color_threshold* has almost no impact on the execution time, while there is a significant correlation between *color_threshold* and *disparity error* (this result will be useful in the third exploration phase).

The correlation value between parameter *wg_col_pixel_width* and metric *disparity error* is due to the irregular shape of the space of feasible solutions (Figure 3.5), since the resource parameters do not change the accuracy of results (see Section 3.2.2). For example, it might not be possible to execute on STHORM a configuration of the stereo-matching application with a high value for both parameters *max_arm_length* and *wg_col_pixel_width*. As a consequence, the feasible solutions with high *wg_col_pixel_width* must have low values of *max_arm_length*, thus the accuracy of results will be low because of the strong negative correlation (-38.0%) between parameters *max_arm_length* and *disparity error*.

Analysis of standard deviation. Table 3.3 shows the average coefficient of variation (also known as Relative Standard Deviation – RSD) of metrics *delay per pixel* and *disparity error* over all random configurations of parameters for each image in the dataset; the last row reports the RSD of the average value of application metrics considering the whole set of seven images. The image with average RSD closest to the average RSD of the dataset is "*cones*", thus this image was selected as representative of the dataset in phase 2.

Table 3.2: *Phase 1, correlation analysis after random sampling.*

Parameters/Objectives	***delay per pixel*** (μs)	***disparity error***
hypo_step	-76.5%	57.4%
color_threshold	1.6%	-12.1%
max_arm_length	12.9%	-38.0%
nb_hypo_per_wg	-13.3%	1.0%
wg_col_pixel_width	-47.2%	23.9%

Table 3.3: *Phase 1, Relative Standard Deviation (RSD) for the dataset.*

Image/RSD	***delay per pixel***	***disparity error***
baby	1.55×10^{-2}	0.88
barn	1.69×10^{-2}	0.73
cones	2.20×10^{-2}	0.63
sawtooth	1.19×10^{-2}	0.47
tsukuba	1.30×10^{-2}	0.46
teddy	1.72×10^{-2}	0.47
venus	0.81×10^{-2}	0.30
dataset	1.79×10^{-2}	0.61

3.4.2 DSE Results of Phase 2

As we noted above, the *disparity error* is independent from the resource parameters. Since resource parameters have some impact only on the *delay per pixel*, the goal of this phase is to identify the configuration of *resource parameters* that minimizes the *delay per pixel*. However, the memory constraint on the feasible solutions creates inter-dependencies between application and resource parameters that should be taken into account to trade-off *disparity error* and *delay per pixel*.

The parameters selected by the inter-dependency analysis include:

- application parameters: *hypo_step*, *max_arm_length*
- resource parameters: *nb_hypo_per_wg*, *wg_col_pixel_width*.

In this way the number of simulations has been reduced to 648. Note that in our exploration *nb_wi_per_wg* is not considered: its value is kept constant to 16, *i.e.* the number of processing elements on the STHORM ENCore, because the provided SDK does not support two kernels on the same cluster.

As said above, the configurations must be profiled with the cycle-accurate simulator. We observe from Table 3.3 that *delay per pixel* does not vary significantly with the input image set (average RSD 1.8%). Since in this phase we are mainly interested in optimizing *delay per pixel*, we decided to

use a single representative image (*cones*) for the whole dataset.

The plots in Figure 3.7 show the results in the *disparity error* vs *delay per pixel* objective space with reference to the values of different parameters. Whenever a clear behavior can be noticed, an arrow has been plotted: in such cases, it is possible to infer that a dependence exists between the parameter and the measured metrics.

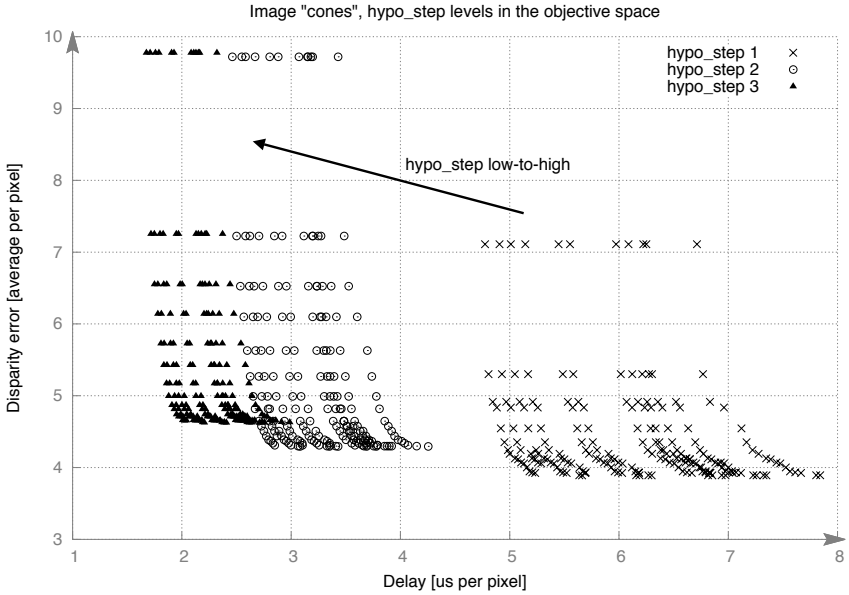
Figure 3.7a focuses on *hypo_step* levels: this parameter is the most significant for the stereo-matching application because it enables the identification of three clusters of operating points. This also means that any change to *hypo_step* has a direct effect on the measured metrics (as highlighted by the arrow). Within each cluster of configurations defined by *hypo_step*, parameter *max_arm_length* allows for additional fine-tuning of the trade-off between *delay per pixel* and *disparity error*: in particular, *disparity error* decreases with higher values of *max_arm_length*. Indeed, in Figure 3.7b we can notice different horizontal rows of configurations, each one characterized by a certain configuration of *application parameters* (*hypo_step* and *max_arm_length*). Thus, the points belonging to the same row have the same value of metric *disparity error* and only the configuration of *resource parameters* changes.

Figure 3.7c and Figure 3.7d show the objective space for parameters *nb_hypo_per_wg* and *wg_col_pixel_width*, respectively: the best configuration of *resource parameters* is *nb_hypo_per_wg* = 2 and *wg_col_pixel_width* = 96, independently from the configuration of other inter-dependent *application parameters*. However, the choice of *nb_hypo_per_wg* = 2 and *wg_col_pixel_width* = 96 does not represent a feasible solution together with some values of *max_arm_length* (e.g. 17 and 18, see Figure 3.5); thus, for these configurations we select a smaller value of *wg_col_pixel_width* (80).

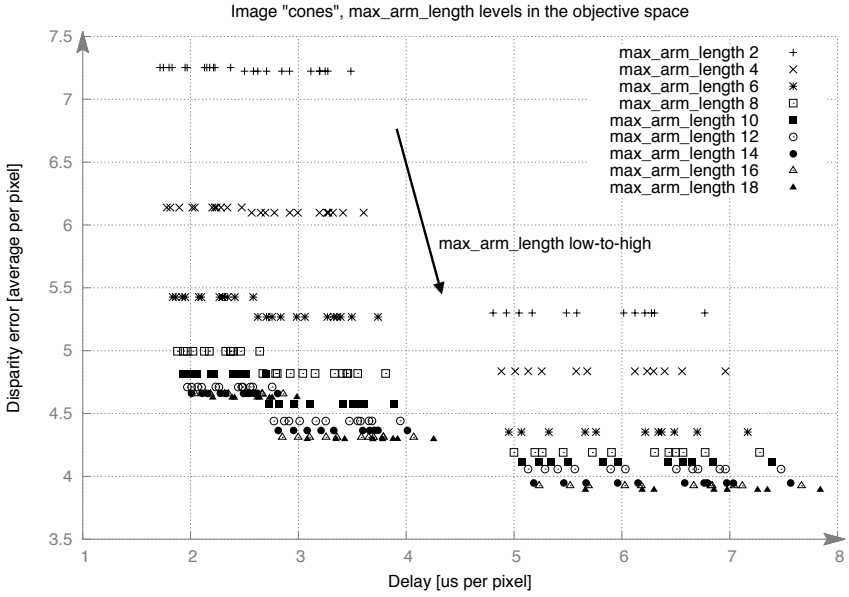
3.4.3 DSE Results of Phase 3

This phase is aimed at identifying the configurations of application parameters that optimize both the *disparity error* and the *delay per pixel* (*color_threshold*, *max_arm_length* and *hypo_step*). For this reason, we include in the DoE also those parameters not considered in the previous phase (*color_threshold*), i.e. the parameters independent from the design constraint (the size of local memory in our implementation).

In principle, this could be done by just exploring all the configurations on the cycle-accurate simulator for the entire dataset: this means 324 configurations for each image. However, the results of the correlation analysis in Table 3.2 show that *color_threshold* almost has no impact on *delay per pixel*.

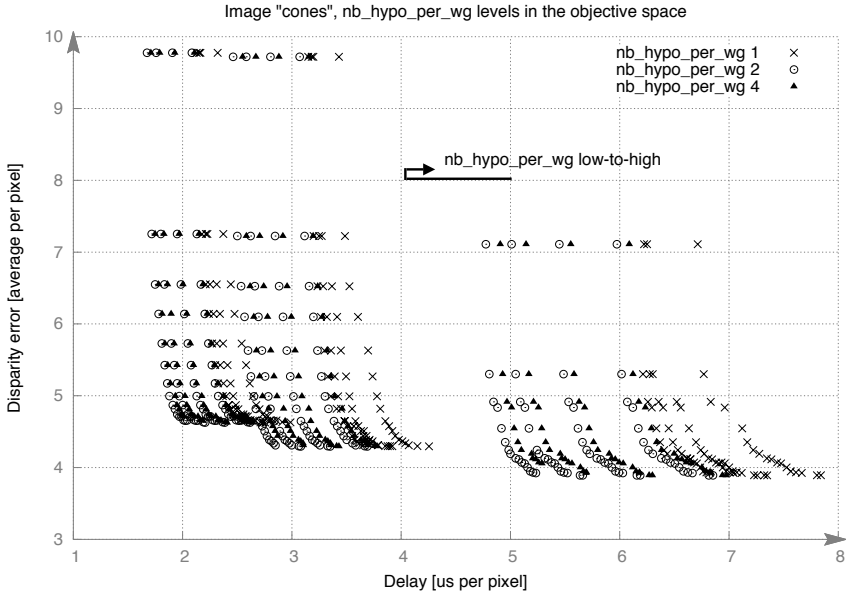


(a) *Phase 2, objectives space for* hypo_step

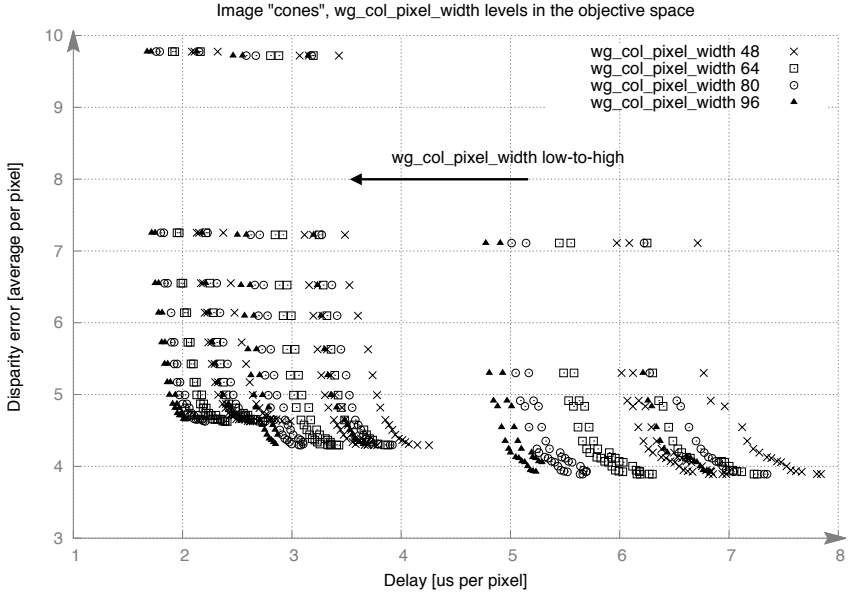


(b) *Phase 2, objectives space for* max_arm_length

Figure 3.7: *Phase 2, results in the objective space* disparity error *vs* delay per pixel *for parameters* hypo_step *(3.7a) and* max_arm_length *(3.7b).*



(c) *Phase 2, objectives space for* nb_hypo_per_wg



(d) *Phase 2, objectives space for* wg_col_pixel_width

Figure 3.7: *Phase 2, results in the objective space* disparity error *vs* delay per pixel *for parameters* nb_hypo_per_wg *(3.7c) and* wg_col_pixel_width *(3.7d).*

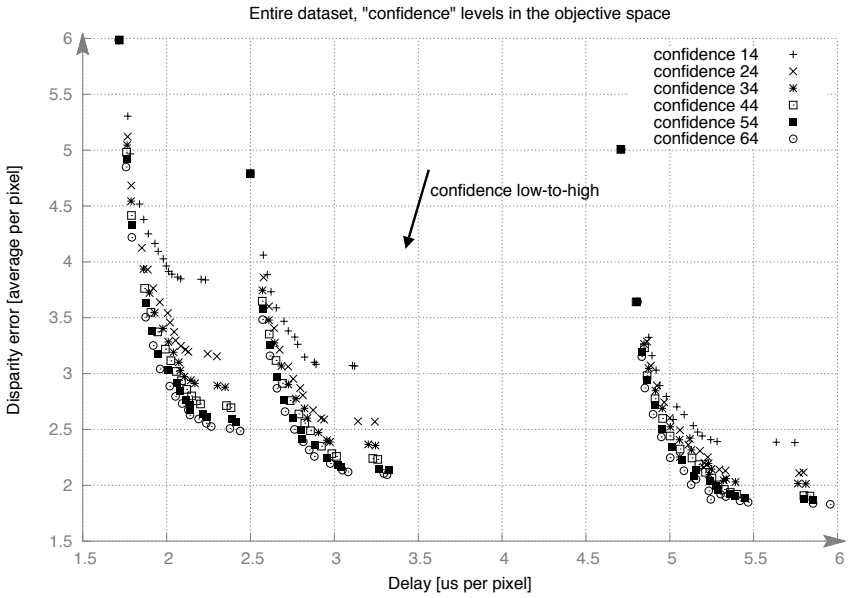


Figure 3.8: *Phase 3, objectives space* disparity error *vs* delay per pixel *for parameter* color_threshold.

Thus, we split this exploration in the following sub-phases:

1. we use the cycle-accurate simulator to profile the average *delay per pixel* δ for one value of *color_threshold* (54 configurations $\times$ 7 images);

2. we use a high-level simulation model to derive the *disparity error* and the *delay per pixel* for all combinations of *color_threshold*, *max_arm_-length* and *hypo_step*.

3. we scale the *delay per pixel* computed in the previous step according to δ.

The high-level simulation model is based on the native OpenCL runtime of a 4-cluster Non-Uniform Memory Access (NUMA) AMD machine. OpenCL being a cross-platform standard, the stereo-matching application can be compiled and executed on this machine. While it is not needed to change the application implementation, the configuration of resource parameters must be modified in order to meet a tighter constraint of local memory size (32KB). Figure 3.8 shows the levels of parameter *color_threshold* in the objective space. It is still possible to identify three clusters of configurations corresponding to the three values of *hypo_step*. Within each cluster it is

possible to achieve different trade-offs between throughput and accuracy by fine-tuning *color_threshold* and *max_arm_length*.

By Pareto-filtering the final points, we find the optimal solutions represented in Figure 3.9. In order to be able to quantify the loss in accuracy caused by introducing the high-level model, we have also run phase 3 entirely on the cycle-accurate simulator. This approach gives us accurate profiling information for the execution time of all configurations: the Pareto-optimal solutions calculated on this database will be referred to as reference Pareto set in the following.

In order to quantify the loss in accuracy, we use the Average Distance from Reference Set (ADRS) metric to measure the distance between the exact Pareto set $\Pi = \Psi(\Phi)$ and the approximate Pareto set $\Lambda = \Psi(\Omega)$, as defined in [32]:

$$\text{ADRS}(\Pi, \Lambda) = \frac{1}{|\Pi|} \sum_{\boldsymbol{x}_R \in \Pi} \left(\min_{\boldsymbol{x}_A \in \Lambda} \{\delta(\boldsymbol{x}_R, \boldsymbol{x}_A)\} \right) \tag{3.13}$$

where δ is a measure of the normalized distance in the objective function space of two configurations:

$$\delta(\boldsymbol{x}_R, \boldsymbol{x}_A) = \max_{j=1,\ldots,m} \left\{ 0, \frac{f_j(\boldsymbol{x}_A) - f_j(\boldsymbol{x}_R)}{f_j(\boldsymbol{x}_R)} \right\} \tag{3.14}$$

The ADRS is usually measured in terms of percentage; the higher the ADRS, the worst is Λ with respect to Π. We calculate the ADRS between the approximate Pareto set obtained with the high-level simulation model and the reference Pareto set obtained with the cycle-accurate simulator: in our tests, the ADRS is less than 3%.

3.4.4 Generation of operating points from the Pareto-set

The approximate Pareto-set obtained in Phase 3 has been used to generate a set of configurations for the stereo-matching application. We decided to cluster the configurations into 10 possible sets and to identify one configuration representative of each cluster (centroid): see the plot in Figure 3.9 and the list of operating points in Table 3.4.

The different symbols in Figure 3.9 represent different clusters, plotted in the objective space of *disparity error* and *throughput* (Kpixel/s). The operating points in Table 3.4 are sorted by *disparity error*. In this case, it happens that they are sorted by *throughput*, so it is easy to identify different trade-offs between performance and QoS.

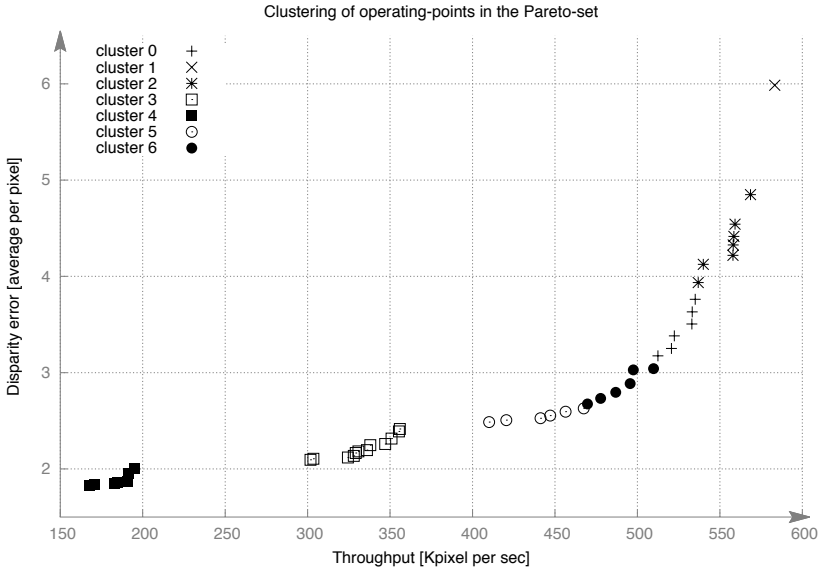


Figure 3.9: *Clustering of Pareto-solutions for generation of operating points.*

The operating points identified by our methodology allow to optimize the application targeted to STHORM, thus the first result achieved is **performance portability**. The same methodology could be used to customize the application for another device, in which case the operating points would probably have different configurations of parameters depending on the target architecture. The operating points could also be exploited by a Run-Time Manager, as described in Chapter 6, to enable **application auto-tuning**: this allows for application self-reconfiguration – by changing operating point – in presence of dynamic QoS requirements or workload variations.

Table 3.4: *List of operating points for stereo-matching on STHORM with single-cluster configuration. Parameters are* hypo_step *(h),* color_threshold *(c),* max_arm_length *(m),* nb_hypo_per_wg *(n),* wg_col_pixel_width *(w).*

h	**c**	**m**	**n**	**w**	***disparity error***	***throughput*** **(Kpixel/sec)**
1	64	18	2	80	1.83	167.88
2	64	18	2	80	2.09	301.72
3	64	18	2	80	2.49	410.11
3	64	12	2	96	2.67	469.76
3	64	7	2	96	3.25	520.62
3	64	4	2	96	4.22	557.93
3	54	2	2	96	5.99	583.29

3.5 Conclusions

This chapter presented a parametric design for an OpenCL application and a methodology to efficiently derive the optimal customization options (*operating points*). The proposed DSE heuristic provides a speed-up by reducing the overall number of simulations and exploiting a fast high-level simulation model. An exhaustive exploration of the application design space targeted to STMicroelectronics STHORM (P2012) requires 27216 simulations, which correspond to approximately 1.5 years simulation time on a modern workstation. With this methodology, only 1672 cycle-accurate simulations are needed, while the simulation time on the high-level model can be neglected. In conclusion, the overall time speed-up is 16x, while the accuracy of the solution with respect to an exhaustive exploration is very good ($ADRS < 3\%$).

Although in this chapter the proposed methodology has been applied to the Stereo-Matching application, it is general enough to be used for customizing any OpenCL application on a different platform (as discussed in Chapter 5), provided that the application exposes a set of parameters to control both the QoS/performance trade-off and the usage of platform resources. The exploration time could be further reduced by applying orthogonal complementary approaches such as a) executing simulations in parallel on a cluster (linear speed-up with the number of nodes), and b) replacing the cycle-accurate simulator with a cycle-approximate one, at the expenses of result accuracy. Next chapter will discuss the second alternative and will propose a modeling technique to provide even better accuracy, by combining cycle-approximate and cycle-accurate simulations.

CHAPTER 4

Ensemble Models for Simulation of Many-core Platforms

The technique presented in Chapter 3 exploited a search heuristic to efficiently explore the design space of an OpenCL application targeted to a many-core embedded platform. Thus, it was aimed at reducing the number of configurations to be simulated on the cycle-accurate platform model. However, the problem was exacerbated by the long simulation time associated with each configuration of the applications running on the cycle-accurate virtual platform. In this chapter, we introduce a modeling methodology, inspired by *ensemble learning* [91], that leverages different simulation models to further reduce the overall simulation time.

A Low-Level, highly accurate model $\mathcal{M}_{LL}$ allows, given an application/architecture configuration x, to derive an estimate y of a specific metric:

$$y = \mathcal{M}_{LL}(x) \tag{4.1}$$

Artificial Neural Networks (ANNs) have already been used effectively as a surrogate of $\mathcal{M}_{LL}$ [92]. The final model is a closed form expression that can be used to predict with reasonable accuracy the target system metric provided by cycle-accurate simulators. However, before being useful, the

model must be trained and the training time can be long depending on the target accuracy. Thus, in this chapter $\mathcal{M}_{LL}$ is combined with a High-Level – thus faster but less accurate – model $\mathcal{M}_{HL}$:

$$\hat{y} = \mathcal{M}_{HL}(x) \tag{4.2}$$

The main contribution of this chapter is a methodology – published in [6] – to combine $\mathcal{M}_{HL}$ and $\mathcal{M}_{LL}$ in order to exceed the speed/accuracy trade-off attainable with state-of-the-art methods such as those presented in [92]. This goal is achieved by applying techniques borrowed from *machine classification*, *statistical analysis of variance* and *multi-objective analysis*.

4.1 Related Work

The efficiency of a Design Space Exploration (DSE) strategy is the trade-off between the simulation time and the accuracy of a multi-objective optimization solution. The accuracy, in turn, depends on the distance of an approximate Pareto-set from the exact one. In simulation-based DSE, the exact Pareto-set can only be identified by Pareto-filtering all the design points, after they have been profiled on the target simulation model to evaluate the objective functions (e.g. performance or power consumption). This represents one main difference between profiled-based DSE and analytical DSE [64]: while in profiled-based DSE the design itself represents a *black box* and design metrics can only be retrieved by means of simulation, in analytical DSE the design properties are known and can be used to formulate the optimization problem in an analytical form.

However, applicability of analytical DSE is limited by the complexity of the design and the type of metrics that have to be profiled. For example, estimation of Worst-Case Execution Time (WCET) for a complex application on a multi-core platform can be done analytically [23, 97], once the WCETs of application tasks are known and the platform guarantees real-time processing and predictable communication latency between the cores. On the contrary, power consumption metrics are more difficult to capture in an analytical representation.

Conversely, simulation-based DSE can be applied to a wider range of design optimization problems, potentially to any type of design provided that a simulation model is available. However, the main problem is the exploration time since, for most kinds of design like many-core computing fabrics as in this chapter, evaluating design metrics of a single system configuration means hours or days of simulation under a realistic workload.

In order to minimize the number of simulations to be executed during the DSE phase, a common approach is to use Design of Experiments (DoEs) and Response Surface Models (RSMs) combined with suitable multi-objective minimization or maximization techniques.

The DoE is typically used to identify an initial plan of simulations to provide the designer with a coarse-grained view of the architectural design space. Each DoE plan differs in terms of the layout of the selected design points in the design space. Consolidated approaches are either based on random sampling or more sophisticated techniques like Box-Behnken and Latin Hypercube designs [101]. However, the designer could also define a custom set of initial experiments, which is usually done after identification of a set of feasible design solutions [53].

RSM techniques are introduced after the initial experimental design to analyze the system response of unknown configurations without incurring into additional delay due to simulation [61]. Indeed, an RSM is a closed form analytical meta-model of the real simulation model: examples of RSMs, used for DSE and analyzed in [92], are linear regression, Shepards interpolation, neural networks and Radial Basis Functions (RBFs).

The evaluation of such analytical expression is typically one or more orders of magnitude faster than real simulation for applications of commercial interest, depending on the design complexity and the system resources dedicated to the simulation. Thus, RSMs have been used to derive main and interaction effect analysis [61, 73] or to extend traditional optimization as in *meta-model assisted optimization*. In particular, they have been used in *iterative refinement* approaches [92], as a filtering criterion to exclude from optimization the worst configurations [41], and even to identify the best experiments to be done in order to improve the model itself [70].

Before being useful, an RSM must be trained and the training samples, typically, are obtained from a single simulation model of the target design. However, different models (cycle-accurate, cycle-approximate, functional) can provide different overlapped views of the same simulated platform, in particular for simulation of parallel applications on many-core computing platforms. In this chapter, a DSE methodology based on neural networks (e.g. [92]) is extended with techniques borrowed from ensemble modeling. Ensemble methods [91] use multiple models to obtain better predictive performance than it could be obtained from any of the constituent models, provided that a suitable aggregation technique is used.

The research on techniques for combining the predictions of multiple classifiers to produce a single classifier goes back to the 90's [25, 44]. Both theoretical and empirical research has demonstrated that a good ensemble

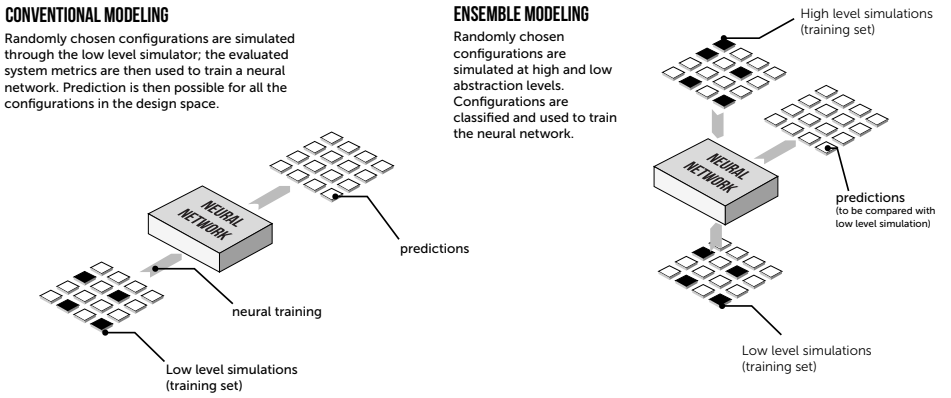


Figure 4.1: *Conventional modeling (left) vs. the proposed ensemble modeling (right).*

is one where the individual classifiers in the ensemble are both accurate and make their errors on different parts of the input space [91]. Indeed, aggregation techniques rely on *resampling* to obtain different training sets for each of the classifiers, in order to reduce the risk of model over-fitting.

However, the technique proposed in this chapter differs from traditional ensemble modeling. While a pure ensemble is a technique for combining many weak learners (as defined in [91]) in an attempt to produce a strong learner, here we are validating a new technique that attempts to combine weak learners (a cycle-approximate simulation model) and strong learners (a cycle-accurate one) to improve the efficiency of the ensemble model, in terms of training time and prediction accuracy.

Therefore, the aggregation technique proposed here is inspired by *model stacking*. As defined in [120], stacked ensemble modeling works by deducing the biases of the models with respect to a provided learning set. When used with multiple models, stacked modeling can be seen as a more sophisticated version of cross-validation, exploiting an aggregation meta-model for combining the individual models instead of cross-validation's based on winner-takes-all. When used with a single model, stacked generalization is a scheme for estimating and then correcting for the prediction error [120].

4.2 Ensemble modeling of applications and architectures

Conventional modeling tries to overcome lengthy simulations by using closed form models (such as neural networks [92], see Figure 4.1). In this scenario, low level simulations are used to train the model by tuning its structure to minimize a certain measure of error. The error can be computed

by considering either configurations not belonging to the training set or belonging to a larger set (potentially as large as the design space).

Closed form models are ideally very *light* given their straightforward mathematical representation. The main drawback, however, is due to the time needed to simulate the training samples. In our approach, we assume that $\mathcal{M}_{LL}$ has already been used to derive a suitable neural surrogate meta-model. As it can be observed in Figure 4.1, the final goal of the model is to make educated *predictions* about the remaining set of configurations, by resorting to a selected random subset of training simulations.

In the proposed technique, a conventional meta-model is extended to an *ensemble* model that, taking into account the information coming from a higher level of abstraction $\mathcal{M}_{HL}$, provides either better accuracy or better performance (see again Figure 4.1).

The methodology is based on the following phases:

- Identify the **structure of the ensemble model**. In the case of the neural network, this consists of identifying the number of hidden layers and the number of neurons per layer, by analyzing the associated design space. Since this problem can be of exponential complexity, suitable heuristics should be applied.

- Identify the correct **percentage of training samples** to be used for both $\mathcal{M}_{HL}$ and $\mathcal{M}_{LL}$ and the overall training set size. This should be done by considering the relative length of simulation of each configuration, which might depend on the target architecture, software resource usage and/or application-level parameters.

These phases are basically aimed at characterizing statistically the **distribution of the prediction errors** and assess the statistical properties of the model. This information is used, at the end, to configure an ensemble model in the framework shown in Figure 4.2.

4.2.1 Problem definition

The methodology proposed in this thesis for ensemble modeling considers the following parameters:

- The **number of hidden layers** and the **number of neurons per layer** in the Artificial Neural Network (ANN) configuration, which change the internal geometry of the ANN.

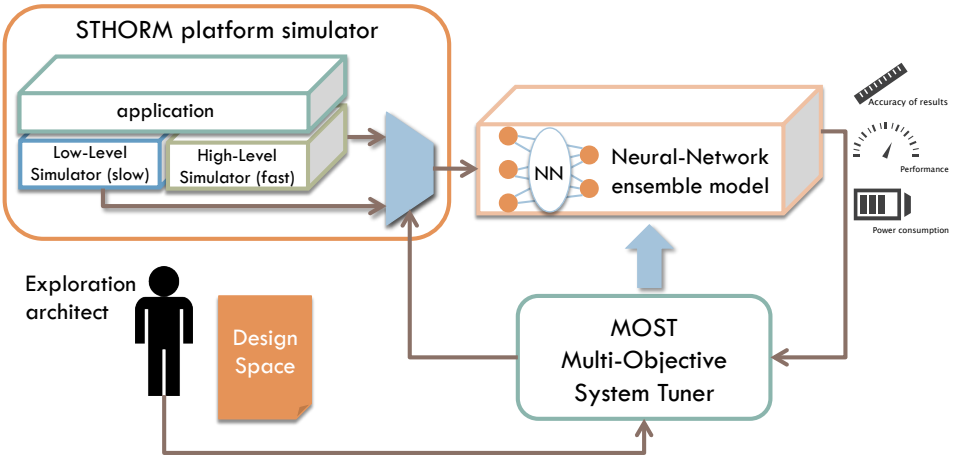


Figure 4.2: *DSE framework based on ensemble modeling of different simulation platforms.*

- The composition of the training set in terms of $\mathcal{M}_{LL}$ and $\mathcal{M}_{HL}$ samples, considered in proportion to the total number of points in the application design space.

A *design point* (either produced by $\mathcal{M}_{LL}$ or $\mathcal{M}_{HL}$) is defined as a tuple:

$$c = \langle \boldsymbol{\alpha}, \boldsymbol{\pi} \rangle \tag{4.3}$$

where:

- $\boldsymbol{\alpha}$ is the array of values for application parameters (also referred to as input configuration);
- $\boldsymbol{\pi}$ is the array of metric values associated to the input configuration $\boldsymbol{\alpha}$.

The possible values for $\boldsymbol{\alpha}$ depend on the design space for the specific application to be explored. In order to distinguish between $\mathcal{M}_{LL}$ or $\mathcal{M}_{HL}$ points, we apply a coloring technique. Beside the application input parameters, we add a flag parameter (either 0 or 1) to each training sample:

$$c = \langle \boldsymbol{\alpha}', \boldsymbol{\pi} \rangle \tag{4.4}$$

$$\boldsymbol{\alpha}' = [\alpha_0, \ldots, \alpha_{n-1}, \phi] \tag{4.5}$$

where ϕ is the flag to classify the two types of training configurations:

$$c = \langle [\alpha_0, \ldots, \alpha_{n-1}, 1], \boldsymbol{\pi}_{LL} \rangle \tag{4.6}$$

$$c = \langle [\alpha_0, \ldots, \alpha_{n-1}, 0], \boldsymbol{\pi}_{HL} \rangle \tag{4.7}$$

We expect the ANN model to be able to learn the commonalities and differences from the information gathered from the two simulation types; in practice, we expect it to learn the correlation – if any – between type 0 and type 1 metrics.

4.3 Proposed ensemble model and experimental results

STMicroelectronis STHORM was chosen as reference platform because the complexity of this multi-cluster architecture can be modeled at different abstraction levels: *posix-posix* for $\mathcal{M}_{HL}$ and *posix-xp70* for $\mathcal{M}_{LL}$, see Section 2.4. At the same time, the customization space for applications targeted to this type of device is too large to think about profiling all possible configurations on a cycle-accurate simulator. Our approach being application-specific, we selected the OpenCL Stereo-Matching application presented in Chapter 3 and we targeted it to STHORM.

The customization is enabled by a set of application parameters that can be of two types (see last two rows in Table 4.1):

- *application specific* parameters, which affect both the Quality of Service (QoS) provided by the application and the computational load;
- *platform resource* parameters, which impact on the OpenCL runtime (kernel scheduling and memory access patterns), but not on the quality of application results.

As seen in Chapter 3, the possibility to change the *platform resource* parameters allows for improved application portability and optimization for the specific target device. In this experiment, we are mainly interested in the metric *cl-cycles*, returned by both *posix-posix* and *posix-xp70* simulation models. This metric represents the number of cycles for execution of the OpenCL kernels on STHORM and depends on both application specific and platform parameters.

Table 4.1: *Design space exploration for OpenCL Stereo-Matching application.*

SIZE OF DESIGN SPACE	486
TIME (H) FOR COMPLETE LL SIMULATION	603.15
TIME (MIN) FOR COMPLETE HL SIMULATION	44.48
PLATFORM RESOURCE PARAMETERS	4
APPLICATION-SPECIFIC PARAMETERS	2

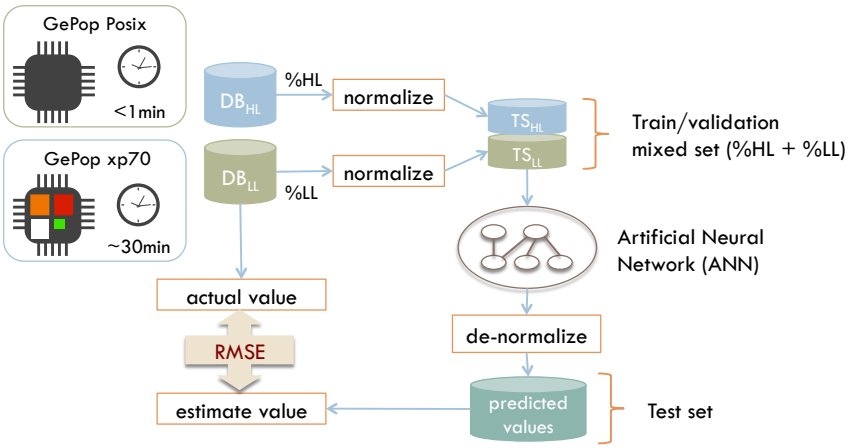


Figure 4.3: *Experimental setup for tuning the neural network configuration.*

4.3.1 Preliminary correlation analysis

Table 4.2 shows the selected range of values for the configuration parameters of the neural network model. The LL% and HL% values represent the percentages of design space that are explored using $\mathcal{M}_{LL}$ and $\mathcal{M}_{HL}$, respectively, to estimate the clock cycles (*cl-cycles*). As shown in Figure 4.3, the set of mixed configurations (in terms of LL% accurate and HL% approximate samples) is used for training and validation of the ANN model. Then, by querying the ANN model, it is possible to predict the clock cycles (*cl-cycles*) for those configurations that were not simulated on $\mathcal{M}_{LL}$ during the first phase. The prediction error for a specific ANN configuration is calculated as the relative Root Mean Square (RMS) error with respect to the solutions obtained by using only $\mathcal{M}_{LL}$.

Figure 4.4 shows a correlation analysis between the parameters presented in Table 4.2 and the simulation time and RMS error associated with the proposed ensemble model. The area as well as the color intensity of the circles in the plot represent the absolute value of correlation, while the color (blue or red) indicates whether the correlation is, respectively, positive or negative (indicated also by the '+' or '–' sign in each cell). The negative correlation between "Error" (RMS), on the one hand, and both "HL%" and "LL%", on

Table 4.2: *Parameters for neural network model tuning of OpenCL Stereo-Matching application.*

# LAYERS	# NEURONS	HL%	LL%
1-5	1-5	0, 50, 100	2, 4, 6, 8, 10

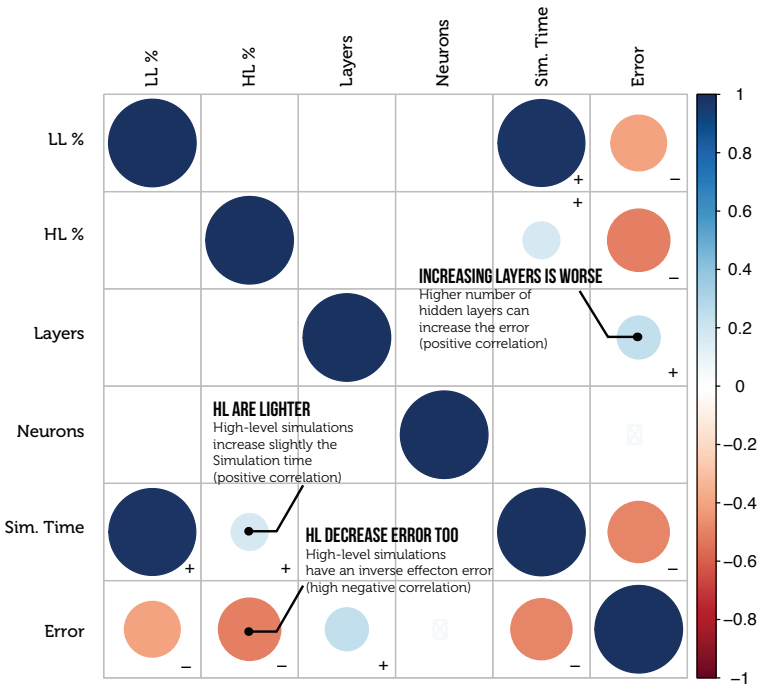
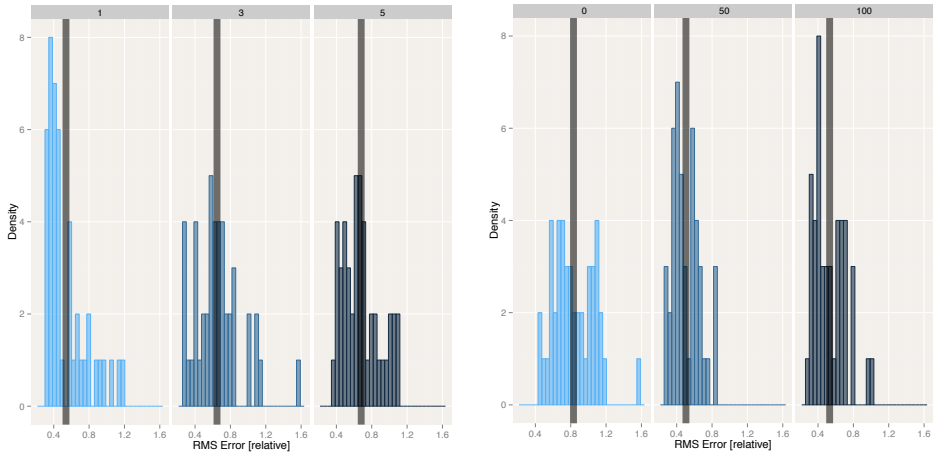


Figure 4.4: *Correlation analysis applied to the neural network model for the OpenCL Stereo-Matching application targeted to STHORM. The metric "Sim. Time" refers to the total simulation time required to build the mixed training set for a given model configuration.*

the other hand, confirms that it is possible to improve prediction accuracy by increasing the size of the training set. This is verified both for $\mathcal{M}_{HL}$ and $\mathcal{M}_{LL}$ training samples, which accounts for the adoption of an ensemble model. Indeed, while reducing the RMS prediction error, the utilization of $\mathcal{M}_{HL}$ simulations has a very low impact on the simulation time, which is confirmed by the low correlation between "HL%" and "Sim. Time". Since the cycle-approximate platform is much faster than the cycle-accurate one (13x according to our measurements, see first and second row in Table 4.1), the time contribution from adding $\mathcal{M}_{HL}$ samples is negligible: thus, the total simulation time to build a mixed training set is almost proportional to the number of simulations on $\mathcal{M}_{LL}$.

4.3.2 Accuracy analysis of the ensemble model

The histogram in Figure 4.5a shows the number of Artificial Neural Network (ANN) configurations for different levels of RMS prediction error. The



(a) *Distribution grouped by the number of hidden layers. Its shows the impact of neural network depth (1, 3 and 5 hidden layers) on the RMS prediction error.*

(b) *Distribution grouped by the percentage of high-level training samples. Its shows the impact of adding high-level training samples (0%, 50% and 100% of the design spaces) to the ensemble model.*

Figure 4.5: *Density distribution of neural network configurations with respect to RMS prediction error.*

configurations are assigned to one of the three histograms according to the number of layers in the ANN. In this plot, the highest density distribution for low levels of RMS error corresponds to ANN configurations with only one hidden layer. In general, the best configuration we have consistently seen in our experiments is a ANN with 1 hidden layer of 5 neurons.

Figure 4.5b shows the distribution of ANN configurations, in relation to the percentage of high-level samples used for training the ensemble model. The configurations with the same number of high-level training samples (either 0%, 50% or 100%) are grouped together in the same vertical window. The vertical gray bar indicates the average RMS error in each group: we observe that by passing from a purely cycle-accurate model (first window from left) to an ensemble model with 50% high-level training samples, the average RMS error on the prediction of metric *cl-cycles* decreases from 0.8 down to 0.5. At the same time, as seen in Figure 4.4 and confirmed by the measurements in Table 4.1, the delay due to additional high-level simulations is negligible compared to the time required for cycle-accurate simulations.

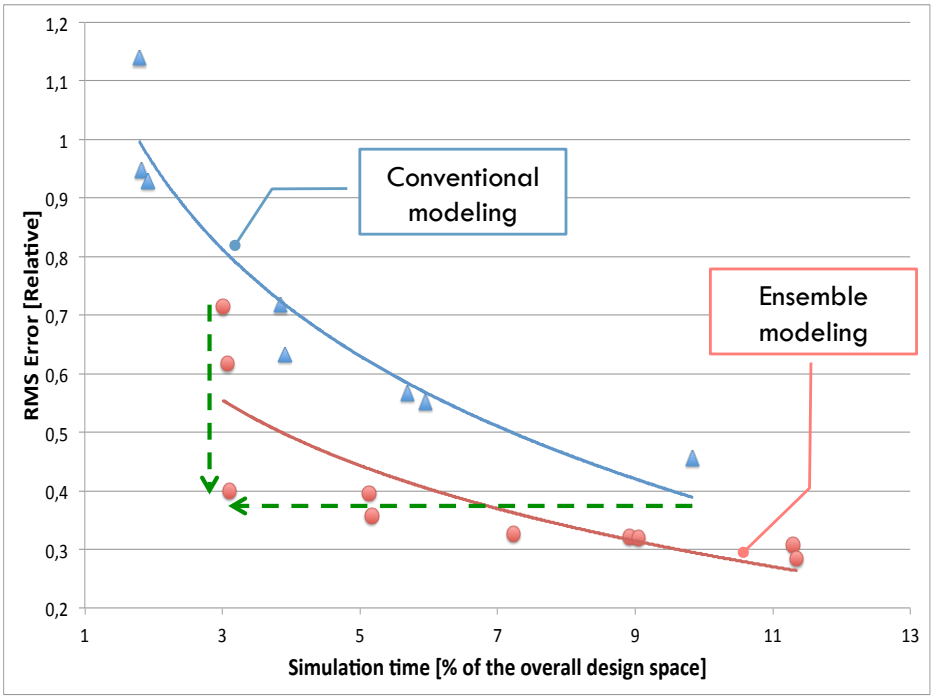


Figure 4.6: *Analysis of efficiency between traditional neural network modeling, as in [92], and the proposed ensemble modeling technique for Design Space Exploration.*

4.3.3 Analysis of variance of the results

We applied the non-parametric Kruskal-Wallis analysis of variance (ANOVA) to assess whether the introduction of $\mathcal{M}_{HL}$ training samples is statistically significant for improvement of the model prediction accuracy. The *p-value* in Table 4.3 indicates the probability that the observed difference in means is due to chance, rather than it being a systematic effect. Thus, because of the low *p-value*, we deduce that the distributions observed for the selected application must be regarded as significant.

Table 4.3: *Kruskal-Wallis analysis for significance of LL% training samples on the Root Mean Square (RMS) of model predictions.*

CHI-SQUARED	DEGREE of FREEDOM	P-VALUE
47.686	2	4.4e-11

4.4 Conclusions

Traditionally, high-level models are used to speed up the simulation process at the expense of profiling accuracy. In this chapter, ensemble models based on neural networks and trained with both low-level and high-level simulations (mixed training set) show better accuracy with respect to conventional models, for which the training set consists of only low-level samples. This result can be achieved because, given a time window as shown in Figure 4.6, ensemble neural network models enable to better exploit the time window in terms of high-level (fast) and low-level (slow) simulations.

The accuracy improvement reported by using ensemble neural network models is up to 30% for the OpenCL Stereo-Matching application running on the STHORM platform. Alternatively, the same level of accuracy could be achieved by replacing part of the cycle-accurate training set with application configurations profiled on a fast simulation platform, with a one order of magnitude speed-up of the design exploration.

CHAPTER 5

Task Mapping under Heterogeneous Platform Constraints

The convergence to *globally heterogeneous locally homogeneous* computational parallelism, in both the embedded and High Performance Computing (HPC) domains, requires common design methodologies to better exploit such complex platforms while easing application porting. The shift towards heterogeneous parallel computing can be observed, for example, in the OpenCV library [24] which includes, starting from version 2.4.3, an OpenCL back-end. This means that augmented reality applications that use OpenCV can now access any type of OpenCL-capable accelerator through standard APIs, on hand-hold devices with embedded GPUs as well as on desktop computers. Indeed, while CUDA limits application portability to Nvidia GPUs, the OpenCL back-end of OpenCV already supports GPUs from different vendors, as well as multi-core processors, and it is planned for a wider range of devices (e.g. OpenCL-enabled FPGAs) in the near future.

However, when targeting an OpenCL application to a heterogeneous parallel platform, task mapping has to cope with the platform constraints. To address this problem, this chapter presents a design methodology [1] that extends the approach presented in Chapter 3, to exploit the concurrency in

the application task graph for efficient mapping on heterogeneous parallel platforms. In the tuning phase, this technique generalizes the previous analysis of inter-dependent parameters [48], by exploiting a *constraint solver* to efficiently identify an initial set of feasible task configurations that are compliant with the platform constraints. In the mapping phase, it improves inter-task parallelism while accounting for the overhead of host-to-device and device-to-host memory transfers – overheads implied by multiple OpenCL contexts for different device vendors, as discussed in Section 2.2.

The proposed design flow is validated on the OpenCL Stereo-Matching application, targeting the two CPU/GPU heterogeneous parallel platforms described in Section 2.4.

5.1 Related Work

The auto-tuning methodology presented in this chapter differs from the one of Chapter 3 in two main aspects. The first difference is that the application in Chapter 3 was targeted to a single OpenCL device, while in this chapter platforms with multiple, heterogeneous accelerators are considered. Thus, this chapter also addresses *inter-task* parallelism, as discussed in Section 2.2, and investigates the optimality of a mapping taking into account the overhead of data transfers between different OpenCL contexts. As in Chapter 3, the methodology presented here separates the application parametric design from the optimization framework; the difference is that the Design Space Exploration (DSE) heuristic presented in Chapter 3 was application and platform specific, while this chapter presents a more generic exploration strategy for both parameter customization and task mapping.

As in Chapter 3, a design-time phase allows for offline optimization and customization of applications targeted to heterogeneous parallel platforms. Optimization of OpenCL streaming applications on heterogeneous platforms has recently been addressed by a number of works based on Domain Specific Languages (DSLs), such as Halide [94], KernelGenius [77] and HIPAcc [84]. These works present different DSLs for the same application domain, namely image processing, and focus on a specific class of kernels, the stencil operation. Stencil kernels are relatively simple but can be fused to better exploit data locality, so the works [94, 77, 84] aim at optimizing complex stencil pipelines used for image processing. The generated code outperforms hand-optimized programs, because the description in a DSL enables fine-grained customization by means of source-to-source code transformations.

While the works in [77] and [84] use the term heterogeneity to refer to different target platforms, Halide [94] provides a concrete example of code

that can efficiently exploit a heterogeneous platform, running concurrently on multiple different accelerators (e.g. CPU-GPU). In this sense, the problem addressed by Halide is the same of this thesis, but there are some major differences: on the one hand, their approach is limited to a specific type of kernel operations and cannot address *kernel heterogeneity* in the application task graph; on the other hand, it requires a DSL description of the algorithm, so it does not enable porting of existing OpenCL application code to next-generation platforms.

The type of customization considered in this chapter is *coarse-grain*, since the application task graph is an input to our methodology and the tasks are atomic. Other works in literature follow a similar *coarse-grain* approach, such as SOCL [52] and OmpSs [39], but they use dynamic task mapping and scheduling. SOCL provides a unified OpenCL platform, built on top of the StarPU runtime system [16], to enable access to heterogeneous accelerators from different vendor platforms as if they belonged to the same platform. Thus, it automatically handles memory transfers between different OpenCL contexts, when needed, and allows to instantiate a single command-queue attached to multiple devices (not supported by the OpenCL standard), dynamically dispatching the application tasks depending on run-time availability. The OmpSs framework not only allows to use heterogeneous accelerators within the same application, but also to mix different programming APIs (OpenMP, OpenCL and CUDA) while automatically handling data dependencies in the task graph.

With respect to [52], the proposed approach is suitable to embedded applications since i) it improves performance predictability by means of offline workload characterization and ii) it avoids the overhead of run-time scheduling. These considerations also apply to [39], moreover OmpSs does not provide a unified programming API, relying instead on different APIs (OpenMP, OpenCL and CUDA) for different target devices. Moreover, both of them lack auto-tuning of tasks for the target platform, thus requiring by-hand code customization.

Several works address the problem of task scheduling on heterogeneous platforms targeting only Nvidia GPUs and focusing primarily on load balancing [85, 113]. By targeting a homogeneous execution context, on computing devices from the same vendor and with the same programming API (CUDA), these techniques can easily move tasks from one computing device to another and exploit specific features of the target architecture to hide memory transfers [113]. Sometimes, this hypothesis is used to completely ignore the cost of memory transfers in the timing model for GPU workload [75]. On the contrary, the framework in [85] does not support the execution of an

application task graph on multiple devices in order not to deal with memory transfers, whereas our approach does. Moreover, these approaches are more suitable to the HPC domain, in which clusters of GPUs can host concurrent execution of multiple applications, while the technique presented in this chapter addresses embedded platforms, where the composition of the run-time workload is known at design time and therefore the overhead of dynamic scheduling can be reduced or even completely avoided.

Elastic computing [117] is another interesting approach, based on the complete separation of functionality from implementation. It uses multiple implementations of the same functionality, not limited to a specific application domain, to find the optimal mapping for a target heterogeneous platform. As in our approach, all alternative implementations of a given functionality must be provided to the optimization framework either by the application developer or in the form of library. However, in [117] the implementations can use different languages, depending on the target device. Thus, the main difference with respect to our methodology is that the work in [117] does not start from a cross-platform programming paradigm such as OpenCL. On the one hand, this limits application portability to the set of available kernel implementations; on the other hand, kernel customization for the target platform is not needed.

5.2 Proposed Methodology

The methodology for the customization of OpenCL applications consists of two main phases: tuning and mapping, as illustrated in Figure 5.1. It applies to multi-task applications expressing task-level parallelism in the form of a *large grain data flow* (LGDF) graph [74, 13]. Each task is supposed to have a *parametric design*, i.e. the task behavior can be customized by tuning source-level parameters. For an OpenCL application, the term *task* refers to the execution of an OpenCL kernel on a platform device, by means of the *clEnqueueNDRangeKernel* API [67]. Therefore, a task includes an iteration space over an *NDRange*, to parallelize the kernel execution on separate data blocks. For OpenCL kernels, the source-level parameters include constants used to allocate buffers in local memory and iterate over the input data, as well as size of the global and local grids which define the *NDRange*.

Figure 5.1: *Proposed Design Space Exploration (DSE) flow for tuning and mapping an OpenCL application to a heterogeneous platform.*

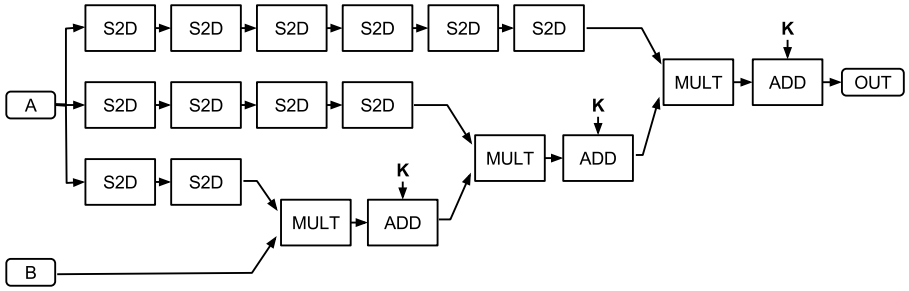


Figure 5.2: *Synthetic OpenCL task graph used to describe the methodology.*

The choice of parameters has been inspired by the work in [104], which provides an overview of the OpenCL performance pitfalls in porting OpenCL applications from GPU to CPU devices. Since the transformations proposed in [104] do not alter the parallelization or the structure of the original program, they could be implemented as a parameterization of the OpenCL code. In this case, code specialization becomes a generic form of parameter tuning, providing good opportunities for automation and improving inter-platform OpenCL performance portability.

The proposed methodology consists of two main phases, as illustrated in Figure 5.1. In Phase 1, each task is first considered independently to apply task-level optimization. We use an analytical technique to identify the subset of source-level parameters that satisfy the constraints of each available platform device (Phase 1, Block A). Then the feasible configurations pass through a *tuning* phase that removes Pareto-dominated solutions (Phase 1, Block B). At the end of Phase 1, each task is associated with its best parameter configurations for each device although it has not yet been assigned to any specific device. At this point, the parametric configurations of the same task on different devices may be significantly different due to the device constraints. In Phase 2, Block C implements the mapping problem, by means of a solver based on constraint programming [86]. The input to the solver is the application task graph and the pruned set of task configurations coming from the previous phase. Phase 2 considers, as its optimization objective, the application throughput, taking into account the overhead of host-to-device and device-to-host memory transfers.

To better illustrate the steps of the proposed methodology, we introduce a synthetic OpenCL application. The final goal is to map this application to different platforms, by exploiting inter-task parallelism. The application Large Grain Data Flow (LGDF) is shown in Figure 5.2. It consists of OpenCL kernels with different memory access patterns and different memory

Table 5.1: *Source-level parameters for matrix multiplication kernel.*

Parameter	Min	Max	Step
num_tiles_x	1	Matrix C width (2048)	1
num_tiles_y	1	Matrix C height (1024)	1
num_tiles_xy	1	Matrix A width (512)	1
wg_size_x	1	Device max wg_size_x	1
wg_size_y	1	Device max wg_size_y	1
block_size_x	1	512	1
block_size_y	1	512	1

requirements, as well as multiple instances of the same kernel processing different data. The input data stream is represented by blocks A and B. Data is processed by multiple tasks in a *pipelined* modality, whereas tasks are instances of a matrix multiplication (MULT), a matrix add (ADD, which adds a constant matrix K) and a matrix stencil (S2D) operator.

Considering the execution context, there are different platforms on which we might want to run the above application. Each OpenCL platform may have specific features that could affect the overall partitioning onto multiple, heterogeneous devices. To address this problem, Phase 1 of the methodology (Section 5.2.1) focuses on each kernel of the application to prune its source-level design space, then Phase 2 (Section 5.2.2) considers the application task graph as a whole.

5.2.1 DSE Phase 1 – Task tuning

This section describes how task tuning applies to one of the tasks of the application shown in Figure 5.2: the *matrix multiplication* (MULT). We consider an extended version of the original matrix multiplication sample available in the Altera OpenCL SDK[1]. OpenCL, indeed, allows to run the very same code provided by Altera for an FPGA target on a CPU or GPU, while the contribution of this task tuning phase consists of customizing source-level parameters for the target platform.

The set of tunable parameters that we consider for the matrix multiplication kernel is shown in Table 5.1. The original implementation already exploits *tiling* – by means of buffers in local memory – in order to reduce the global memory access bandwidth. However, to broaden the scope of this investigation, the application has been modified to let the *workgroup size* be set independently from the *tile size*. Besides, the workgroup shape has

[1] http://www.altera.com/products/software/opencl/opencl-index.html

also been changed from a squared to a rectangular one – with dimensions `wg_size_x` and `wg_size_y`. Finally, two additional parameters allow to increase the work for a single work-item, by making it process a block of dimensions `block_size_x` × `block_size_y`.

This modification adds more opportunities for the mapping stage but, since the tile size is forced to be a square, it provides an additional set of constraints to be met:

- `tile_size_x` = `wg_size_x` × `block_size_x`
- `tile_size_y` = `wg_size_y` × `block_size_y`
- `tile_size_x` == `tile_size_y`
- The size of the tile must meet the limits of the local memory size of the device.
- The size of input matrices, in each dimension, must be a multiple of the tile size.

By solving at design time the last constraint, the methodology allows to simplify the control logic for checking the boundary conditions within the kernels themselves. This is important for GPU architectures, whose performance is affected by *thread divergence*. As described in the next paragraph, sizing of the parameters shown in Table 5.1 is done through a constraint problem formulation.

Constraint problem formulation. In constraint programming (CP), a problem is formulated using constraints to define relations between variables, then a solver generates solutions that satisfy such constraints [96]. This makes CP a form of declarative programming, since the constraints do not specify a sequence of steps to find a solution, but rather the properties of a solution to be found.

The kernel parameterization problem is formulated as a CP problem, using the formalism provided by the Minizinc constraint solver [86]. We define a modular composition of variables and constraints by dividing them in two sets:

- OpenCL platform variables and constraints on the dimension of the workgroups and local memory size (Listing 5.1).
- Application-specific variables and constraints on the properties of the tiles used by the matrix multiplication kernel (Listing 5.2).

The problem formulation consists of a sharp separation between the definition of generic constraint rules and the specialization of these on a

Listing 5.1: *OpenCL platform constraints*

```
% Platform parameter declaration
int: max_wg_size;
int: max_wg_size_x;
int: max_wg_size_y;
int: max_wg_size_z;
int: local_mem_size;

% Platform decision variables
var 1 .. max_wg_size_x : wg_size_x;
var 1 .. max_wg_size_y : wg_size_y;
var 1 .. max_wg_size_z : wg_size_z;
var int: local_mem_usage;

% Platform constraints
constraint (wg_size_x <= max_wg_size_x);
constraint (wg_size_y <= max_wg_size_y);
constraint (wg_size_z <= max_wg_size_z);
constraint (wg_size_x*wg_size_y*wg_size_z <= max_wg_size);
constraint (local_mem_usage <= local_mem_size);
```

specific platform. This represents one of the advantages of using CP. For example, considering the platform constraints in Listing 5.1, there could be different values for the `max_wg_size` bound that are dependent on the device actually used. Thus, in order to keep the problem formulation generic, the parameters are initialized in a separate file.

The application-specific constraints represent an extension of the more general OpenCL platform constraints. Thus, another advantage of using CP is that the problem formulation can be easily extended. Moreover, the problem formulation is independent from the search strategy used by the solver, which allows to evaluate – or eventually to develop in parallel – an optimized search strategy. Finally, when dealing with minimization or maximization goals (like the task mapping problem in Section 5.2.2), the solution is potentially optimal.

The output of the constraint solver (Block A in Figure 5.1) is the feasible solution set for matrix multiplication, which is a subset of the original design space shown in Table 5.1. The main advantage introduced by this technique is a drastic reduction of the exploration space and, more important, of the exploration time, while not disregarding any feasible – and therefore possibly optimal – solution.

Listing 5.2: *Application-specific constraints*

```
% Parameter declaration
int: size_x;
int: size_y;
int: size_xy;
int: max_block_size_x;
int: max_block_size_y;

% Decision variables
var 1 .. size_x : num_tiles_x;
var 1 .. size_y : num_tiles_y;
var 1 .. size_xy: num_tiles_xy;
var 1 .. max_block_size_x : block_size_x;
var 1 .. max_block_size_y : block_size_y;
var int: tile_size_x = ( wg_size_x * block_size_x );
var int: tile_size_y = ( wg_size_y * block_size_y );

% Variable assignment
wg_size_z = 1;
local_mem_usage = (tile_size_x * tile_size_y * 4 * 2);

% Application constraints
constraint (tile_size_x == tile_size_y );
constraint (size_x  == (tile_size_x * num_tiles_x ));
constraint (size_y  == (tile_size_y * num_tiles_y ));
constraint (size_xy == (tile_size_x * num_tiles_xy));
```

Task tuning. The constraint programming solution is used as a starting point for the next sub-phase — the *task tuning* (Block B in Figure 5.1). Here, for each task in the LGDF, we search over the feasible set in the application design space to find the optimal configuration with respect to some design objective. In our flow, the optimization objective consists of minimizing the kernel execution time, by averaging the profiled metrics over a dataset representative for the target application in order to account for data-dependent variability.

To automate this process we use MOST, the profile-based DSE tool for multi-objective optimization presented in Section 2.5. Figure 5.3 shows the tools used in the proposed design flow: the MiniZinc solver, implementing the analytical technique described in the previous section, allows for fast pruning of those configurations of an OpenCL kernel unfeasible with respect to device constraints; then, MOST adopts heuristics [48, 92] to navigate and prune the feasible solution set, by profiling the kernel on the target device.

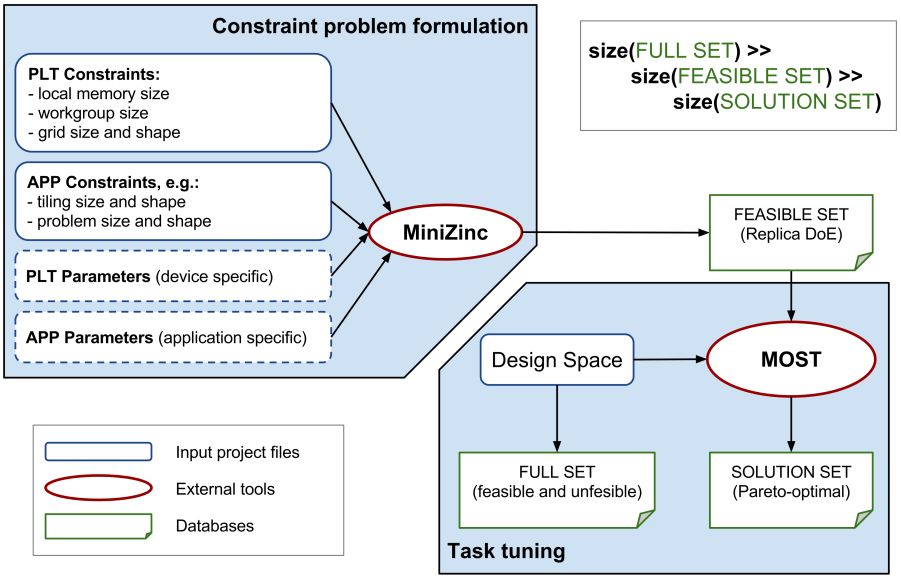


Figure 5.3: *The proposed DSE framework implementing Phase 1.*

5.2.2 DSE Phase 2 – Task mapping

The input of Phase 2 (see Figure 5.1) is a set of optimal configurations for each task – one for each platform device – derived in the previous phase. Each configuration is characterized by its own *kernel execution time* (EXE), as measured by the task tuning in Phase 1. However, in order to also take into account inter-task communication when tasks are mapped to different devices, additional information is needed: i) the time required to copy input data from host to device buffers (H2D), and ii) the time to retrieve the kernel output from the device buffers (D2H). To measure the communication overhead, the application is structured as a network of *components* (see *ocl_task* in Figure 5.4), where each component is an instrumented version of the original OpenCL kernels. This instrumentation allows to measure H2D and D2H average times, which are later used by the constraint solver to find the optimal task mapping, as described below.

Task graph mapping. To expose concurrency, the mapping process is based on well-known pipelining principles [74]; it corresponds to identifying *feed-forward cut-sets* in the LGDF and placing *buffers* in-between. Each cut-set of the task graph becomes thus a pipeline stage. The optimal mapping – with respect to the throughput – minimizes the critical path of the decomposed graph, i.e. the slowest pipeline stage for kernel execution time and buffering overhead (H2D, D2H).

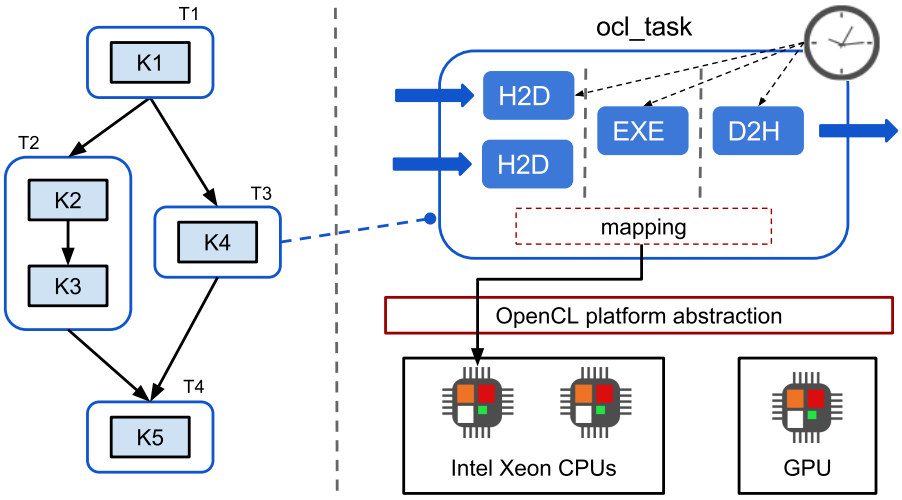


Figure 5.4: *Skeleton component ocl_task used to wrap OpenCL kernels and profile execution (*EXE*) and memory transfer times (*H2D*,* D2H*).*

Figure 5.5 shows a minimal OpenCL task graph mapped to a heterogeneous platform with different OpenCL contexts, since the computing devices belong to different vendors (e.g. Nvidia for the GPU device and Intel for the CPU one). When two tasks, connected by an edge in the task graph, are mapped to computing devices belonging to different OpenCL contexts (e.g. tasks A-D or C-E), the data dependency requires a copy of data buffers from one device to another, which results in a D2H and H2D operation. Different accelerators – and even the same accelerator in different PCI slots – can experience significantly different communication times. For example, the memory transfers from GPU on our target platform PLT1 (see Section 5.3) are 46% slower than from the CPU. These transfer times are taken into account in the analysis and optimization.

The mapping problem is solved as a constraint optimization problem. The program formulation shown in Listing 5.3 is applicable to any *acyclic* OpenCL task graph. It defines a set of constraints in terms of precedence rules and memory transfers that are applied only when required by a specific mapping. In fact, H2D and D2H overhead times are taken into account only when two tasks connected through an edge – data dependency – are scheduled in different OpenCL contexts; otherwise, they are ignored, since data are already in the device memory. In the final result, multiple tasks could be executed on the same device. This corresponds to instantiating a task that wraps several kernels (see Figure 5.4), ensuring that internal

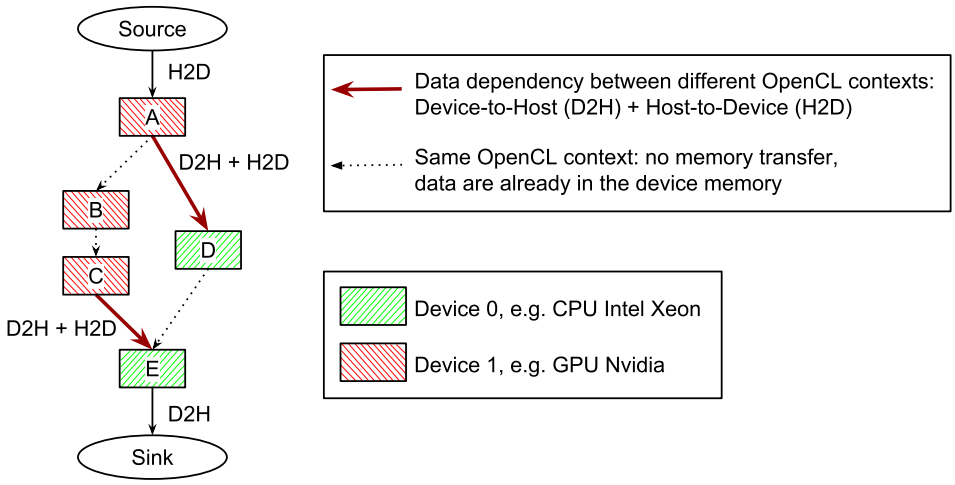


Figure 5.5: *Task mapping example.*

scheduling of each kernel meets the dependencies expressed in the original task graph.

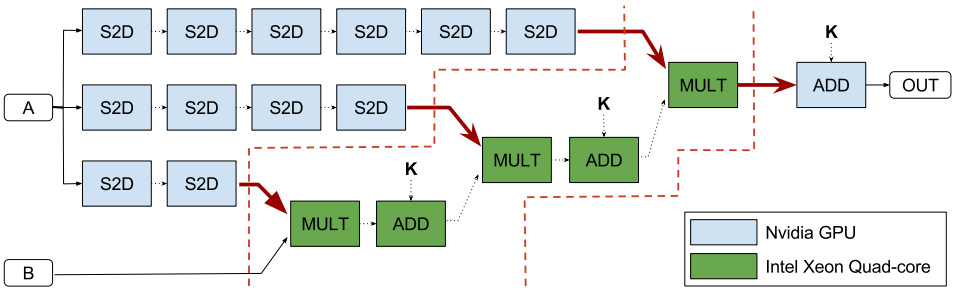
Figure 5.6 shows the application task graph associated with the mapping solutions found for the two target platforms. Kernels with the same color are mapped to the same computing device while the dashed lines show the *feed-forward cut-sets* in the LGDF, which define different pipeline stages.

5.3 Experimental Setup

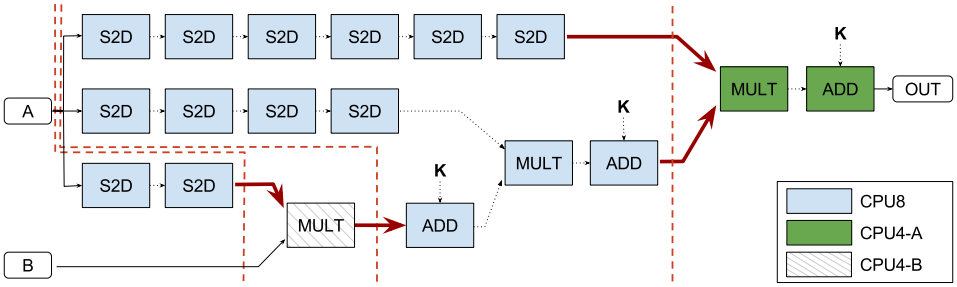
The presented flow has been automated to a large extent: the task tuning phase is fully automated and the constraint solver, both for pruning the unfeasible solutions and task mapping, directly provides the solution. The only information required as input is the list of kernel constraints and customizable parameters. Nevertheless, the application host code is not generated automatically, but we rely on template code to realize the final mapping of OpenCL kernels to platform devices. This enabled the validation of the proposed methodology on a real world use case: the OpenCL Stereo-Matching application described in Section 2.3, targeted to two commercial heterogeneous parallel platforms. Stereo-Matching (SM) is a streaming application, since it processes an input stream of stereo frames, and therefore the *throughput* of the application (frame-rate) can benefit from the software pipelining technique proposed in Section 5.2.2. The SM behavior can be tuned by setting some specific parameters, to trade-off the *throughput* (frame-rate) and the *Quality-of-Service* (QoS), which measures the accuracy of the result.

Listing 5.3: *Problem formulation for application task mapping*

```
% TaskPrec[j] contains the predessors of task j
% TaskNext[j] contains the successors of task j
% TaskType[j] is the type of task j (instance)

% Total host-to-device memory transfer time
% In this implementation,
% we consider T_H2D = T_D2H for a device buffer
constraint forall (i in DEVICES) (
  sum_time_h2d[i] = sum (j in TASKSET) (
    sum (p in TASKSET) (
      bool2int( (p in TaskPrec[j]) and (mapping[j]==i)
          and (mapping[p]!=i) ) * T_MEM[i,TaskType[p]]
    )
  )
);

% Total kernel execution time on each device
constraint forall (i in DEVICES) (
  sum_time_exe[i] = sum (j in TASKSET) (
    bool2int( mapping[j]==i ) * T_EXE[i,TaskType[j]]
  )
);

% Total device-to-host memory transfer time
constraint forall (i in DEVICES) (
  sum_time_d2h[i] = sum (j in TASKSET) (
    sum (n in TASKSET) (
      bool2int( (n in TaskNext[j]) and (mapping[j]==i)
          and (mapping[n]!=i) ) * T_MEM[i,TaskType[n]]
    )
  )
);

% Optimize pipeline stage
constraint forall (i in DEVICES) (
  (sum_time_h2d[i] + sum_time_exe[i]
         + sum_time_d2h[i]) <= StagePeriod
);

% Minimization solver
solve minimize StagePeriod;
```



(a) *Mapping for* PLT1



(b) *Mapping for* PLT2

Figure 5.6: *Optimal mappings of the synthetic task graph to two heterogeneous platforms.*

Our analysis is focused on the parameter which controls the step between consecutive disparity hypotheses, the `hypo_step`. Since the algorithm mainly consists of minimizing the matching-cost while iterating over the disparity range, the tuning of `hypo_step` directly affects the performance in terms of *throughput* and *QoS*, mainly the *disparity error* (the average error in the pixel disparity computation).

Figure 5.7 shows the OpenCL kernels invoked by SM. Some of them are instantiated several times (*WinBuild*, *ConvertInty*, *AggVer*) so that each instance can be considered a different task. The gray box represents one step of the iterative cost aggregation, which is repeated for each disparity hypothesis. The OpenCL kernels within this box have been modeled as a single task, in this experiment, for two reasons: 1) the constraint program formulation for solving the mapping problem currently supports only tasks with a single output and 2) these kernels represent steps of a cost aggregation super-task which would not benefit from execution on different devices.

In this experiment we use a stream of input frames with resolution 384×288 and we fix the maximum disparity hypothesis to 24: thus, given an anchor pixel in the left stereo frame, its corresponding pixel in the right

Table 5.2: *Experimental setup of OpenCL platforms.*

Platform	OpenCL devices
PLT1	One 4-core CPU device, one discrete GPU
PLT2	One 8-core CPU device, two 4-core CPU devices

frame should have a maximum horizontal offset of 24 pixels. We consider 4 values of `hypo_step`, from 1 to 4, corresponding to four different ranges of disparity hypotheses. Figure 5.8 shows the task graph flattened with `hypo_step` set to 1, i.e. the largest task graph, which results in 24 cost-matching tasks.

In the rest of this chapter, PLT1 and PLT2 refer to the two CPU/GPU platforms described in Section 2.4. The configuration of the two platforms is summarized in Table 5.2. The CPU nodes on PLT2 are exposed by the OpenCL runtime as a homogeneous multi-core CPU device. In order to emulate a heterogeneous platform, we used the device fission API [67] to partition the 16 cores into 3 CPU sub-devices: one sub-device with 8 cores and two with 4 cores each. Thus, both target platforms provide heterogeneity: architectural heterogeneity in PLT1 and computational heterogeneity in PLT2. Although they are general purpose, we expect comparable computational capability and architectural complexity to be soon available on high-end embedded platforms.

5.4 Experimental Results

The design flow is applied to task optimization and mapping of the OpenCL Stereo-Matching (SM) task graph, presented in the previous section.

DSE Phase 1 – Task tuning. The constraint solver is very efficient in finding the feasible task configurations: the feasible sets of all SM kernels on all target devices (Table 5.3) were found in $11s$, using one CPU core on PLT1. The full design space contains 2^{24} points for the GPU device on PLT1, while it is even larger on the CPU devices since they support larger workgroups. For this experimental setup, the feasible set is always smaller than 0.1% of the full design space for all kernels and target devices. These cut-sets represent the result of Phase 1, Block A: they are very precise and capture all the candidate optimal solutions.

Figure 5.9 shows the distribution of feasible design configurations for kernel *WinBuild*, with respect to parameters WG_SIZE_X and WG_SIZE_Y. The set of points is different for the two target OpenCL devices on PLT1, mainly because the platform constraint on the local memory is smaller on

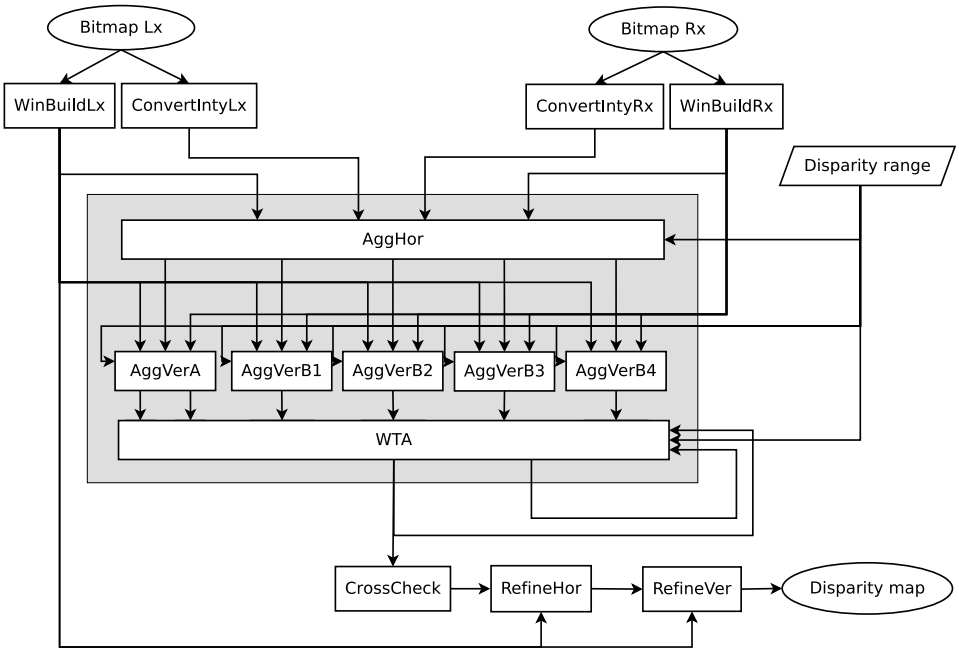


Figure 5.7: *Task graph of Stereo-Matching OpenCL kernels.*

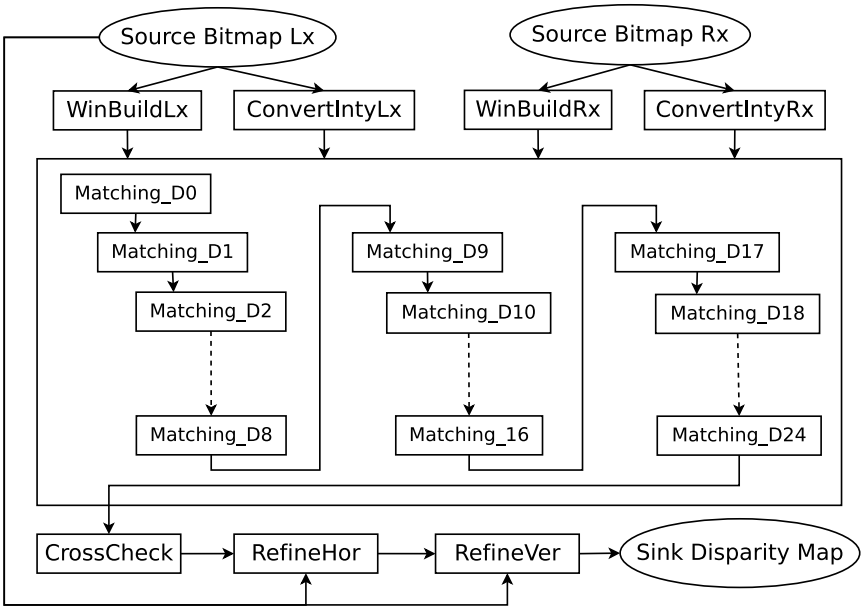
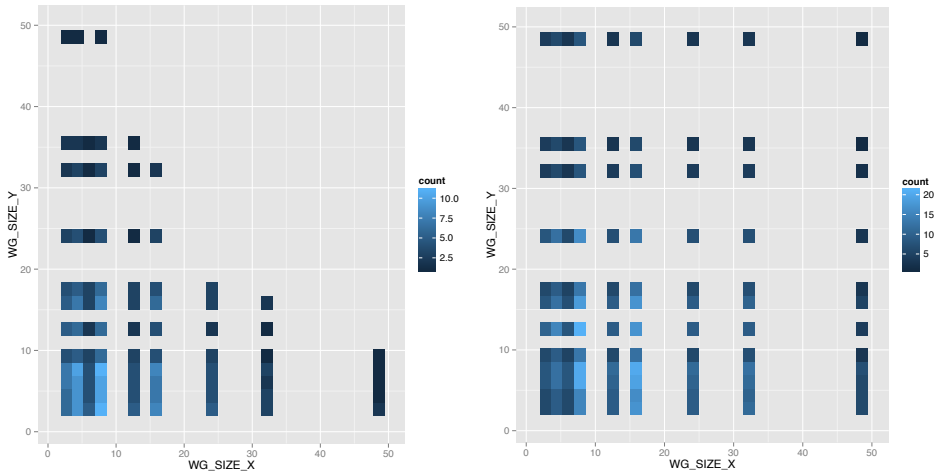


Figure 5.8: *Flattened task graph of Stereo-Matching OpenCL kernels, with parameter hypo_step set to 1 (24 disparity hypotheses).*



(a) *WinBuild* - PLT1, *GPU*

(b) *WinBuild* - PLT1, *CPU*

Figure 5.9: *Distribution of design points in the feasible solution set for kernel WinBuild, on* PLT1 *OpenCL devices, with respect to parameters* WG_SIZE_X *and* WG_SIZE_Y.

GPU than on CPU ($16KB$ vs. $32KB$). If there were no constraints, the feasible set would have been the same for the two target devices and the points would have been homogeneously distributed. On the contrary, only a small area is covered with feasible solutions, moreover their distribution is not homogeneous. This shows a first interesting result: by adopting the *ad hoc* filtering technique based on constraint programming, the design space to be explored was reduced by three orders of magnitude.

The optimization phase (Phase 1, Block B) is driven by the profile-based DSE tool, presented in Section 2.5, which was configured for this experiment to select up to 50% points of the feasible set and to minimize the average kernel execution time. The percentage of points can be tuned to control the trade-off between accuracy of the solution and exploration time [108]. Since we set a single design objective for the optimization of OpenCL kernels, the result is one configuration of parameters, specific for the target device, which is optimal with respect to the average throughput.

Table 5.4 shows the tile size for each kernel, which results from the combination of optimal parameters. We can observe that the tile size of *ConvertInty*, *WTA* and *CrossCheck* is in average higher than for other kernels, because these kernels do not allocate buffers in local memory and therefore such constraint is not considered. On the contrary, the tile size of the other kernels – those kernels which use shared memory to exploit data reuse – on the GPU device is always smaller than on the CPU devices (except

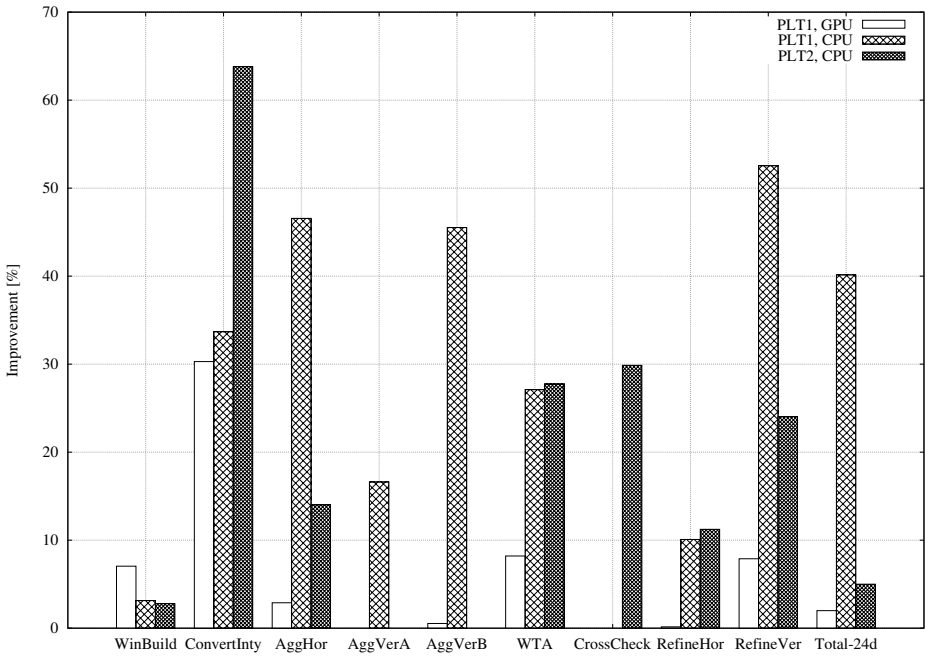


Figure 5.10: *Performance improvement of the Stereo-Matching kernels, after applying the task tuning phase, vs. the original implementation.*

for one case, but this might be due to the fact that the provided solution is approximate). Also in this case, the difference in optimal tile size for CPU and GPU devices mainly depends on the platform constraint of local memory size, which is smaller on GPU than on CPU.

In the original CUDA implementation of the Stereo-Matching application [126], the number of threads per block (equivalent to the workgroup size in OpenCL) is 16x16. This design choice was based on the maximum number of CUDA threads per block and on code optimizations for data fetching into local memory [126]. Thus, we consider a reference configuration of the Stereo-Matching kernels with 16x16 workgroup size and 1x1 block size. Figure 5.10 shows that the tuning phase provides optimized task configurations, with up to 60% performance improvement on the target CPUs, with respect to the reference configuration.

Table 5.3: *For each kernel of the Stereo-Matching application, rows 1-3 show the number of feasible solutions on the platform devices considered in the experimental setup. The last row shows the average size of the feasible set, among the target devices, as percentage of the full design space.*

PLT	DEV	WinBuild	ConvertInty	AggHor	AggVerA	AggVerB	WTA	CrossCheck	RefineHor	RefineVer
PLT1	GPU	365	3856	1106	1212	1557	3856	3856	1462	1035
PLT1	CPU	1005	4994	1713	1738	2138	4994	4994	2111	1538
PLT2	CPU	963	4308	1683	1718	2081	4308	4308	2035	1529
% Design Space		0.01%	0.06%	0.02%	0.02%	0.03%	0.06%	0.06%	0.03%	0.02%

Table 5.4: *Results of DSE on the feasible solution set of OpenCL Stereo-Matching (WS = workgroup size, BS = block size, TS = tile size, with $TS = WS_x \times WS_y \times BS_x \times BS_y$). The results contain one configuration of optimal kernel parameters for each target device.*

	PLT1, GPU					PLT1, CPU					PLT2, CPU				
Kernel	WS_x	WS_y	BS_x	BS_y	TS	WS_x	WS_y	BS_x	BS_y	TS	WS_x	WS_y	BS_x	BS_y	TS
WinBuild	16	32	1	1	512	32	48	1	1	1536	12	3	2	6	432
ConvertInty	96	3	2	8	4608	48	72	1	1	3456	48	6	8	4	9216
AggHor	16	4	1	1	64	48	1	2	8	768	12	1	8	2	192
AggVerA	16	32	1	1	384	4	3	8	8	768	2	8	8	4	512
AggVerB	16	18	1	1	288	6	2	8	8	768	16	24	1	1	384
WTA	128	1	1	1	128	384	18	1	1	6912	384	2	1	1	768
CrossCheck	16	3	4	6	1152	32	1	6	2	384	48	9	2	1	864
RefineHor	16	6	1	1	96	128	1	1	6	768	24	1	8	1	192
RefineVer	8	36	1	1	288	2	6	4	8	384	16	3	1	8	384

DSE Phase 2 – Task mapping. The mapping of the OpenCL task graph to the target heterogeneous platforms follows the proposed approach. For each platform we consider 4 different task graphs, one for each value of `hypo_step`, depending on the *QoS* (result accuracy) required by the user. For each task graph configuration, the mapping problem is solved to identify the optimal mapping. These points represent the solution of a multi-objective optimization, since *throughput* and *QoS* are two design objectives.

Figure 5.11 reports cycle time (the time between two consecutive output frames) with respect to the average disparity error (the higher the `hypo_step`, the higher this error) for PLT1 and PLT2. For each platform, four mapping configurations have been considered, by varying from a single device up to multiple heterogeneous devices. This plot shows the improvement on the throughput metric for a given level of average disparity error. The maximum observed improvement is 3x when `hypo_step` is 1, thus for the largest task graph shown in Figure 5.8. Not necessarily the throughput increases when using multiple devices: the fourth configuration in Figure 5.11 shows a sub-optimal mapping, where tasks are equally divided among all available devices. On PLT1, this configuration does not perform better than the single-device execution on CPU: the reason is that the speedup contribution from multi-device execution (CPU and GPU) does not compensate the overhead of H2D and D2H memory transfers for copying data from one device to the other. Thus, without the proposed optimization driven by a constraint solver, it might be difficult for the developer to find a mapping that balances the pipeline stages and maximizes the throughput.

Scalability. The flattened model of Figure 5.8 is given as input to the constraint program to identify the best mapping which maximizes the average throughput. By setting a constraint on the maximum cycle time ($150ms$ on PLT1 and $250ms$ on PLT2), the Minizinc solver identifies a solution that satisfies the mapping problem of Section 5.2.2. The search time is less than $2s$ for all values of `hypo_step`, except for `hypo_step` set to 1 on PLT2 ($> 30min$). In this case, the constraint solver spends most time in evaluating all possible mappings (3 platform devices) of the cost-matching cascade (24 tasks). To cope with this problem, we added one constraint to the mapping formulation, based on the analysis of the task graph. When a sequence of tasks is connected like in the cost-matching cascade, i.e. the output of one task is the input to the next task, the optimal solution will be found by splitting this cascade into up to ND sub-ranges, where ND is the number of platform devices. This mapping minimizes the time overhead of H2D and D2H memory transfers between consecutive tasks. By adding the new constraint, the search time is reduced to $2.5s$.

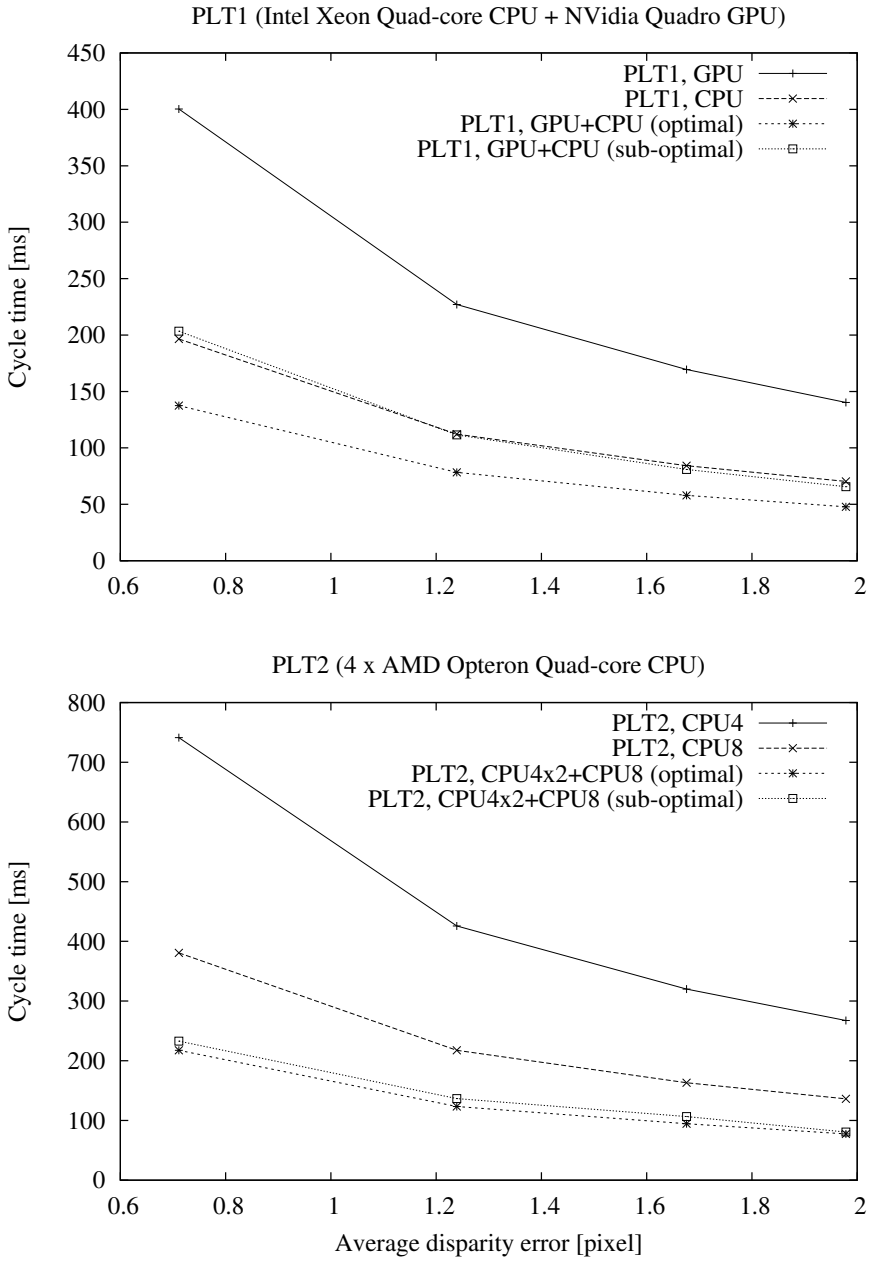


Figure 5.11: *Trade-off between cycle time and disparity error of the Stereo-Matching task graph deployed on PLT1 and PLT2, for different mapping configurations.*

5.5 Conclusions

The optimization and customization design flow presented in this chapter allows to efficiently map OpenCL applications to heterogeneous parallel platforms. For such platforms, it is generally hard to exploit pure data-parallelism, since devices from different vendors have to be instantiated as different OpenCL contexts. The conversion to a task-parallel implementation, if possible, may suffer from unnecessary data transfers from host to device, and vice-versa, so it requires manual optimization to achieve good performance for each target computing device.

The proposed design flow is based on a constraint solver to identify a set of feasible configurations with respect to platform-specific constraints and on a DSE framework for tuning task parameters. The constraint solver automatically pruned the search space of the Stereo-Matching OpenCL kernels to 0.1% of the original size. Combined with the proposed mapping methodology, this enabled the generation of optimized OpenCL software pipelines, some of them even 3x faster than the single-device implementation for the selected use cases.

In the current mapping solution, the scheduling of tasks mapped to the same computing device has been limited to satisfy data dependencies in the application task graph. Future work will investigate more advanced scheduling techniques to enable overlapping of computation and communication.

Part II

Application Auto-Tuning and Run-Time Management

Overview

To provide the performance requirements in multi-application scenarios, this thesis exploits both application auto-tuning and Run-Time Resource Management (RTRM). Resource allocation on multi- and many-core accelerators can either exploit physical partitioning of the available processing cores among the applications (e.g. by means of the *device fission* OpenCL API) or use time-based scheduling. In any case, in order to enable *performance-aware* allocation, the run-time manager needs to know the resource requirements of each application. The design-time exploration phase presented and discussed in Part I is exploited to build such knowledge-base in the proposed approach. A similar approach was already studied for Multi-Processor System-on-Chip (MPSoC) platforms in [123, 80], whereas this thesis considers the problem for general purpose platforms with parallel accelerators.

The contribution of Chapter 6 is a light-weight RTRM technique to provide resource sharing for computationally intensive OpenCL applications. It exploits an Application-Specific Run-Time Manager (AS-RTM) to enable each running application to take autonomous decisions for run-time adaptation. Then, Chapter 7 presents a more general Run-Time Management (RTM) approach, which exploits both centralized RTRM and application-level auto-tuning. This framework, which integrates different independent tools, was developed in the context of the 2PARMA European project [107].

CHAPTER 6

Application Auto-Tuning with Autonomous RTRM

This chapter presents the application run-time support to exploit the set of optimal Operating Points (OPs) identified by Design Space Exploration (DSE) in Chapter 3. The proposed approach exploits application auto-tuning, implemented by the Application-Specific Run-Time Manager (AS-RTM), of both application-specific *dynamic knobs* and computational parallelism.

The AS-RTM allows an application to take run-time adaptation decisions, autonomously. This has two main advantages: i) a non-invasive application design, in terms of source code, and ii) a very low run-time overhead, since it does not require any central coordination of a supervisor nor communication between the applications.

Another contribution of this chapter is a light-weight Run-Time Resource Management (RTRM) technique for computationally intensive applications on multi-core platforms, published in [4]: it considers the information of system workload gathered by platform sensing to take reconfiguration decisions, while minimizing the impact on other applications that share the same resources.

6.1 Related Work

The run-time management approach described in this chapter has some flavors of *autonomic computing* [65], an initiative started by IBM in 2001 to develop self-managing features for distributed computing systems. This vision leverages on a variety of architectural frameworks based on *self-regulating* autonomic components, mainly exploited in the research area of multi-agent systems. However, this vision can be extended to embrace also Run-Time Management (RTM) techniques for adaptive resource allocation and application auto-tuning on multi- and many-core platforms. Indeed, the RTM technique presented in this chapter exposes three main characteristics of an autonomic-system: automatic, adaptive and context aware.

In line with the autonomic computing vision, *invasive computing* [114] represents a design paradigm for resource-aware programming on parallel computing platforms. This paradigm is based on the *invade and retreat* approach, in which an application enters a phase called *invasion* whenever some computation needs to be parallelized over some neighbor processors; afterwards, when the compute-intensive code region has completed, a *retreat* phase allows to release resources.

A proof-of-concept implementation of an *invasive* resource manager was made in [27]. The proposed approach, based on a central X10 invasive framework, exploits *scalability curves* provided by the application developer for performance-aware resource allocation. As in our approach, in [27] resource adaptation is not expected to speed up each running application, but the goal is to speed up the system as a whole. However, the approach proposed in this chapter tries to solve the same problem without a central manager, while accounting for performance degradation due to resource sharing, such as for the memory bandwidth, and therefore exploiting *dynamic scalability curves*.

Invasive computing was also presented, in [119], as a distributed framework based on game theory for homogeneous many-core systems. However, distributed approaches suffer from communication overhead and convergence time. With respect to [119], our approach behaves as a multi-agent distributed system too, but it exploits performance-accuracy trade-offs at application-level in order to extend the operational range within a given resource quota.

Effective application-level tuning is leveraged by software-based approximate computing [99]. As discussed in Section 2.1, these techniques allow to design an application so that the trade-off between performance and Quality-of-Result (QoR) metrics can be dynamically tuned at run-time, by means of *dynamic knobs* [56]. The work presented in [55] combines

the concept of *dynamic knobs*, to support run-time adaptivity, and that of *heartbeat*, to provide a way to monitor application latency and throughput. Their framework, SEEC, uses a run-time manager with different levels of adaptation, from a simple closed-loop control scheme to a more refined machine learning manager. However, this solution differs from the one proposed in this chapter because it lacks design-time support.

A machine learning approach to multi-core resource management is presented in [81]. In this approach, a resource manager monitors each application execution and learns a predictive model of their responses to allocation decisions. The performance model is indeed a Response Surface Model (RSM), based on Artificial Neural Networks (ANNs). Thus, it allows to take into account interference between concurrent applications due to the contention on shared resources, which cannot be assigned in an isolated manner (e.g. memory bandwidth, cache utilization). This generates a performance degradation, which is considered both by the run-time manager in [81] and ours. However, in order to reduce the run-time overhead due to model training as well as the complexity of the decision space, our run-time manager exploits a knowledge-base obtained at design-time through DSE – the Pareto-set – and just refines the *operating point* to account for performance degradation.

Similarly, in [122] design-time and run-time techniques are combined in order to train a global resource manager. A step forward made on top of the previous approach has been done in [80] with a run-time management framework, called ARTE, supported by DSE. While in [122] the run-time manager was designed for instantaneous throughput maximization, the goal of ARTE is to minimize application response times. However, even in this case, the run-time manager is a single one (system-wide) and, at the application level, it provides only the possibility to change the parallelization.

Thus, design-time application characterization to support run-time management – as done in this thesis – is not a new idea, but there are some main differences with respect to [122, 80]:

- The performance model obtained at design-time, in our approach, is specific for the individual applications, but the run-time workload can be mixed. Thus, our run-time management approach is not application-specific and allows to cope with unpredictable workload variations.

- The run-time manager presented here controls not only the application parallelism but also application-specific metrics, e.g. the quality of a decoded video or the accuracy of some computation, enabling more fine-grained optimal auto-tuning.

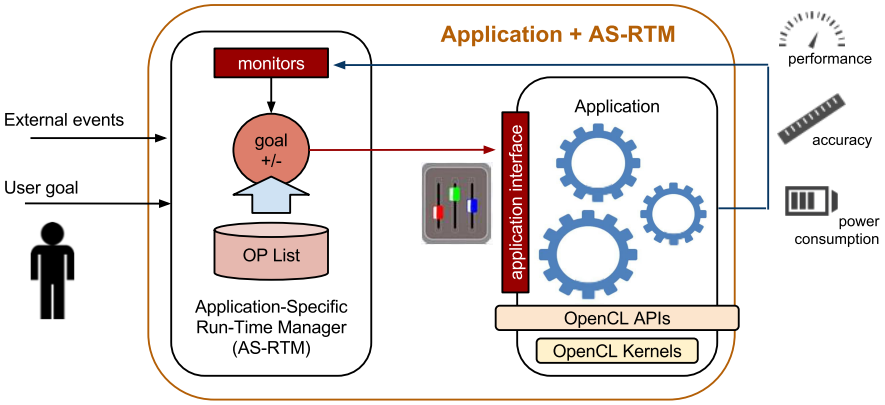


Figure 6.1: *Application adaptivity through the Application-Specific Run-Time Manager (AS-RTM).*

- While the previous work mainly targeted Multi-Processor System-on-Chips (MPSoCs) platforms, the techniques presented in this thesis have been validated on general-purpose multi-core platforms, thus demonstrating that the solution is portable.

6.2 Target Adaptive Framework

The basic idea of the proposed adaptive framework consists of exploiting the orthogonality between application auto-tuning and Run-Time Resource Management (RTRM) for computationally intensive OpenCL applications. The run-time decisions are taken based on profiling information gathered at design-time, which consists of a set of application configurations, namely the Operating Points (OPs), optimal with respect to application performance metrics (e.g. throughput) and resource usage (see Chapter 3).

In this section, first the application-oriented self-adaptive layer is presented in Section 6.2.1, then the proposed resource-aware extension in Section 6.2.2.

6.2.1 Application adaptivity through *dynamic knobs*

In the proposed approach, each application is linked to the framework library that provides an Application-Specific Run-Time Manager (AS-RTM). The main purpose of the AS-RTM is to manage application adaptivity by tuning the *dynamic knobs* [56] – application parameters that can be changed at run-time without recompiling the application.

The AS-RTM is generic, however its behavior can be customized for each application given a different list of run-time configurations – the OPs. The OP data structure contains a set of parameters, which represent the values of the application *dynamic knobs* for a specific configuration, and the metric values profiled at design-time. As shown in Figure 6.1, the AS-RTM allows to define one or more application goals. A goal represents a *soft constraint* on a certain metric – e.g. the frame-rate or the Quality of Service (QoS) – or even on a parameter – e.g. the number of threads – that can be dynamically set by the user or selected by the application itself depending on external events. Such constraints are assumed to be strictly ordered by their priority.

Similarly to the *Heartbeat* framework [54], this AS-RTM uses high-level monitors of the performance (such as a throughput monitor, a QoS monitor or any user-defined monitor) to sense the execution context and to react to any change in the application run-time requirements.

The AS-RTM integration in a third-party application is straightforward: it does not require refactoring the application logic to meet an execution template (such as in [19] and [123]) but only wrapping the code to be profiled with the monitor calls.

6.2.2 Proposed Resource-Aware AS-RTM

The proposed approach aims at analyzing RTRM for compute-intensive workloads, such as OpenCL applications, in multi-application scenarios.

In a plain OpenCL application, the platform resources are managed by the OpenCL runtime at application-level, so an application is enabled to use all devices available on an OpenCL platform and, by default, the entire quota of each device. On a multi-core CPU, for example, the OpenCL run-time binds each application to all compute units by default and relies on the OS scheduler to assign CPU time to all applications (seen as different processes by the scheduler).

Any resource-related parameter (e.g. the computational parallelism) could be treated as a generic application parameter. However, a plain management of such parameters could lead to system configurations where the total amount of computational parallelism required by the running applications exceeds the system resources. In turn, this would result in a degradation of application performance since the OS scheduler limits the process CPU usage, thus the application performance would be unpredictable, as the number of deployed applications (processes) changes over time.

To overcome this problem, we propose a resource-aware AS-RTM, which takes into account the CPU usage (as we target multi-core CPU platforms),

for self-limiting the application parallelism (e.g. the number of working threads). At each application cycle, the AS-RTM monitors the goal status. Whenever a goal is not satisfied or has been changed (e.g. the required frame-rate has been decreased), in order to select the most suitable OP, the AS-RTM follows this procedure:

1. The AS-RTM updates the internal constraints on the OPs to meet the current goals.
2. If there is at least one OP that satisfies all the constraints, the AS-RTM chooses the one with the highest rank value. Otherwise, it chooses the OP closest to the valid region defined by the constraints, starting from the one with highest priority.

According to this decision policy, we add a constraint on the process CPU usage, on top of the application-specific ones. The value of this constraint is initialized to the maximum system CPU quota (Γ). At run-time, by monitoring the system CPU usage (γ) and the process CPU usage ($\pi_{measured}$), the AS-RTM selects an OP only if the profiled CPU usage of the OP ($\pi_{profiled}$) satisfies the following constraint:

$$\pi_{profiled} \leq \Gamma - \gamma + \pi_{measured} \tag{6.1}$$

If only one application is running, γ and $\pi_{measured}$ are equal, thus the application is allowed to use the entire CPU resource. Otherwise, if the platform is congested, Γ and γ have the same value, which forces the AS-RTM to select among the OPs whose profiled CPU usage fits the quota assigned by the OS scheduler.

6.3 Experimental Setup

This experimental section considers a case study based on the OpenCL Stereo-Matching application, targeted to the two multi-core CPU platforms described in Section 2.4.

6.3.1 Definition of metrics

The Stereo-Matching application has two metrics of interest, namely the *frame-rate* (measured as [frames/s]) and the *disparity error*, which represents a measure of the average error associated with the application result (the pixel disparity, see Section 2.3). However, the tests in this chapter consider only normalized metrics, defined as follows, in order to abstract our analysis from the specific application.

Normalized Actual Penalty (NAP)

This metric measures the degree of user satisfaction, with respect to a frame-rate goal set at the application start. The frame-rate goal is a soft real-time constraint, which should be met independently from the machine workload and resource availability.

$$NAP = \frac{GOAL_{measured} - GOAL_{demanded}}{GOAL_{measured} + GOAL_{demanded}} \tag{6.2}$$

Normalized Error

This is a measure of the output quality normalized on the range of valid values, so that $ERR = 1$ when the application runs with the configuration that provides the lowest – but still acceptable, with respect to design requirements – output quality; while $ERR = 0$ when the quality is highest. It was obtained for Stereo-Matching from the *disparity error* (DErr) as follows:

$$ERR = \frac{DErr_{OP} - DErr_{MIN}}{DErr_{MAX} - DErr_{MIN}} \tag{6.3}$$

Difference w.r.t. to off-line profiling

Another metric of interest is the *deviation* (DEV) of the metrics (e.g. cycle period) observed at run-time with respect to the expected values, i.e. the OP metrics profiled at design-time.

$$DEV = \left| \frac{Tcycle_{measured}}{Tcycle_{OP}} - 1 \right| \tag{6.4}$$

Since the following tests consider dynamic scenarios, for the NAP and ERR metrics a synthetic value is computed to take into account the temporal dimension:

$$NAP_{AVG} = \frac{\int NAP(t)\, dt}{\Delta t}\ ,\ ERR_{AVG} = \frac{\int ERR(t)\, dt}{\Delta t} \tag{6.5}$$

6.3.2 Definition of dynamic workload

A dynamic workload, in this thesis, consists of a set of applications with different schedules (start time), amount of data to process (number of frames in Stereo-Matching) and performance requirements (frame-rate). This use-case is aimed at mimicking the workload expected in resource consolidation, specifically targeted to offloading computationally intensive OpenCL

applications [42]. Although it uses only one type of application (Stereo-Matching), a dynamic workload is mimicked by exposing the following parameters:

- *Start delay*: each application instance is started upon user request, thus different start times are used.
- *Amount of input data*: each Stereo-Matching instance is required to process a different number of frames.
- *Frame-rate goal*: soft real-time constraint to guarantee a certain response time, as demanded by the user.

The above parameters are randomly chosen for each Stereo-Matching run, within a range of values shown in Table 6.1.

6.3.3 Run-Time Management description

Three Run-Time Management (RTM) configurations have been considered:

1) Plain-Linux: Baseline implementation without run-time adaptivity. Each application instance is deployed as a plain OpenCL application, thus it is bound by default to all processing elements available on the CPU. On the one hand, since there is no manager, this configuration relies on the OS to schedule tasks from different applications. On the other hand, the application runs a fixed configuration, with 50% QoS.

2) AS-Linux: In this configuration the AS-RTM can switch the Operating Point to trade-off between performance and QoS. Although the computational parallelism can be controlled by the AS-RTM through an application dynamic knob, the effective resource usage still depends on the allocation of CPU user time by the OS scheduler.

3) RA-AS-Linux: Configuration implementing the proposed Resource-Aware AS-RTM. Differently from the previous configuration, here the computational parallelism is used orthogonally with respect to the application-specific knobs. It implements the technique presented in Section 6.2.2, based on monitoring of the system CPU usage for *smart* adaptation of the resource requirement.

Table 6.1: *Range of values for the random parameters of dynamic workload tests.*

Parameter	AMD	Intel
Number of frames	10-840	
Frame-rate goal [frames/s]	1-7	
Start delay [s]	0-90	
Num. instances	1-6	1-4

6.4 Experimental Results

The experiments described in this section have been carried out on the AMD and Intel multi-core CPU platforms presented in Section 2.4. In Section 6.4.1, a single stereo-matching application, with some constraints on resource usage, is used to assess the capability of the proposed AS-RTM framework of exploiting the available trade-offs between performance metrics. In Section 6.4.2, we evaluate the dynamic behavior of the stereo-matching application, in a multi-application scenario, with different RTM techniques, including the proposed one for efficient resource sharing based on platform sensing (*Resource-Aware AS-Linux*). We conclude this section with a campaign of experiments in Section 6.4.3 that execute random dynamic workloads to compare the different techniques analyzed individually in the previous experiments.

6.4.1 Application Auto-Tuning Results

This experiment aims at assessing the benefits of application adaptivity. It consists of a single Stereo-Matching application deployed on the Intel platform (PLT1 in Section 2.4), with 200 frames to process. The test is repeated for each possible number of cores (4 in total on the Intel platform), with the frame-rate goal incremented at each run from 3 to 21 frames/s.

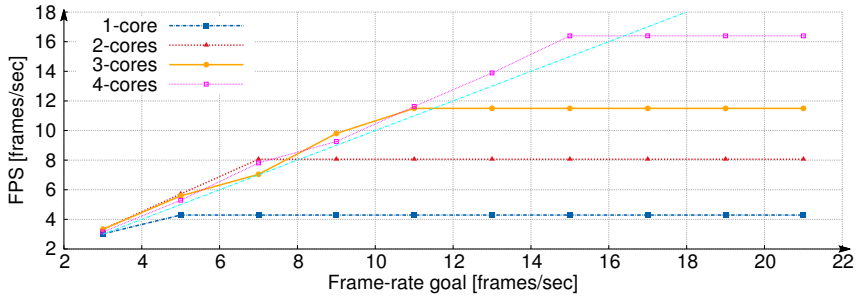
The results are shown in the three plots of Figure 6.2, where the x-axis is the goal value and y-axis represents, in order, the average measured frame-rate (6.2a), the average normalized error (6.2b), and the average NAP (6.2c). With the highest resource availability (4-cores) the AS-RTM can provide 3 frames/s without quality loss (ERR$\simeq$0% in Figure 6.2b). On the contrary, configurations with lower resource availability show a quality loss which ranges from 20% to 50%, depending on the number of cores. This means that there is a range of goal values, different for each amount of available resources, where the AS-RTM can reduce the computational accuracy in return for higher performance, in order to meet the requested goal.

Figure 6.2a shows a similar behavior in all tests but with different thresholds for the maximum reachable frame-rate: the test with 1-core provides up to 4.3 frames/s, with 2-cores up to 8.1 frames/s, with 3-cores up to 11.5 frames/s and with 4-cores up to 16.4 frame/s. After these frame-rate thresholds, the AS-RTM (already in the OP with lowest quality of result) cannot find any suitable OP to meet the goal, thus the NAP value starts growing (Figure 6.2c).

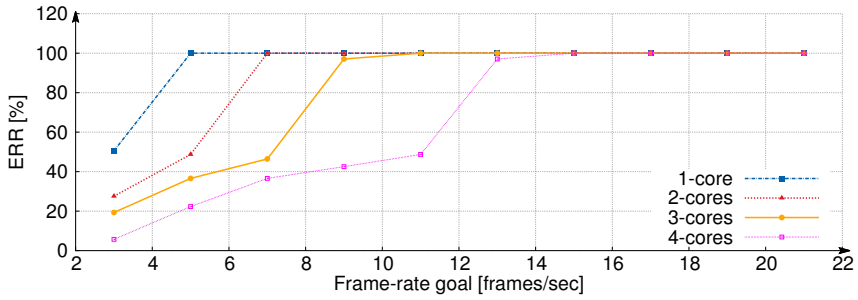
In conclusion, this test demonstrates that the AS-RTM allows to satisfy higher throughput demands by exploiting the possible trade-offs in terms of

performance versus computational error. The dynamic workloads presented in the next section will benefit from this feature, since in a multi-application deployment scenario each instance cannot use the full platform, but is constrained to a subset of resources.

(a) *Average measured frame-rate vs. frame-rate goal*



(b) *Average normalized disparity error vs. frame-rate goal*



(c) *Average Normalized Actual Penalty vs. frame-rate goal*

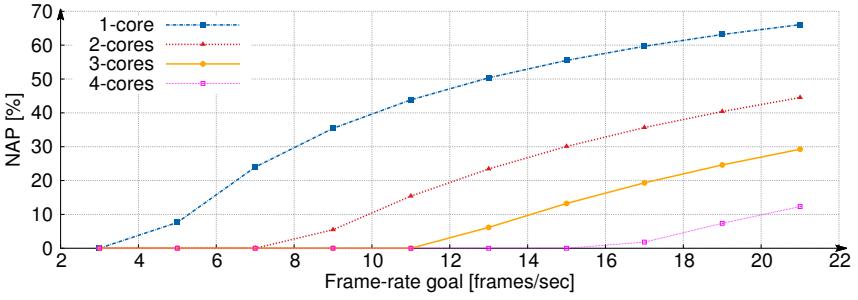


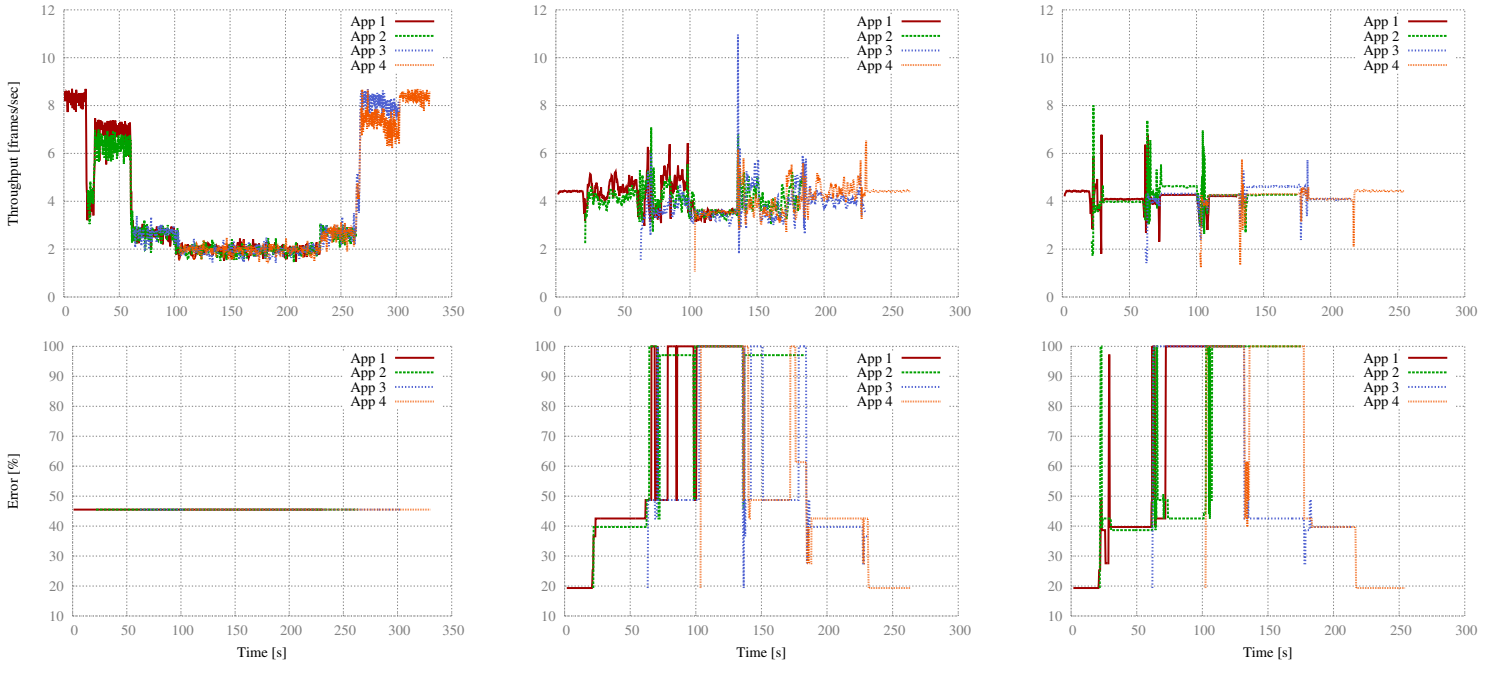
Figure 6.2: *Observed frame-rate, normalized error and Normalized Actual Penalty (NAP) by varying the frame-rate goal and the number of CPU cores.*

6.4.2 Evaluating RTM Strategies

In this section, three RTM strategies are compared in terms of adaptability, predictability and fairness, by analyzing a multi-application sequential scenario. It consists of four Stereo-Matching instances, which are executed on the Intel platform, with the following start times: $t_{A1} = 0s$, $t_{A2} = 20s$, $t_{A3} = 60s$ and $t_{A4} = 100s$. The number of frames to be processed by each instance has been chosen to let all the applications run together for approximately $30s$, then they complete their execution at different times. All instances have the same throughput goal (4 frames/s) and their AS-RTM is configured to minimize the disparity error. We can logically partition the experiment in two phases. In the first phase new applications are launched, so we can observe how already running applications react when the new applications steal resources. The second phase begins when the oldest instance has completed its execution. In this phase, one by one, all applications leave the execution context, so we can observe how the remaining instances exploit the resources that are released. Figure 6.3 shows the three evaluated RTM strategies: Plain-Linux (6.3a), AS-Linux (6.3b) and the proposed Resource-Aware AS-Linux (6.3c). For each strategy, the plots show the throughput and disparity error profiled at run-time, in a time window of 300 seconds.

Plain-Linux. Since the application knobs are fixed, the error is constant; conversely, the throughput is strictly dependent on the workload. With one or two instances, the goal is met, although when the second instance starts we can observe some instability ($t = 20s$). When more than two instances are running, the goal cannot be reached. Moreover, the performance presents a high rate of fluctuations and is not predictable.

AS-Linux. When only one application is running, the throughput is stable and the disparity error is constant. As soon as the second application is started ($t = 20s$), the throughput of both instances starts oscillating but the error remains constant. The reason for this is that the AS-RTM does not change the OP (the throughput is above the goal) but, since the total amount of resources demanded doubles the number of cores, the throughput is strongly related to the scheduler policies. After $60s$, the third application is started and even more resources are demanded, strengthening the relation between OS scheduling and throughput oscillation. In this case, the measured throughput can go below the goal value, forcing the AS-RTM to select a faster OP, which in turns boosts the oscillation.
Although all application instances have the same priority and the same list of OPs, they select different OPs as we can notice in the error plot. In conclusion, this configuration is not fair nor predictable.



(a) *Plain-Linux* **(b)** *AS-Linux* **(c)** *RA-AS-Linux*

Figure 6.3: *Behavior of the run-time management strategies, in terms of throughput and normalized error, for Plain-Linux (6.3a), AS-Linux (6.3b) and the proposed Resource-Aware AS-Linux (6.3c). The application throughput goal is set to 4 frames/s.*

Resource-Aware AS-Linux. The behavior of the proposed RTM technique is quite different: after an initial transitory period, the constraint on the CPU utilization forces the AS-RTM to use only OPs that fit in the available resources, preventing throughput oscillations. Whenever a new application starts or ends, the AS-RTM waits until the CPU usage, of both the system and the application, becomes stable before updating the CPU usage constraint. For short periods, the number of threads might be greater than the number of cores, thus some oscillations can be observed (e.g. $t = 20s$, $t = 60s$). Then, such undesired oscillations end once the platform resources have been partitioned among the applications.
The CPU monitor allows the AS-RTM to gain predictability, however – as the disparity error plot shows – this strategy is not fair because the resource allocation is not coordinated among the running applications. This problem will be overcome with the two-level RTRM approach presented in the next chapter.

6.4.3 Dynamic Workload Results

This section describes the results obtained by deploying a multi-application configuration on both reference platforms. The maximum number of instances and the maximum frame-rate goal are shown in Table 6.1.

As shown in Figures 6.4a and 6.4b, *Plain-Linux* has the worst NAP metric: although the single application can reach all throughput demands, concurrent execution of applications with different resource demands introduces high penalties on the performance metrics. In this configuration, all applications use by default the entire CPU (device fission is disabled). This introduces a high rate of context-switches, which degrades the measured frame-rate. As a consequence, the difference between the design-time and run-time profiling is highest for this configuration (Figures 6.4c and 6.4d) and such deviation continues increasing as we deploy more concurrent applications. This result was expected because the OpenCL library relies on the OS scheduler to allocate user time to different applications.

In the case of *AS-Linux*, the QoS metric (Figures 6.4e and 6.4f) is tuned at run-time to react to variations in the system workload. The error associated to the application output is below *Plain-Linux* in scenarios with 1-3 instances, above for scenarios with 4-6 instances. The Normalized Actual Penalty (NAP) benefits from the wider range of trade-offs, so it is lower than in *Plain-Linux*; however, the predictability of performance metrics is low, as shown by the performance deviation bars (Figures 6.4c and 6.4d).

This limitation is overtaken by the proposed *Resource-Aware AS-Linux*,

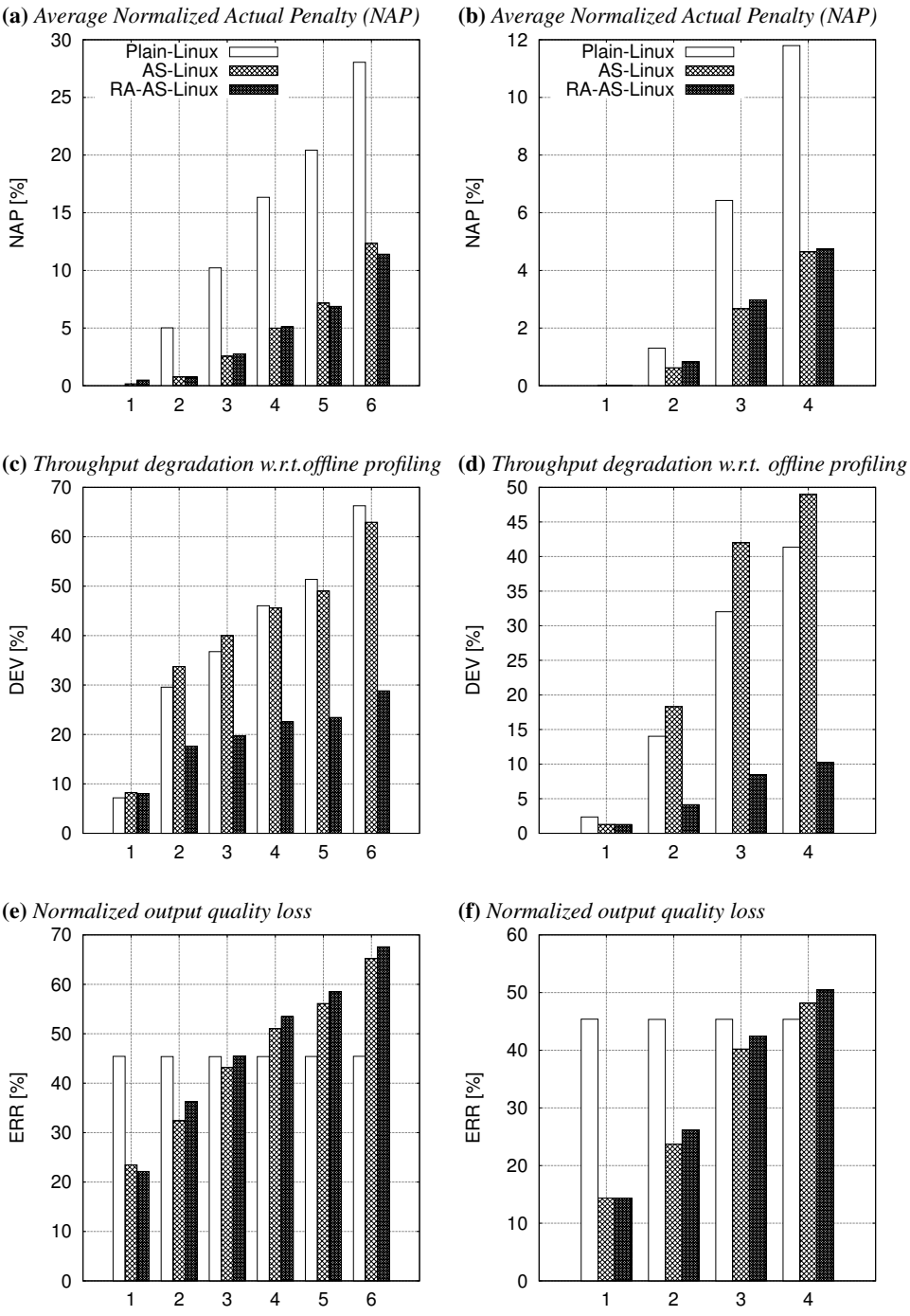


Figure 6.4: *Dynamic workload analysis by varying the number of Stereo-Matching instances, on the AMD platform (6.4a, 6.4c, 6.4e) and Intel platform (6.4b, 6.4d, 6.4f).*

where an application is allowed to use resources only if these are available. Thus, *RA-AS-Linux* performance is slightly better than *AS-Linux*, in most cases, because the AS-RTM can take better adaptation decisions, at run-time, thanks to higher accuracy and predictability of the performance metrics in high contention scenarios.

The average Normalized Actual Penalty (NAP) (Figures 6.4a and 6.4b) is the metric that better summarizes this analysis. We can observe, as expected, an increasing NAP for all configurations as the workload grows. Nevertheless, the adaptive configurations (supported by the AS-RTM) always reduce the NAP with respect to the plain configuration, which means that the frame-rate goal is met much more frequently.

6.5 Conclusions

The experimental results show that, while Run-Time Resource Management (RTRM) is necessary to provide lower variance of the application performance, the application auto-tuning layer is fundamental to trade it off with respect to the computation accuracy, in order to extend application operability with low resource availability. At this aim, the light-weight RTM technique, presented in this chapter and targeted to compute-intensive applications, allows to take local decisions on resource utilization at application level, for efficient resource sharing.

Differently from previous approaches (e.g. "invade and retreat" [119] or the *Barbeque* run-time resource manager presented in Section 2.6), in this approach applications act like autonomous agents, without coordination among them. On the one hand, this solution has the advantage of being non-intrusive from a design point of view, since it does not require a communication infrastructure; on the other hand, it does not provide any guarantee of *fairness* nor *optimality* in resource allocation.

To achieve system-level objectives such as *fairness*, the next chapter will present a more general solution which exploits a two-level run-time management framework: resource allocation is delegated to a centralized resource manager while application-specific auto-tuning is controlled by the AS-RTM.

CHAPTER 7

Combining Application Adaptivity and System-Wide RTRM

The Run-Time Resource Management (RTRM) technique presented in the previous chapter was specifically targeted to computationally intensive applications. Besides not supporting different priority levels and being limited to multi-core x86 platforms, that technique did not ensure a fair allocation of resources. Therefore, this chapter presents a more efficient and portable run-time management solution, developed in the context of the 2PARMA FP7 European project [107] and published in [3], which could scale from embedded to High Performance Computing (HPC) systems.

The proposed methodology still exploits – as in Chapter 6 – the synergy between design-time and run-time but, in addition to the Application-Specific Run-Time Manager (AS-RTM), it integrates *Barbeque*, an open source system-wide run-time resource manager presented in Section 2.6. By combining application auto-tuning, provided by the AS-RTM, with system-wide RTRM, a two-level Run-Time Management (RTM) framework is implemented and demonstrated on a multi-core x86 Non-Uniform Memory Access (NUMA) workstation. With respect to Chapter 6, resource management is not done by each application autonomously but it is coordinated by

a central manager, in order to take into account the application priorities and to enable optimization of system-wide metrics.

7.1 Related Work

Several previous works targeting run-time resource management have been already presented in the previous chapter, especially focusing on adaptive and self-aware computing. This section presents some examples from a specific category that exploits, like in the proposed approach, design-time profiling and optimization to support run-time management.

One reference methodology, in this context, exploits Design Space Exploration (DSE) to customize a run-time manager for embedded multi-core platforms [122, 79]. Just like in our approach, their assumption is that the set of applications is known at design-time, therefore it is possible to prune the decision space from suboptimal resource partitioning schemes and implement a light-weight run-time manager. However, there are some main limitations:

- The previous work targeted an embedded platform, such as an MPEG4 encoder chip, but the run-time manager was demonstrated on a simulation platform. Therefore, depending on the simulation/speed accuracy, performance deviation with respect to offline profiling does not occur. Under these conditions, run-time scheduling is precise, thus enabling an instantaneous convergence to a stable configuration. On a real platform, as we target, this never happens because of contention on shared resources and feedback control is necessary to compensate for performance variations.
- The demonstrator, although in simulation, was configured with a limited number of cores (only 7 in [122, 79]) while today's multi-core accelerators typically contain 64 (see [77]) or more PEs.
- The only application parameter controlled by the run-time manager is the number of threads, in [122], beside the core frequency on platform cores, in [79], while our framework allows to consider more fine-grained application auto-tuning.
- The run-time manager in previous work just takes decisions on optimal resource partitioning but does not enforce the assigned quota, whereas our framework does.
- With respect to [122, 79], our framework exploits a hierarchical approach, in which resource allocation is done by a central resource

manager while fine-grained application auto-tuning is done at the application level. This enables better scalability, both with the number of platform cores and the number of concurrent applications.

The synergy between design-time and run-time was also exploited in [123]: the proposed framework is hierarchical like ours but the application integration is not as much straightforward as with the *Barbeque* API [19]. Moreover, it does not consider application parallelization and it was not demonstrated with a real multi-application workload.

The work in [80] is an extension of [79], considering not only instantaneous throughput maximization but also latency to ensure application responsiveness. However, this is achieved through a run-time manager precisely tuned at design-time for a specific application use case. Thus, it is most suitable for an embedded platform, while the proposed approach – including the design methodology and the run-time software layer – can be applied on a wider range of platforms. The portability of the framework proposed in this thesis addresses one important aspect of modern computing platforms, namely the convergence of both architectures and programming paradigms in the embedded and HPC domains.

A novel contribution of this chapter is that the design-time profiling phase is aimed at optimizing applications, written in OpenCL, for a specific platform besides characterizing the workload. At this aim, we exploit platform performance counters exported through a portable *Barbeque* API to tailor an application implementation for a specific architecture. In this context, *cache pirating* [100] is a promising technique to model cache sharing, in order to predict throughput variations in multi-application scenarios. Indeed, cache misses have an immediate impact on the bandwidth used to access shared main memory, which represents a shared resource on multi-core platforms. Thus, co-scheduling decisions taken by a run-time manager should take into account the application profile, being it memory-bound or compute intensive, as recently proposed in a *Barbeque* extension [78].

Another important aspect, discussed in several points of this thesis, is platform heterogeneity. The work in [85] presents a scheduling framework for GPU accelerators in the domain of medical imaging with timing requirements. The scheduler is capable of utilizing multiple GPUs in a system to minimize the average response time of applications. Moreover, it enables priority-based task scheduling and preemption on GPU to fulfill the timing requirements of high-priority dynamic tasks. Like the methodology proposed in this chapter, the framework in [85] requires off-line characterization of the kernel execution time. However, their framework is based on CUDA, which limits platform heterogeneity to different Nvidia GPU devices.

StarPU [16] and OmpSs [39] propose run-time frameworks specifically focused on performance optimization of heterogeneous platforms. However, in both cases, task scheduling strategies do not target any other objective but performance maximization and load balancing. On the contrary, the framework proposed in this chapter exploits the *Barbeque* Run-Time Resource Management to enable fair, priority-based resource allocation [19].

We conclude with an example of run-time management from the industry, the Grand Central Dispatch (GCD) technology introduced by Apple in both MacOS and iOS. GCD allows parallel tasks in a program to be queued up for execution and, depending on the availability of processing resources, to be scheduled on any of the available platform resources. The support to OpenCL, starting from version 10.7 [14], allows GCD to also handle OpenCL compute devices, suggesting the application which available OpenCL device is best for running a particular kernel.

7.2 Proposed Approach

The proposed methodology provides effective run-time management of parallel applications sharing computing resources on a multi-core platform. As in Chapter 6, the goal is to support dynamic workloads and to improve the average system performance, but considering the resource requirements and the priorities of each individual application. Thus, this section describes how three independent tools – *MOST*, *Barbeque* and the *AS-RTM* – have been integrated to build a hierarchical Run-Time Resource Management (RTRM) framework. The experiments in Section 7.3 will show the benefits of using the proposed framework on a NUMA x86 workstation, compared to the light-weight solution presented in Chapter 6 or to a plain *Barbeque* configuration [19].

7.2.1 Design-Time

DSE has been applied to the optimization of parallel streaming applications, implemented with a parametric design as described in Chapter 3. The methodology presented in this chapter exploits in the DSE phase the synergy between *MOST* and *Barbeque* (BBQ), as shown in Figure 7.1.

The exploration is driven by the DSE framework (*MOST*), which forces a fixed configuration of resources allocated to the application, for each profiled design point: this is done by creating, before starting the application profiling, a custom application recipe (Section 2.6) containing a single Application Working Mode (AWM), with the CPU quota specified by *MOST*. In this way *Barbeque* creates an execution context, limiting the platform

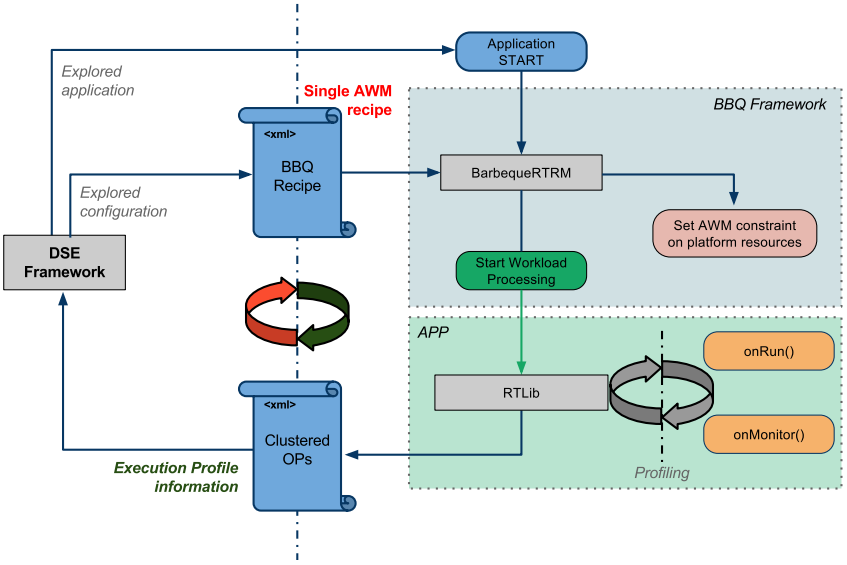


Figure 7.1: *Design Space Exploration and SW-RTRM synergy for the characterization of Application Working Modes.*

resources available to the application. On the one hand, this ensures the same operating conditions at design-time and at run-time, providing a higher profiling accuracy. On the other hand, it allows the designer to define in *MOST* a minimization objective on the amount of allocated resources.

The synergy between *MOST* and *Barbeque* also enables a simplified low-level profiling, because *MOST* can use in the optimization phase the performance counters exported by the *RTLib*. The target platform used in Section 7.3 provides some hardware counters, including, for example, the number of accesses to the main memory (DRAM). These counters represent platform metrics that are collected by the *RTLib* while the application is running in a *Barbeque* execution context and then can be exported directly to *MOST*, at the end of each simulation, through a standardized XML interface. Therefore, both platform and application-specific metrics can be used by the designer to define the multi-objective optimization problem.

Still in Figure 7.1, the set of Operating Points (OPs) output by the DSE phase is filtered and clustered with respect to platform specific metrics, in order to identify the set of optimal AWMs. The performance level that can be achieved in each AWM – the AWM value – is automatically generated by *MOST*: since it is based on profiling on the target platform, this value is more accurate than any hand-coded value, as done in previous work [19], and allows the *Barbeque* scheduler to take better run-time decisions.

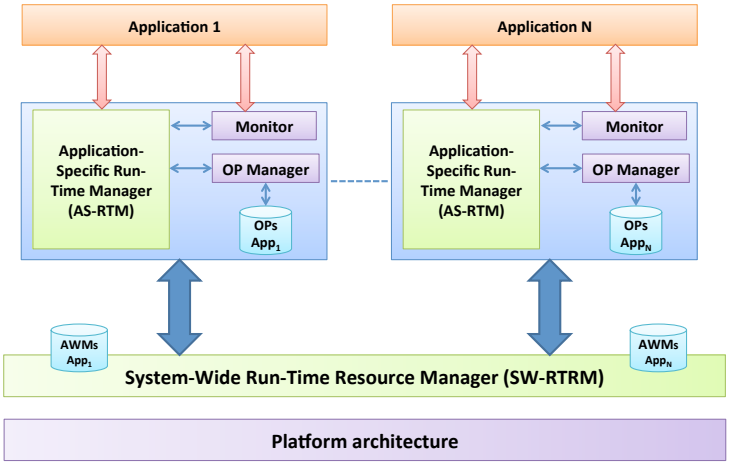


Figure 7.2: *Proposed two-level framework for run-time resource management.*

7.2.2 Run-Time

Similarly to the approach described in Chapter 6, each running application is linked to an AS-RTM, implementing the application auto-tuning layer. The AS-RTM selects the optimal OP to meet the performance requirement (e.g. frame-rate) by tuning the provided Quality of Service (QoS), depending on the amount of available platform resources. However, while in the previous approach each AS-RTM was allowed to take autonomous decisions on resource usage, here resource allocation is delegated to the System-Wide Run-Time Resource Manager (SW-RTRM), as shown in Figure 7.2.

As seen in Section 2.6, *Barbeque* can force an application to use a limited amount of platform resources by means of the Linux *Control Groups*. Each time an event triggers the *Barbeque* scheduler and the running applications are reconfigured by *Barbeque*, the new AWM is sent to the application through the *RTLib* API and the AS-RTM can eventually change the OP. *Barbeque* always monitors the system state, in terms of running applications and resource requirements, thus it can take optimal decisions for priority-based and performance-aware resource allocation.

Moreover, *Barbeque* also knows the underlying platform architecture, in terms of resource heterogeneity and memory organization, and this information is taken into account by the application mapping. The experiments in Section 7.3 allow to analyze this problem on a multi-core x86 NUMA platform: as demonstrated in [22], the performance of applications running on NUMA platforms is significantly affected by cache misses and high access rates to the main memory.

Application resource requirements include both static requirements, defined in the application recipe, and dynamic ones, since each application can assert resource requests at run-time by means of the *SetGoalGap()* API, provided by the *RTLib*. Any call to the *SetGoalGap()* API triggers a rescheduling event, which eventually causes a reallocation of platform resources and the rescheduling on a higher AWM. The parameter passed to this API is the Normalized Actual Penalty (NAP), defined in Section 6.3, which represents a measure of the distance between the application performance requirements and the values profiled at run-time.

In high contention scenarios, the AS-RTM is useful to reduce the rate of such requests, since it allows an application to trade off performance metrics (e.g. frame-rate) with QoS. This enables to account for small performance fluctuations at application-level, by just switching OP within the same AWM, and to reduce the reconfiguration overhead at system-level.

7.3 Experimental Results

Beside the configuration based on the resource-aware AS-RTM (RA-AS-Linux), proposed in Chapter 6, we consider two other configurations that use *Barbeque*: the *Plain-SW-RTRM* and the *AS-SW-RTRM*.

Plain-SW-RTRM: Implementation of the approach proposed in [19], a plain system-wide run-time resource manager (*Barbeque*). Application requirements are mainly defined by the set of Application Working Modes (AWMs), identified at design-time, each one corresponding to a given amount of required resources. However, if the resource requirement gets higher at run-time, an application can also request a higher AWM to the manager, through the *SetGoalGap()* API. In this configuration the AS-RTM is not used, thus the application runs with QoS fixed to 50%.

AS-SW-RTRM: The proposed two-level run-time management framework, which uses both the system-wide run-time resource manager (*Barbeque*) and the AS-RTM. It extends the *Plain-SW-RTRM* configuration by linking each application to an AS-RTM. The AS-RTM is used to control at application-level the trade-off between performance and accuracy metrics, by tuning the parameters orthogonal w.r.t. resource usage. Differently from RA-AS-Linux, in this configuration the number of worker threads is selected according to the AWM assigned by *Barbeque*.

In the first two experiments, we consider the same analysis done in Section 6.4.2 and Section 6.4.3, with sequential and dynamic workload tests, respectively. Then, in Section 7.3.3 we analyze the results from the perspec-

tive of the run-time application requirements, in a mixed-priority scenario. Finally, the experiment in Section 7.3.4 focuses on system-level metrics.

7.3.1 Evaluating RTM Strategies

This section uses the same experimental setup of Section 6.4.2. Four Stereo-Matching instances are executed on the Intel platform (see Section 2.4 for the configuration details), with a throughput goal of 4 frames/s: the applications are started one after the other, with the following start times: $t_{A1} = 0s$, $t_{A2} = 20s$, $t_{A3} = 60s$ and $t_{A4} = 100s$.

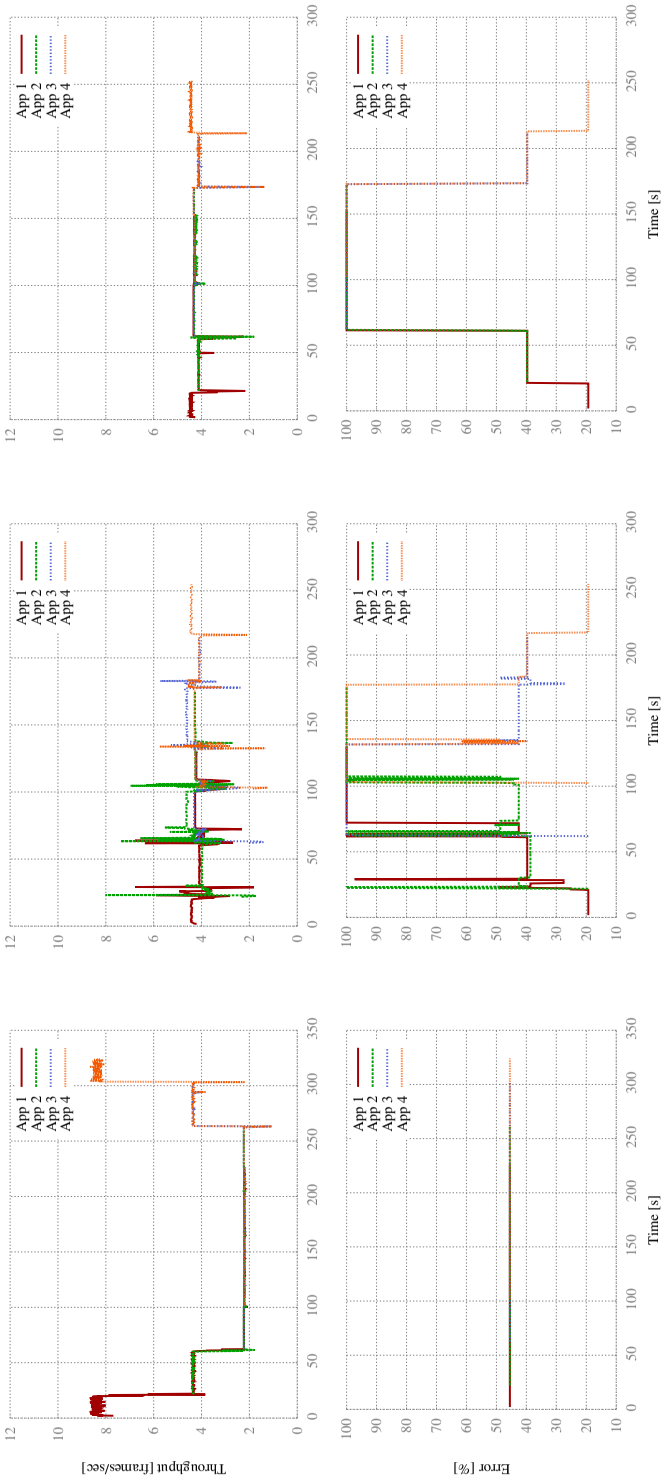
The plot in Figure 7.3a (*Plain-SW-RTRM*) shows the temporal behavior when the applications use a plain *Barbeque* configuration [19], without application-level adaptivity. Whenever the number of running applications changes, *Barbeque* corrects the resource assignment and selects a different AWM for each application. However, since the QoS is fixed (see the horizontal lines in the *Error* plot), any resource reconfiguration has an immediate impact on the profiled throughput.

The plot in Figure 7.3b (*RA-AS-Linux*) is the approach proposed in the previous chapter, where each application lets the AS-RTM to take autonomous decisions for resource allocation. In this case, the throughput goal (4 frames/s) can be met much more frequently by trading off accuracy of results in return for faster execution.

Then, Figure 7.3c shows the behavior of the approach proposed in this chapter (*AS-SW-RTRM*), which combines system-wide RTRM with application-level auto-tuning. In this configuration, two features can be observed:

- The throughput fluctuations whenever the AWM changes is very small, compared to *RA-AS-Linux*.
- The QoS provided by all running applications is the same, meaning that they select the same OP.

Barbeque provides a better isolation of the execution contexts, which results in a more accurate OP selection and a faster response to workload variations: this effect can be observed by comparing the throughput plots of Figure 7.3b and Figure 7.3c, since the throughput fluctuations in *RA-AS-Linux* are much more frequent than in *AS-SW-RTRM*. Moreover, the second feature highlights another important property of the proposed approach: since all running applications are identical and have the same priority in this test, *Barbeque* is fair in allocating platform resources and all applications are configured in the same AWM. Then, the AS-RTM of each application *autonomously* selects the same OP, because all have the same throughput goal.



(a) *Plain-SW-RTRM* **(b)** *RA-AS-Linux* **(c)** *AS-SW-RTRM*

Figure 7.3: *Behavior of the run-time management strategies, in terms of throughput and normalized error, for Plain-Linux (6.3a), AS-Linux (6.3b) and the proposed Resource-Aware AS-Linux (6.3c). The application goal is set to 4 frames/s.*

7.3.2 Dynamic Workload Results

Also in this experiment (see Figure 7.4), the hierarchical Run-Time Management technique (AS-SW-RTRM) shows several advantages with respect to the resource-aware AS-RTM proposed in the previous chapter (RA-AS-Linux) and to the plain Barbeque framework (Plain-SW-RTRM) [19].

From a point of view of the achieved throughput (frame-rate for Stereo-Matching), RA-AS-Linux and AS-SW-RTRM are approximately equivalent in Figure 7.4a, while AS-SW-RTRM is better in Figure 7.4b. However, the metric that mostly benefits from the proposed RTM framework is the frame-rate difference with respect to off-line profiling (the *DEV* metric in Figures 7.4c and Figure 7.4d), since the application performance becomes much more predictable than in RA-AS-Linux.

Both RA-AS-Linux and AS-SW-RTRM perform better than the plain *Barbeque* configuration, because they can benefit from application adaptivity implemented by the AS-RTM: while in Plain-SW-RTRM the application QoS is fixed to 50%, in the other configurations the QoS can get lower or higher than 50%, depending on platform workload and resource availability.

7.3.3 Mixed Priority Analysis

In this sub-section we want to analyze the effectiveness of a priority-based approach to ensure application Quality of Service (QoS) requirements. We bound the QoS to the frame-rate metric monitored at run-time by the Stereo-Matching (SM) AS-RTM with respect to a dynamic goal. In the plots, a positive goal-gap means that the provided frame-rate is above the goal, thus it ensures good QoS to the user.

We consider only the adaptive configuration with *Barbeque* (AS-SW-RTRM), in which we can define a static priority for each application instance. As in previous experiments, whenever the current AWM does not allow SM to provide the demanded QoS (negative goal-gap), the application can send a request to *Barbeque* for an AWM featuring more resources.

We used two instances of SM, one with priority 1 (SM-P1) and the other with priority 2 (SM-P2), where priority 1 is higher than priority 2. We start these two instances at the same time and each SM instance is assigned the highest AWM by *Barbeque*, since the platform is configured with 3 device nodes and each instance is deployed on a different node. In this experiment, we introduced a synthetic application – TestApp – to simulate some unpredictable workload. The TestApp has no functionality. Its only purpose is to increase the resource contention, on the different application priority levels, at pseudo-random time intervals. To this end, the *recipe*

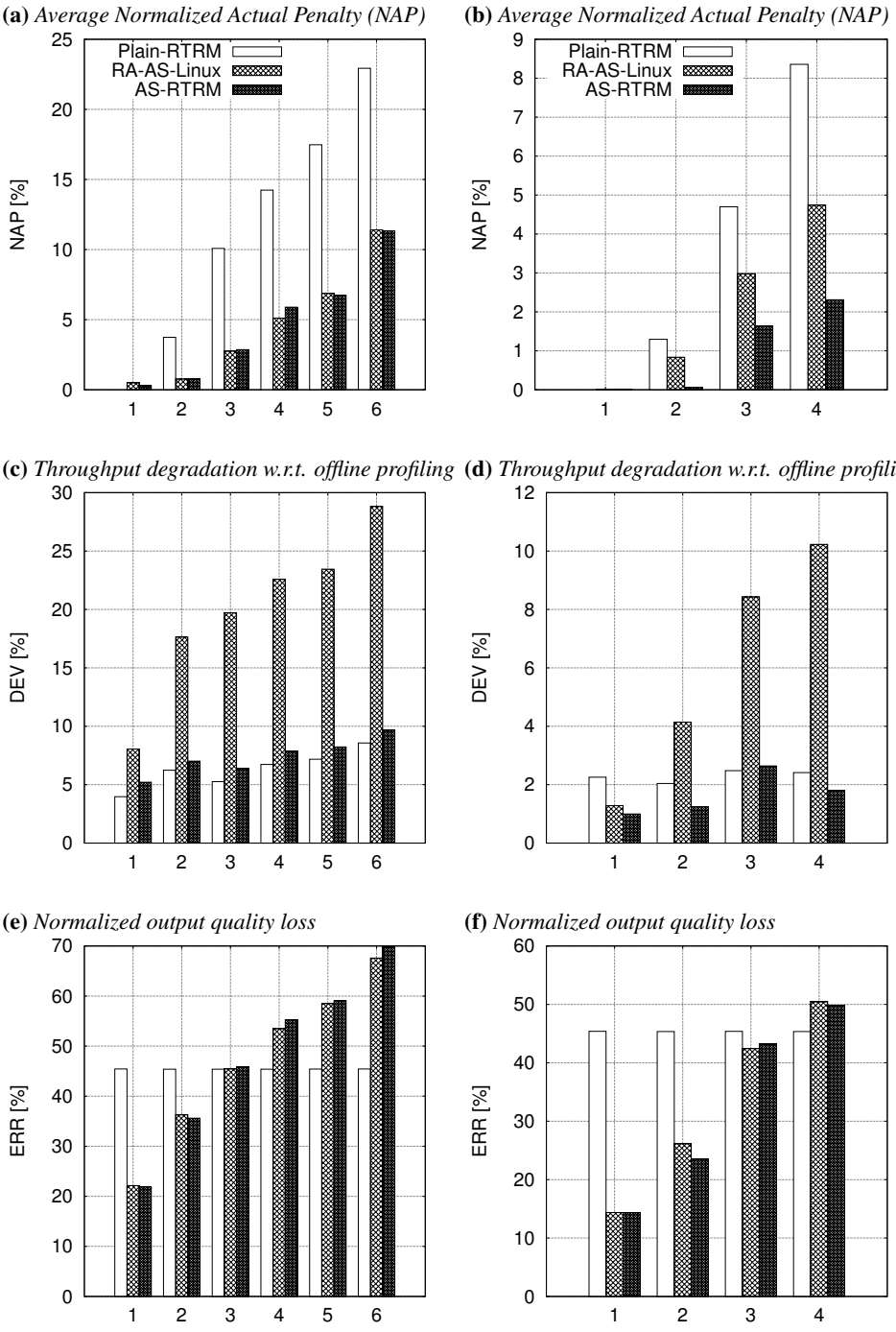


Figure 7.4: *Dynamic workload analysis by varying the number of Stereo-Matching instances, on the AMD platform (7.4a, 7.4c, 7.4e) and Intel platform (7.4b, 7.4d, 7.4f).*

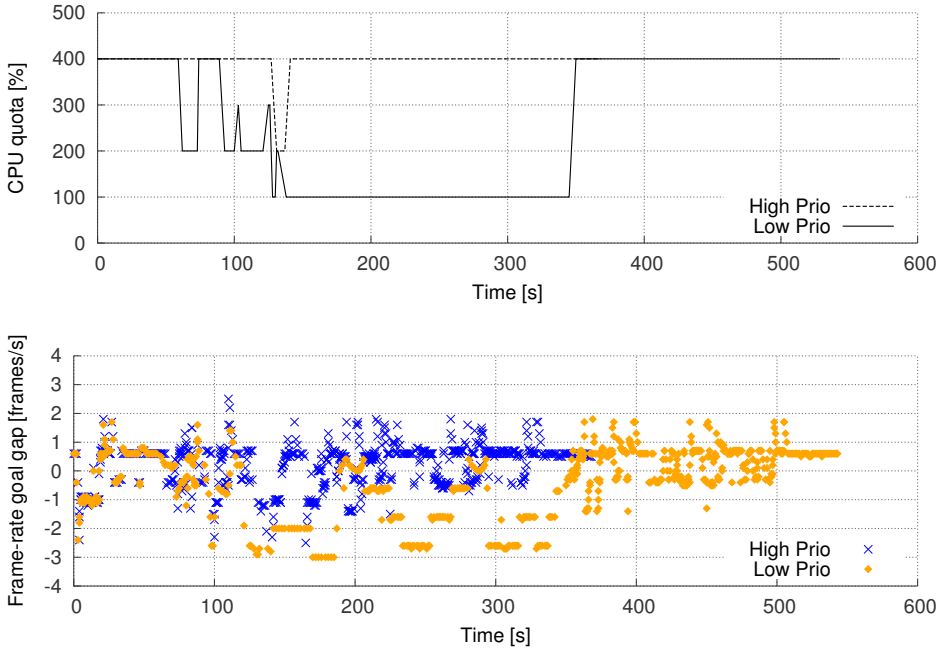


Figure 7.5: *CPU quota and frame-rate goal-gap: SM high priority vs. SM low priority.*

associated to the TestApp includes AWMs with CPU quota requests ranging from 100% to 300%.

We started 3 instances of TestApp, with the following schedule (time elapsed since SM-P1 and SM-P2 started, *t0*):

1. TestApp1 (TA1-P2), with priority 2, after 60s (*t1*)
2. TestApp2 (TA2-P1), with priority 1, after 90s (*t2*)
3. TestApp2 (TA3-P1), with priority 1, after 130s (*t3*)

What happens in this scenario is that *Barbeque* tries to be *fair* in allocating resources to applications with the same priority. When a SM instance asks for more resources, *Barbeque* manages the request according to the priority of the application, the status of the system and its multi-objective optimization policy.

Figure 7.5 shows the CPU quota assigned to SM-P1 and SM-P2. *Barbeque* reserves an entire node (400% CPU quota) to SM-P1, except for a very short time at *t3*, when TA3-P1 starts. On the contrary, SM-P2 is forced by *Barbeque* to run in a lower AWM, which explains why it is assigned only 100% CPU quota until TA2-P1 and TA3-P1 complete their execution. In other words, since there are not enough resources to schedule all applications

Table 7.1: *Performance counters and metrics.*

Counter	Description
CTIME	Workload completion time [s]
POWER	System power consumption [W]
IPC	Effective Instructions-per-Cycles
CYCLES	Total number of CPU cycles
CPU-USED	CPUs utilization
CTX	Total number of context switches
MIG	Total number of CPU migrations
FES	Total number of front-end stalled-cycles
BES	Total number of back-end stalled-cycles
B-RATE	Effective rate of branch instructions
B-MISS-RATE	Effective percentage of missed branches

at the highest AWM, *Barbeque* acquires resources from low-priority applications (TA1-P2 and SM-P2) to ensure the demanded QoS to high-priority applications. As a consequence, the frame-rate goal-gap measured by the AS-RTM of SM-P2 (see Figure 7.5) gets negative at *t2*, which results in a lower QoS.

7.3.4 System-Wide Analysis

This section discusses the effects of the proposed methodology on managing resource contention as well as addressing some platform-specific issues. Three experimental scenarios have been considered by using 1, 3 and 6 instances of the Stereo-Matching application on the AMD platform, and comparing a non adaptive version of this application with the proposed adaptive version. In the first case, every Stereo-Matching instance spawns a fixed number of threads (8), while in the second case the number of threads is dynamically set at runtime, up to a maximum of 8.

The Linux standard `perf` tool has been used to collect architectural performance counters, while the Intelligent Platform Management Interface (IPMI) enabled measurement of the system-wide power consumption. It is worth to underline that the `cpufreq` governor was set to *ondemand*, thus giving complete control of CPU frequencies to the Linux kernel. The AMD NUMA platform has been partitioned into a host (4 cores) to execute the system processes and a managed device (12 cores) to be directly managed by *Barbeque*.

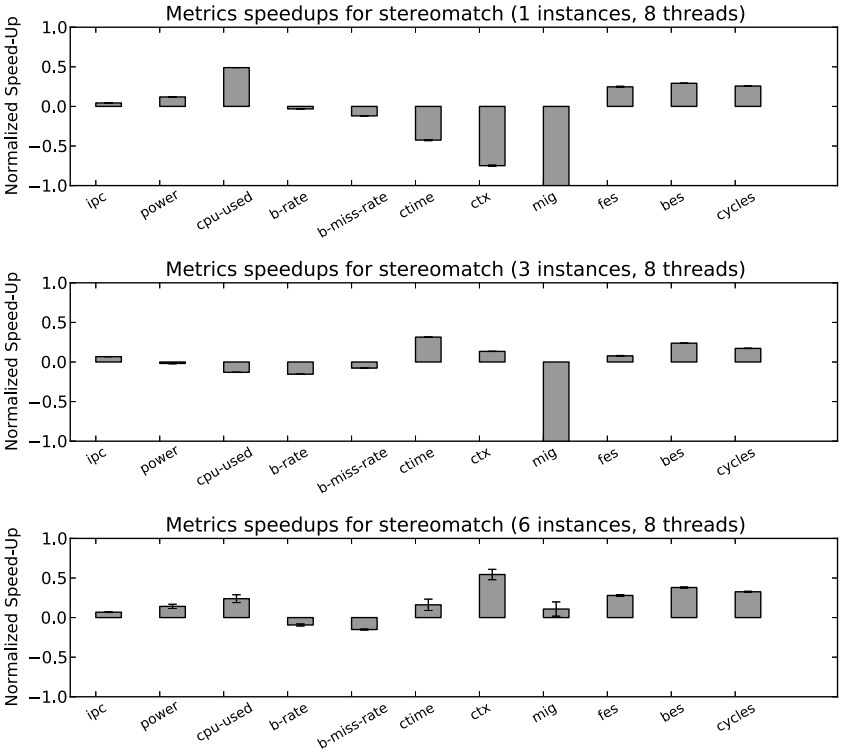


Figure 7.6: *Benefits and loss on the considered performance metrics. Positive bars represent benefits introduced by the proposed methodology.*

For each execution, we sampled the *completion time* of the scenario, the *system power consumption* and a set of statistics provided by the performance counters of the CPUs, listed in Table 7.1. To state the significance of the statistics, we repeated the execution of each scenario 30 times, with a mean confidence interval between 95% and 99%. A scenario stops when all the instances have completed their input stream.

A first analysis has been done, by observing the performance metrics listed in Table 7.1. Figure 7.6 provides an overall picture of the experimental results in terms of normalized metrics speed-up, where positive bars highlight the improvements introduced by the proposed methodology with respect to Linux OS.

The case of a single running instance shows an increase of the completion time of about 40%, but at the same time a reduction of the system power consumption of 12%. This is mainly due to the default *Barbeque* scheduling policy, which allocates resources to the application from a sin-

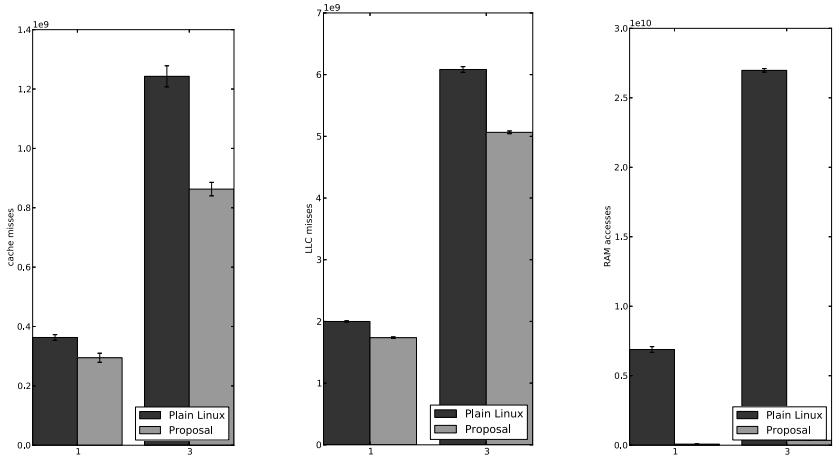


Figure 7.7: *Memory events comparison, plain Linux vs. proposal for single instance and 3-instances scenarios.*

gle processing node. Benefits are higher when the system is subjected to resource contention. In the 3-instances scenario, without noticeable changes in power consumption, the completion time decreases by 35%, while, in the 6-instances scenario, both completion time and power consumption decrease by approximately 15%.

Given the scenarios, one of the key results is that the CPU pipeline works better, being affected by less stalls (both on *front-end* and *back-end*), and less *branch mispredictions*. This results into a lower number of CPU *cycles* spent for the execution, and a slightly higher throughput (*IPC*). The number of *context switches* and *task migrations* gets worse, but since these are intra-node and not inter-node, we do not experience significant penalties. Indeed, the results show that even these drawbacks are strongly mitigated in the presence of higher contention on resources. Looking at the most critical scenario (6-instances), it is evident that almost all metrics have been improved. In the case of multiple Stereo-Matching instances, the results point out the effectiveness of the proposed methodology in managing the resource contention, and thus improving the utilization of resources by each competing application.

To complete the analysis, we focused on the memory events to evaluate the benefits in overcoming platform-specific critical issues. Indeed, as demonstrated in [22], the performance of applications running on NUMA platforms is significantly affected by cache misses and high access rates to the main memory. These issues are exacerbated as the contention on resources increases, when multiple applications run on the system.

Table 7.2: *Performance metrics on memory.*

Counter	Description
CACHE-MISSES	Total number of cache misses
LLC-MISSES	Total number Last-level cache misses
DRAM	Total number of DRAM accesses

To address these platform specific issues, we exploited the synergy between *MOST*, *AS-RTM* and *Barbeque* by a) including the minimization of DRAM accesses in the DSE multi-objective optimization, and b) allocating resources at run-time, i.e. scheduling the applications, taking into account the memory hierarchy of the target NUMA machine (where the DRAM is logically shared and Last-Level Caches (LLCs) are physically distributed among nodes). The results in Figure 7.7 show that the number of *cache misses* (last level included, LLC, see Table 7.2) has been reduced, as well as the number of accesses to the main memory (DRAM). The latter is the most interesting result, because contention on the memory controller represents the main performance bottleneck on NUMA architectures. The importance of this result demonstrates the capabilities of our methodology to face platform-specific weaknesses.

7.4 Conclusions

In this chapter a design methodology for run-time management on multi-core platforms was presented. Design Space Exploration (DSE) identifies a set of Operating Points and Application Working Modes. Then at run-time, the SW-RTRM exploits multi-objective resource allocation policies, while the AS-RTM fine tunes the application parameters according to time-varying performance or QoS goals, within the assigned resource quota.

The experiments proved the benefits of the proposed approach by considering an adaptive Stereo-Matching application, running on a multi-core NUMA platform for a target scenario featuring resource contention and mixed-priority workload. From the application perspective, the resource contention analysis has shown the benefits of our approach in terms of average performance (frame-rate) of the running Stereo-Matching instances. Moreover, the mixed-workload analysis has demonstrated that applications with different priority levels are properly served, ensuring the requested QoS to high-priority applications while keeping under control low-priority workload.

From the system side, the outcome of the low-level analysis has shown performance improvements ranging from 35% to 40% and a power consumption reduction between 12% and 15% for the selected set of experiments. Based on the statistics collected from several performance counters, we can assess that these results are significant for a better utilization of the system resources, compared to the *plain-Linux* case. Finally, the hierarchical and modular nature of the approach makes it promising for scaling with the application performance requirements and platform computational capabilities, whatever the target market will be, embedded systems or HPC. The approach has also been validated on the STHORM embedded platform by STMicroelectronics in the context of the 2PARMA European project [9].

CHAPTER 8

Conclusions

This thesis has addressed the problem of optimizing OpenCL applications for a target heterogeneous parallel platform. A design-time phase, presented in Part I, exploits different Design Space Exploration (DSE) techniques for efficient customization of a parametric OpenCL application for the target platform, considering both device-specific constraints and concurrency in the application task graph. Then, the framework proposed in Part II enables effective application auto-tuning at run-time, with flavors of autonomic computing, while a centralized resource manager allows for optimal resource allocation under varying workloads and dynamic application requirements.

All experiments have been carried out on industrial platforms, in order to demonstrate the applicability of the proposed methodology to real case studies. Also the target application – a stereo-matching OpenCL implementation – represents an example of computationally intensive multimedia application, that can benefit from acceleration on a parallel computing platform. Indeed, some of the outcomes of this thesis have been used within the 2PARMA FP7 European project and implemented in official prototypes delivered to the project consortium [9, 10, 11, 12].

The applicability to real case studies is motivated by the rapidly growing adoption of the OpenCL programming API by many platform vendors. Thus,

the problem of application optimization for parallel heterogeneous platforms – addressed by this thesis – represents a real challenge today. The same way compilers allow to automate several kinds of code optimization, application domain experts need design methodologies and tools for automated optimization and deployment of their applications on heterogeneous parallel platforms. This problem is exacerbated by the rapid evolution of commercial platforms, which requires fast and effective porting of applications to new hardware with tight time-to-market.

The choice of a stereo-matching application, in turn, is coherent with the main application domain targeted by OpenCL today, namely computer vision. As an example, the OpenCV library [24], widely used to provide vision algorithms on embedded systems, has recently added an OpenCL back-end. However, other application domains are expected to benefit from software-programmable parallel accelerators, for example in the field of digital signal processing, as demonstrated by the adoption of the OpenCL API by Texas Instruments[1]. Moreover, the possibility to "program" even FPGA platforms through the OpenCL API allows to embrace the problem of High-Level Synthesis (HLS) and that of hardware-software partitioning, by just treating the hardware partition as an OpenCL context on FPGA and analyzing task mapping within the same programming framework.

In this context, the design and optimization methodology proposed in this thesis enables portability of application code along with platform-specific performance optimization. Besides, the run-time management layer implements effective application auto-tuning and resource allocation, in order to improve the system performance in multi-application scenarios.

8.1 Design-Time Conclusions

The integration with the MOST framework allows to automate the DSE process and to implement advanced exploration strategies. The DSE techniques presented in Part I aim at speeding up the exploration with respect to a full combinatorial search of all parameters, while ensuring better results than design optimization done by hand in a limited time window.

Chapter 3 describes a parametric OpenCL application design and a methodology for customizing application parameters and identifying a set of Pareto-optimal Operating Points (OPs). This methodology allows to sample the design space by first considering the application parameters subject to platform-specific constraints, such as the size of local memory on an OpenCL accelerator. This custom Design of Experiment (DoE), together

[1] See TI wiki webpage: http://processors.wiki.ti.com/index.php/OpenCL_User's_Guide

with a precise selection of the dataset by means of a correlation analysis, allows to explore just a small fraction of the design space and provides, at the same time, a very accurate Pareto solution. Combined with a high-level simulation model in the final refinement phase, it provides a significant speed-up, reducing the overall exploration time by 16x while keeping the approximation error of the solution below 3%.

Chapter 4 exploits low-level and high-level simulations too, but in the training set of a Response Surface Model (RSM) based on Artificial Neural Networks (ANNs). The proposed ensemble neural network model improves prediction accuracy; alternatively, the same level of accuracy can be achieved by replacing part of the training set with high-level samples, in return for 10x speed-up of the training phase.

Finally, Chapter 5 analyzes the problem of task mapping on a heterogeneous OpenCL platform. An auto-tuning phase based on a constraint solver allows to prune the design space of the target application down to 0.1% of the original size, by exploiting platform constraints. Combined with a constraint problem formulation to analytically find an efficient task mapping, the proposed methodology generates optimized OpenCL software pipelines that exploit the concurrency in a task graph on multiple accelerators.

8.2 Run-Time Conclusions

The set of configurations identified at design-time is used by a run-time management framework to support resource allocation and application auto-tuning. Indeed, the Pareto-optimal configurations present different trade-offs between performance (e.g. frame-rate), accuracy of the result and resource requirement. The proposed framework implements the logic for selection of the best run-time configuration, while optimizing resource allocation based on dynamic application requirements. This allows to cope with workload variations and resource sharing in multi-application scenarios on the one hand, with thermal and variability platform issues on the other hand.

In Chapter 6 I presented an application layer for light-weight Run-Time Management (RTM) – the Application-Specific Run-Time Manager (AS-RTM) – which exploits platform sensing for dynamic control of resource utilization. The AS-RTM is generic, however its behavior can be customized for each application given a different list of OPs. While enabling 50% better average performance with respect to a plain Linux configuration, this approach has the advantage of being non-intrusive from a design point of view, since applications take run-time decisions autonomously and no communication infrastructure is needed.

Finally, in Chapter 7 the AS-RTM is integrated with *Barbeque*, to improve performance predictability and some system-wide metrics, such as fairness of resource allocation and priority-based Quality of Service (QoS). In this approach, *Barbeque* accounts for system-wide optimization – mainly resource allocation, based on design-time characterization of the Application Working Modes (AWMs) – while the AS-RTM fine-tunes the application behavior to react to small workload variations and dynamic requirements. The resulting two-level RTM approach has been validated on the STHORM embedded platform as well as on a NUMA x86 workstation and represents one of the main outcomes of the 2PARMA project.

8.3 Future Works

The techniques proposed in this thesis could be further developed in several ways or re-targeted to other design problems, as listed below.

Custom code generation. The proposed optimization methodology requires an application parametric design, customizable by means of either command line options or constants defined during dynamic compilation of OpenCL kernels. This does not represent a limitation of the approach, since it is common practice to use a parametric design when OpenCL applications are optimized by hand. An alternative approach exploits dynamic generation of custom code, such as in [77]. The limitation of custom code generation is that the application is usually described with a Domain Specific Language (DSL), so this approach is domain specific. However, it also enables several kinds of code optimization, by exploiting some recurrent computational patterns. Thus, a possible extension of this work consists of starting from a DSL representation of the application, rather than from a parametric design, and include in the design space also code transformations for generation of an optimized implementation for a target platform.

Co-scheduling profiling support. Another important aspect of this work is run-time management. The DSE methodology presented in Chapter 7 exploits platform performance counters to consider also the application memory bandwidth in the multi-objective optimization problem. This knowledge-base is needed by the run-time resource manager for co-scheduling decisions, as in the Co-Scheduling Workloads (CoWs) scheduler policy recently added to *Barbeque* [78]. Thus, the exploration methodology presented in this thesis could be extended to identify and export a set of alternative application configurations, with different trade-offs between compute-intensive and memory-bound profiles.

OpenCL-enabled FPGA platforms. The OpenCL SDK recently released by Altera allows prototyping and synthesis of an FPGA-based heterogeneous system (CPU + FPGA). For this platform, the OpenCL programming paradigm allows to embrace both High-Level Synthesis (HLS) and application portability. However, optimization of the OpenCL kernels is much more challenging than for general-purpose platforms, since logic- and gate-level optimizations are difficult to capture in a high-level input design. While for code optimization the designer still has to rely on the synthesis tools provided by the platform vendor, the methodology presented in this thesis could effectively support the task mapping phase. OpenCL-enabled FPGAs are usually provided with a PCI interface[2], in order to be used as co-processors for application acceleration. Thus, they could be found on general-purpose workstations beside other accelerators, such as GPGPUs or the Intel Xeon Phi[3]. For such heterogeneous platforms, mapping of an application task graph on the available devices represents a large design space but the mapping options are limited by area and timing constraints on FPGAs. Thus, a proper analytical technique, based on a *constraint solver* as proposed in Chapter 5, could help pruning the design space from unfeasible or sub-optimal configurations.

Mixed-criticality systems. The mapping technique and run-time framework presented in this thesis are being improved to support the design of mixed-criticality systems in the CONTREX FP7 European project[4]. Deployment of *hard real-time* and *soft real-time* applications on the same multi-core platform requires design methodologies to find the correct – and preferably optimal – task mapping and scheduling. On the one hand, analytical techniques are needed to analyze the Worst-Case Execution Time (WCET) of hard real-time applications; on the other hand, profiled-based DSE and run-time management, based on the techniques presented in this thesis, are expected to improve the overall performance of low-criticality applications with respect to soft timing constraints.

[2] See for example the Altera Stratix V FPGA computing card, which has a PCI Express interface to the host CPU: http://www.nallatech.com/PCI-Express-FPGA-Cards/pcie-385n-altera-stratix-v-fpga-computing-card.html

[3] Intel Xeon Phi website: http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html

[4] CONTREX website: http://contrex.offis.de

Abbreviations

A

ADRS	Average Distance from Reference Set
AEM	Abstract Execution Model
ANN	Artificial Neural Network
API	Application Programming Interface
AS-RTM	Application-Specific Run-Time Manager
ATLAS	Automatically Tuned Linear Algebra Software
AWM	Application Working Mode

C

CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture

D

DMA	Direct Memory Access
DoE	Design of Experiment
DRAM	Dynamic Random Access Memory
DSE	Design Space Exploration
DSL	Domain Specific Language
DSP	Digital Signal Processor

E

EDA	Electronic Design Automation

F

FFT	Fast Fourier Transform
FMA	Fused Multiply-Add
FPGA	Field Programmable Gate Array

G

GALS	Globally Asynchronous Locally Synchronous
GCD	Grand Central Dispatch
GPGPU	General Purpose GPU
GPU	Graphics Processing Unit

H

HLS	High-Level Synthesis
HPC	High Performance Computing

I

IP	Intellectual Property
IPMI	Intelligent Platform Management Interface
ISS	Instruction Set Simulator

L

LLC	Last-Level Cache

M

MEMS	Micro Electro-Mechanical System
MIMD	Multiple Instruction Multiple Data
MMMKP	Multi-choice Multi-dimension Multiple Knapsack Problem
MPSoC	Multi-Processor System-on-Chip

N

NAP	Normalized Actual Penalty
NoC	Network-on-Chip
NUMA	Non-Uniform Memory Access

O

OP	Operating Point
OpenCL	Open Computing Language

OS	Operating System
P	
PCI	Peripheral Component Interconnect
PE	Processing Element
PIL	Platform Integration Layer
Q	
QoS	Quality of Service
R	
RBF	Radial Basis Function
RISC	Reduced Instruction Set Computer
RMS	Root Mean Square
RSD	Relative Standard Deviation
RSM	Response Surface Model
RTM	Run-Time Management
RTRM	Run-Time Resource Management
S	
SDK	Software Development Kit
SDR	Software Defined Radio
SMP	Symmetric Multi-Processing
SPMD	Single Program Multiple Data
SW-RTRM	System-Wide Run-Time Resource Manager
T	
TBB	Threading Building Blocks
TCDM	Tightly Coupled Data Memory
W	
WCET	Worst-Case Execution Time

Author's Publication List

International Conferences

[1] **Edoardo Paone**, Francesco Robino, Gianluca Palermo, Vittorio Zaccaria, Ingo Sander, and Cristina Silvano. Customization of OpenCL Applications for Efficient Task Mapping under Heterogeneous Platform Constraints. Accepted for publication in *Proceedings of the Conference on Design, Automation and Test in Europe – DATE'15*.

[2] Davide Gadioli, Simone Libutti, Giuseppe Massari, **Edoardo Paone**, Michele Scandale, Patrick Bellasi, Gianluca Palermo, Vittorio Zaccaria, Giovanni Agosta, William Fornaciari, and Cristina Silvano. OpenCL Application Auto-Tuning and Run-Time Resource Management for Multi-Core Platforms. In *Proceedings of the 12th International Symposium on Parallel and Distributed Processing with Applications – ISPA'14*, pages 127–133. IEEE, August 2014.

[3] Giuseppe Massari, **Edoardo Paone**, Patrick Bellasi, Gianluca Palermo, Vittorio Zaccaria, William Fornaciari, and Cristina Silvano. Combining Application Adaptivity and System-wide Resource Management on Multi-Core Platforms. In *Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation – SAMOS'14*, pages 26–33. IEEE, July 2014.

[4] **Edoardo Paone**, Davide Gadioli, Gianluca Palermo, Vittorio Zaccaria, and Cristina Silvano. Evaluating Orthogonality between Application Auto-Tuning and Run-Time Resource Management for Adaptive OpenCL Applications. In *Proceedings of the 25th International*

Conference on Application-specific Systems, Architectures and Processors – ASAP'14, pages 161–168. IEEE, June 2014.

[5] Giuseppe Massari, **Edoardo Paone**, Michele Scandale, Patrick Bellasi, Gianluca Palermo, Vittorio Zaccaria, Giovanni Agosta, William Fornaciari, and Cristina Silvano. Data Parallel Application Adaptivity and System-Wide Resource Management in Many-Core Architectures. In *Reconfigurable Computing: Architectures, Tools, and Applications*, volume 8405 of *Lecture Notes in Computer Science*, pages 345–352. Springer International Publishing, April 2014.

[6] **Edoardo Paone**, Nazanin Vahabi, Vittorio Zaccaria, Cristina Silvano, Diego Melpignano, Germain Haugou, and Thierry Lepley. Improving Simulation Speed and Accuracy for Many-Core Embedded Platforms with Ensemble Models. In *Proceedings of the Conference on Design, Automation and Test in Europe – DATE'13*, pages 671–676. IEEE, March 2013.

[7] **Edoardo Paone**, Gianluca Palermo, Vittorio Zaccaria, Cristina Silvano, Diego Melpignano, Germain Haugou, and Thierry Lepley. An Exploration Methodology for a Customizable OpenCL Stereo-matching Application Targeted to an Industrial Multi-Cluster Architecture. In *Proceedings of the 8th International Conference on Hardware/Software Codesign and System Synthesis – CODES+ISSS'12*, pages 503–512. ACM, October 2012.

Technical Reports

[8] Giovanni Agosta, William Fornaciari, Cristina Silvano, Vittorio Zaccaria, Patrick Bellasi, **Edoardo Paone**, Michele Scandale, Dimitrios Soudris, Benno Stabernack, Daniel Guenther, Xi Zhang, Junaid Ansari, Diego Melpignano, Germain Haugou, and Antoine Dejonghe. 2PARMA Lesson Learnt. Deliverable D7.7.1, FP7 European Project 2PARMA, 2013.

[9] Germain Haugou, Jens Brandenburg, and **Edoardo Paone**. Integrated prototype based on STM P2012 architecture. Deliverable D5.3.1, FP7 European Project 2PARMA, 2013.

[10] Vittorio Zaccaria, **Edoardo Paone**, Sotirios Xydis, Patrick Bellasi, Jens Brandenburg, Alex Bartzas, and Ioannis Koutras. Integrated prototype based on x86 multi-core architecture. Deliverable D5.4.1, FP7 European Project 2PARMA, 2013.

[11] **Edoardo Paone** and Vittorio Zaccaria. Final prototype of the design space exploration methodologies for run-time support. Deliverable D3.3.2, FP7 European Project 2PARMA, 2012.

[12] Jens Brandenburg, **Edoardo Paone**, Vittorio Zaccaria, Cristina Silvano, Gianluca Palermo, Germain Haugou, J. M. Zins, Diego Melpignano, Xi Zhang, and Junaid Ansari. Run-time monitoring of application parameters. Deliverable D4.2.3, FP7 European Project 2PARMA, 2012.

Bibliography

[13] W.B. Ackerman. Data Flow Languages. *Computer*, 15(2):15–25, Feb 1982.

[14] Apple Inc. Using Grand Central Dispatch With OpenCL, 2013. https://developer.apple.com/library/mac/documentation/Performance/Conceptual/OpenCL_MacProgGuide/UsingGCDwOpenCL/UsingGCDwOpenCL.html.

[15] A.H. Ashouri, V. Zaccaria, S. Xydis, G. Palermo, and C. Silvano. A framework for Compiler Level statistical analysis over customized VLIW architecture. In *IFIP/IEEE 21st International Conference on Very Large Scale Integration (VLSI-SoC), 2013*, pages 124–129, October 2013.

[16] Cédric Augonnet, Samuel Thibault, Raymond Namyst, and Pierre-André Wacrenier. StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures. *Concurrency and Computation: Practice & Experience*, 23(2):187–198, February 2011.

[17] Subhasis Banerjee, G Surendra, and SK Nandy. On the effectiveness of phase based regression models to trade power and performance using dynamic processor adaptation. *Journal of Systems Architecture*, 54(8):797–815, 2008.

[18] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the Art of Virtualization. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, pages 164–177, New York, NY, USA, 2003. ACM.

[19] Patrick Bellasi, Giuseppe Massari, and William Fornaciari. A RTRM proposal for multi/many-core platforms and reconfigurable applications. In *7th International Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2012*, pages 1–8, July 2012.

[20] Patrick Bellasi, Giuseppe Massari, and William Fornaciari. Exploiting Linux Control Groups for Effective Run-time Resource Management. In *PARMA 2013 Workshop HiPEAC 2013, Jan 2013, Berlin, Germany*, 2013.

[21] Luca Benini, Eric Flamand, Didier Fuin, and Diego Melpignano. P2012: Building an Ecosystem for a Scalable, Modular and High-efficiency Embedded Computing Accelerator. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '12, pages 983–987, San Jose, CA, USA, 2012. EDA Consortium.

[22] Sergey Blagodurov, Sergey Zhuravlev, Alexandra Fedorova, and Ali Kamali. A Case for NUMA-aware Contention Management on Multicore Systems. In *Proceedings of the 19th*

International Conference on Parallel Architectures and Compilation Techniques, PACT '10, pages 557–558, New York, NY, USA, 2010. ACM.

[23] A. Bonfietti, L. Benini, M. Lombardi, and M. Milano. An efficient and complete approach for throughput-maximal SDF allocation and scheduling on multi-core platforms. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, pages 897–902, March 2010.

[24] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[25] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[26] Kevin J. Brown, Arvind K. Sujeeth, Hyouk Joong Lee, Tiark Rompf, Hassan Chafi, Martin Odersky, and Kunle Olukotun. A Heterogeneous Parallel Framework for Domain-Specific Languages. In *Proceedings of the 2011 International Conference on Parallel Architectures and Compilation Techniques*, PACT '11, pages 89–100, Washington, DC, USA, 2011. IEEE Computer Society.

[27] Hans-Joachim Bungartz, Christoph Riesinger, Martin Schreiber, Gregor Snelting, and Andreas Zwinkau. Invasive Computing in HPC with X10. In *Proceedings of the third ACM SIGPLAN X10 Workshop*, pages 12–19. ACM, 2013.

[28] Manuel M.T. Chakravarty, Gabriele Keller, Sean Lee, Trevor L. McDonell, and Vinod Grover. Accelerating Haskell Array Codes with Multicore GPUs. In *Proceedings of the Sixth Workshop on Declarative Aspects of Multicore Programming*, DAMP '11, pages 3–14, New York, NY, USA, 2011. ACM.

[29] Koen Claessen, Mary Sheeran, and Bo Joel Svensson. Expressive Array Constructs in an Embedded GPU Kernel Programming Language. In *Proceedings of the 7th Workshop on Declarative Aspects and Applications of Multicore Programming*, DAMP '12, pages 21–30, New York, NY, USA, 2012. ACM.

[30] Yann Collette and Patrick Siarry. *Multiobjective Optimization: Principles and Case Studies*. Springer, 2003.

[31] Matthew Curtis-Maury, Filip Blagojevic, Christos D. Antonopoulos, and Dimitrios S. Nikolopoulos. Prediction-Based Power-Performance Adaptation of Multithreaded Scientific Codes. *IEEE Transactions on Parallel and Distributed Systems*, 19(10):1396–1410, October 2008.

[32] Piotr Czyżżak and Adrezej Jaszkiewicz. Pareto simulated annealing – a metaheuristic technique for multiple-objective combinatorial optimization. *Journal of Multi-Criteria Decision Analysis*, 7(1):34–47, 1998.

[33] Usman Dastgeer, Johan Enmyren, and Christoph W. Kessler. Auto-tuning SkePU: A Multi-backend Skeleton Programming Framework for multi-GPU Systems. In *Proceedings of the 4th International Workshop on Multicore Software Engineering*, IWMSE '11, pages 25–32, New York, NY, USA, 2011. ACM.

[34] K. Datta, M. Murphy, V. Volkov, S. Williams, J. Carter, L. Oliker, D. Patterson, J. Shalf, and K. Yelick. Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures. In *International Conference for High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008.*, pages 1–12. IEEE Press, November 2008.

[35] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, April 2002.

[36] Denis Demidov, Karsten Ahnert, Karl Rupp, and Peter Gottschling. Programming CUDA and OpenCL: A Case Study Using Modern C++ Libraries. *Computing Research Repository (CoRR)*, abs/1212.6326, 2012.

[37] Zachary DeVito, James Hegarty, Alex Aiken, Pat Hanrahan, and Jan Vitek. Terra: A multi-stage language for high-performance computing. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '13, pages 105–116, New York, NY, USA, 2013. ACM.

[38] Yuri Dotsenko, Sara S. Baghsorkhi, Brandon Lloyd, and Naga K. Govindaraju. Auto-tuning of Fast Fourier Transform on Graphics Processors. In *Proceedings of the 16th ACM Symposium on Principles and Practice of Parallel Programming*, PPoPP '11, pages 257–266, New York, NY, USA, 2011. ACM.

[39] Alejandro Duran, Eduard Ayguadé, Rosa M Badia, Jesús Labarta, Luis Martinell, Xavier Martorell, and Judit Planas. OmpSs: a Proposal for Programming Heterogeneous Multi-Core Architectures. *Parallel Processing Letters*, 21(02):173–193, 2011.

[40] Marc Duranton, David Black-Schaffer, Koen De Bosschere, and Jonas Maebe. The HiPEAC Vision for Advanced Computing in Horizon 2020. 2013.

[41] M. T.M. Emmerich, K. C. Giannakoglou, and B. Naujoks. Single- and Multiobjective Evolutionary Optimization Assisted by Gaussian Random Field Metamodels. *IEEE Transactions on Evolutionary Computation*, 10(4):421–439, August 2006.

[42] Holger Endt and Kay Weckemann. Remote Utilization of OpenCL for Flexible Computation Offloading using Embedded ECUs, CE Devices and Cloud Servers. In *PARCO*, pages 133–140, 2011.

[43] Michael Flynn. *Flynn's Taxonomy*, pages 689–697. Springer US, 2011.

[44] Yoav Freund, Robert E Schapire, et al. Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference Machine Learning, 1996*, volume 96 of *ICML*, pages 148–156, 1996.

[45] M. Frigo and S.G. Johnson. The Design and Implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, February 2005.

[46] Benedict Gaster, Lee Howes, David R Kaeli, Perhaad Mistry, and Dana Schaa. *Heterogeneous Computing with OpenCL: Revised OpenCL 1.2*. Morgan Kaufmann, 2012.

[47] Kourosh Gharachorloo, Daniel Lenoski, James Laudon, Phillip Gibbons, Anoop Gupta, and John Hennessy. Memory Consistency and Event Ordering in Scalable Shared-memory Multiprocessors. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, ISCA '90, pages 15–26, New York, NY, USA, 1990. ACM.

[48] Tony Givargis, Frank Vahid, and Jörg Henkel. System-level Exploration for Pareto-optimal Configurations in Parameterized Systems-on-a-chip. In *Proceedings of the 2001 IEEE/ACM International Conference on Computer-aided Design*, ICCAD '01, pages 25–30, Piscataway, NJ, USA, 2001. IEEE Press.

[49] Dominik Grewe and Michael F. P. O'Boyle. A Static Task Partitioning Approach for Heterogeneous Systems Using OpenCL. In *Proceedings of the 20th International Conference on Compiler Construction: Part of the Joint European Conferences on Theory and Practice of Software*, CC'11/ETAPS'11, pages 286–305, Berlin, Heidelberg, 2011. Springer-Verlag.

[50] Owen Harrison and John Waldron. Practical Symmetric Key Cryptography on Modern Graphics Hardware. In *USENIX Security Symposium*, pages 195–210, 2008.

[51] John L Hennessy and David A Patterson. *Computer Architecture: A Quantitative Approach*. Elsevier, 2012.

[52] Sylvain Henry, Alexandre Denis, Denis Barthou, Marie-Christine Counilh, and Raymond Namyst. Toward OpenCL Automatic Multi-Device Support. In *Euro-Par 2014 – 20th International Conference on Parallel Processing*, pages 776–787. Springer, 2014.

[53] Fernando Herrera and Ingo Sander. Combining analytical and simulation-based design space exploration for time-critical systems. In *Forum on Specification Design Languages (FDL), 2013*, pages 1–8, Sept 2013.

[54] Henry Hoffmann, Jonathan Eastep, Marco D. Santambrogio, Jason E. Miller, and Anant Agarwal. Application Heartbeats: A Generic Interface for Specifying Program Performance and Goals in Autonomous Computing Environments. In *Proceedings of the 7th International Conference on Autonomic Computing*, ICAC '10, pages 79–88, New York, NY, USA, 2010. ACM.

[55] Henry Hoffmann, Jim Holt, George Kurian, Eric Lau, Martina Maggio, Jason E. Miller, Sabrina M. Neuman, Mahmut Sinangil, Yildiz Sinangil, Anant Agarwal, Anantha P. Chandrakasan, and Srinivas Devadas. Self-aware Computing in the Angstrom Processor. In *Proceedings of the 49th Annual Design Automation Conference*, DAC '12, pages 259–264, New York, NY, USA, 2012. ACM.

[56] Henry Hoffmann, Stelios Sidiroglou, Michael Carbin, Sasa Misailovic, Anant Agarwal, and Martin Rinard. Dynamic Knobs for Responsive Power-aware Computing. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVI, pages 199–212, New York, NY, USA, 2011. ACM.

[57] Ching Lai Hwang, Abu Syed Md Masud, et al. *Multiple Objective Decision Making – Methods and Applications*, volume 164. Springer, 1979.

[58] Intel Corporation. Intel Software Autotuning Tool, 2010. https://software.intel.com/en-us/articles/intel-software-autotuning-tool.

[59] Intel Corporation. General Matrix Multiply Sample, 2013. https://software.intel.com/sites/products/vcsource/files/GEMM.pdf.

[60] Gangwon Jo, Won Jong Jeon, Wookeun Jung, Gordon Taft, and Jaejin Lee. OpenCL Framework for ARM Processors with NEON Support. In *Proceedings of the 2014 Workshop on Programming Models for SIMD/Vector Processing*, WPMVP '14, pages 33–40, New York, NY, USA, 2014. ACM.

[61] P. J. Joseph, K. Vaswani, and Matthew J. Thazhuthaveetil. A Predictive Performance Model for Superscalar Processors. In *39th Annual IEEE/ACM International Symposium on Microarchitecture, 2006. MICRO-39.*, pages 161–170, Dec 2006.

[62] S. Kamil, C. Chan, L. Oliker, J. Shalf, and S. Williams. An auto-tuning framework for parallel multicore stencil computations. In *IEEE International Symposium on Parallel Distributed Processing (IPDPS), 2010*, pages 1–12. IEEE Press, April 2010.

[63] Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack problems*. Springer, 2004.

[64] Torsten Kempf, Gerd Ascheid, and Rainer Leupers. *Multiprocessor Systems on Chip: Design Space Exploration*. Springer, 2011.

[65] Jeffrey O. Kephart and David M. Chess. The Vision of Autonomic Computing. *Computer*, 36(1):41–50, January 2003.

[66] K. Keutzer, A.R. Newton, J.M. Rabaey, and A. Sangiovanni-Vincentelli. System-level design: orthogonalization of concerns and platform-based design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(12):1523–1543, December 2000.

[67] Khronos Group. OpenCL Specification, v1.2. https://www.khronos.org/registry/cl/specs/opencl-1.2.pdf, 2012.

[68] Jungwon Kim, Sangmin Seo, Jun Lee, Jeongho Nah, Gangwon Jo, and Jaejin Lee. SnuCL: An OpenCL Framework for Heterogeneous CPU/GPU Clusters. In *Proceedings of the 26th ACM International Conference on Supercomputing*, ICS '12, pages 341–352, New York, NY, USA, 2012. ACM.

[69] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. seL4: Formal Verification of an OS Kernel. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, SOSP '09, pages 207–220, New York, NY, USA, 2009. ACM.

[70] Joshua Knowles. ParEGO: A hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, 10(1):50–66, 2006.

[71] Sebastian Kobbe, Lars Bauer, Daniel Lohmann, Wolfgang Schröder-Preikschat, and Jörg Henkel. DistRM: Distributed Resource Management for On-chip Many-core Systems. In *Proceedings of the Seventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, CODES+ISSS '11, pages 119–128, New York, NY, USA, 2011. ACM.

[72] Bradford Larsen. Simple Optimizations for an Applicative Array Language for Graphics Processors. In *Proceedings of the Sixth Workshop on Declarative Aspects of Multicore Programming*, DAMP '11, pages 25–34, New York, NY, USA, 2011. ACM.

[73] Benjamin C. Lee and David M. Brooks. Accurate and Efficient Regression Modeling for Microarchitectural Performance and Power Prediction. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XII, pages 185–194, New York, NY, USA, 2006. ACM.

[74] E.A. Lee and D.G. Messerschmitt. Synchronous Data Flow. *Proceedings of the IEEE*, 75(9):1235–1245, Sept 1987.

[75] Haeseung Lee and Mohammad Abdullah Al Faruque. GPU-EvR: Run-time Event Based Real-time Scheduling Framework on GPGPU Platform. In *Proceedings of the Conference on Design, Automation & Test in Europe*, DATE '14, pages 220:1–220:6. EDA Consortium, 2014.

[76] Jaejin Lee, Jungwon Kim, Sangmin Seo, Seungkyun Kim, Jungho Park, Honggyu Kim, Thanh Tuan Dao, Yongjin Cho, Sung Jong Seo, Seung Hak Lee, Seung Mo Cho, Hyo Jung Song, Sang-Bum Suh, and Jong-Deok Choi. An OpenCL Framework for Heterogeneous Multicores with Local Memory. In *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques*, PACT '10, pages 193–204, New York, NY, USA, 2010. ACM.

[77] Thierry Lepley, Pierre Paulin, and Eric Flamand. A Novel Compilation Approach for Image Processing Graphs on a Many-core Platform with Explicitly Managed Memory. In *Proceedings of the 2013 International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, CASES '13, pages 6:1–6:10, Piscataway, NJ, USA, 2013. IEEE Press.

[78] Simone Libutti, Giuseppe Massari, Patrick Bellasi, and William Fornaciari. Exploiting Performance Counters for Energy Efficient Co-Scheduling of Mixed Workloads on Multi-Core Platforms. In *Proceedings of Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures and Design Tools and Architectures for Multicore Embedded Computing Platforms*, PARMA-DITAM '14, pages 27–32, New York, NY, USA, 2014. ACM.

[79] G. Mariani, P. Avasare, G. Vanmeerbeeck, C. Ykman-Couvreur, G. Palermo, C. Silvano, and V. Zaccaria. An Industrial Design Space Exploration Framework for Supporting Run-time Resource Management on Multi-core Systems. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '10, pages 196–201. EDA Consortium, 2010.

[80] Giovanni Mariani, Gianluca Palermo, Vittorio Zaccaria, and Cristina Silvano. ARTE: An Application-specific Run-Time managEment framework for multi-cores based on queuing models. *Parallel Computing*, 39(9):504–519, 2013.

[81] J.F. Martinez and E. Ipek. Dynamic Multicore Resource Management: A Machine Learning Approach. *Micro, IEEE*, 29(5):8–17, September 2009.

[82] K. Matsumoto, N. Nakasato, and S.G. Sedukhin. Performance Tuning of Matrix Multiplication in OpenCL on Different GPUs and CPUs. In *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, pages 396–405. IEEE Press, November 2012.

[83] Diego Melpignano, Luca Benini, Eric Flamand, Bruno Jego, Thierry Lepley, Germain Haugou, Fabien Clermidy, and Denis Dutoit. Platform 2012, a Many-core Computing Accelerator for Embedded SoCs: Performance Evaluation of Visual Analytics Applications. In *Proceedings of the 49th Annual Design Automation Conference*, DAC '12, pages 1137–1142, New York, NY, USA, 2012. ACM.

[84] R. Membarth, F. Hannig, J. Teich, M. Korner, and W. Eckert. Generating Device-specific GPU Code for Local Operators in Medical Imaging. In *IEEE 26th International on Parallel Distributed Processing Symposium (IPDPS), 2012*, pages 569–581, May 2012.

[85] Richard Membarth, Jan-Hugo Lupp, Frank Hannig, Jürgen Teich, Mario Körner, and Wieland Eckert. Dynamic Task-scheduling and Resource Management for GPU Accelerators in Medical Imaging. In *Proceedings of the 25th International Conference on Architecture of Computing Systems*, ARCS'12, pages 147–159, Berlin, Heidelberg, 2012. Springer-Verlag.

[86] Nicholas Nethercote, Peter J Stuckey, Ralph Becket, Sebastian Brand, Gregory J Duck, and Guido Tack. Minizinc: Towards a standard CP modelling language. In *Principles and Practice of Constraint Programming – CP 2007*, pages 529–543. Springer, 2007.

[87] Akira Nukada and Satoshi Matsuoka. Auto-tuning 3-D FFT Library for CUDA GPUs. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 1–10, New York, NY, USA, 2009. ACM.

[88] NVIDIA Corporation. NVIDIA CUDA C Programming Guide. http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html, 2011.

[89] OpenMP Architecture Review Board. OpenMP Application Program Interface Version 3.1. http://www.openmp.org/mp-documents/OpenMP3.1.pdf, 2011.

[90] OpenMP Architecture Review Board. OpenMP Application Program Interface Version 4.0. http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf, 2013.

[91] David Opitz and Richard Maclin. Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research*, 11:169–198, 2009.

[92] Gianluca Palermo, Cristina Silvano, and Vittorio Zaccaria. ReSPIR: A Response Surface-Based Pareto Iterative Refinement for Application-Specific Design Space Exploration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(12):1816–1829, Dec 2009.

[93] M. Puschel, J.M.F. Moura, J.R. Johnson, D. Padua, M.M. Veloso, B.W. Singer, J. Xiong, F. Franchetti, A. Gacic, Y. Voronenko, K. Chen, R.W. Johnson, and N. Rizzolo. SPIRAL: Code Generation for DSP Transforms. *Proceedings of the IEEE*, 93(2):232–275, February 2005.

[94] Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing Pipelines. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '13, pages 519–530, New York, NY, USA, 2013. ACM.

[95] James Reinders. *Intel Threading Building Blocks*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, first edition, 2007.

[96] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA, 2006.

[97] Kathrin Rosvall and Ingo Sander. A constraint-based design space exploration framework for real-time applications on MPSoCs. In *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, pages 1–6, March 2014.

[98] Shane Ryoo, Christopher I. Rodrigues, Sara S. Baghsorkhi, Sam S. Stone, David B. Kirk, and Wen-mei W. Hwu. Optimization Principles and Application Performance Evaluation of a Multithreaded GPU Using CUDA. In *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '08, pages 73–82, New York, NY, USA, 2008. ACM.

[99] Adrian Sampson, Werner Dietl, Emily Fortuna, Danushen Gnanapragasam, Luis Ceze, and Dan Grossman. EnerJ: Approximate Data Types for Safe and General Low-power Computation. In *Proceedings of the 32Nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '11, pages 164–174, New York, NY, USA, 2011. ACM.

[100] Andreas Sandberg, David Black-Schaffer, and Erik Hagersten. Efficient techniques for predicting cache sharing and throughput. In *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*, PACT '12, pages 305–314, New York, NY, USA, 2012. ACM.

[101] Thomas J Santner, Brian J Williams, and William I Notz. *The Design and Analysis of Computer Experiments*. Springer, 2003.

[102] Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International journal of computer vision*, 47(1-3):7–42, 2002.

[103] Jiwon Seo, Yu-Hsuan Chen, David S De Lorenzo, Sherman Lo, Per Enge, Dennis Akos, and Jiyun Lee. A real-time capable software-defined receiver using GPU for adaptive anti-jam GPS Sensors. *Sensors*, 11(9):8966–8991, 2011.

[104] Jie Shen, Jianbin Fang, H. Sips, and AL. Varbanescu. Performance Traps in OpenCL for CPUs. In *21st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), 2013*, pages 38–45, Feb 2013.

[105] Hamid Shojaei, AmirHossein Ghamarian, Twan Basten, Marc Geilen, Sander Stuijk, and Rob Hoes. A parameterized compositional multi-dimensional multiple-choice knapsack heuristic for CMP run-time management. In *Proceedings of the 46th Annual Design Automation Conference*, pages 917–922. ACM, 2009.

[106] Stelios Sidiroglou-Douskos, Sasa Misailovic, Henry Hoffmann, and Martin Rinard. Managing Performance vs. Accuracy Trade-offs with Loop Perforation. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ESEC/FSE '11, pages 124–134, New York, NY, USA, 2011. ACM.

[107] C. Silvano, W. Fornaciari, S. Crespi Reghizzi, G. Agosta, G. Palermo, V. Zaccaria, P. Bellasi, F. Castro, S. Corbetta, A. Biagio, E. Speziale, M. Tartara, D. Melpignano, J. M. Zins, D. Siorpaes, H. Hübert, B. Stabernack, J. Brandenburg, M. Palkovic, P. Raghavan, C. Ykman-Couvreur, A. Bartzas, S. Xydis, D. Soudris, T. Kempf, G. Ascheid, R. Leupers, H. Meyr, J. Ansari, P. Mähönen, and B. Vanthournout. 2PARMA: Parallel Paradigms and Run-time Management Techniques for Many-Core Architectures. In *VLSI 2010 Annual Symposium*, volume 105 of *Lecture Notes in Electrical Engineering*, pages 65–79. Springer Netherlands, 2011.

[108] C. Silvano, W. Fornaciari, G. Palermo, V. Zaccaria, F. Castro, M. Martinez, S. Bocchio, R. Zafalon, P. Avasare, G. Vanmeerbeeck, C. Ykman-Couvreur, M. Wouters, C. Kavka, L. Onesti, A. Turco, U. Bondi, G. Mariani, H. Posadas, E. Villar, C. Wu, Fan Dongrui, Zhang

hao, and T. Shibin. MULTICUBE: Multi-objective Design Space Exploration of Multi-core Architectures. In *2010 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 488–493, July 2010.

[109] Deshanand P. Singh, Tomasz S. Czajkowski, and Andrew Ling. Harnessing the Power of FPGAs Using Altera's OpenCL Compiler. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '13, pages 5–6, New York, NY, USA, 2013. ACM.

[110] K.I. Smith, R.M. Everson, J.E. Fieldsend, C. Murphy, and R. Misra. Dominance-Based Multiobjective Simulated Annealing. *IEEE Transactions on Evolutionary Computation*, 12(3):323–342, June 2008.

[111] Bharat Sukhwani and Martin C. Herbordt. GPU Acceleration of a Production Molecular Docking Code. In *Proceedings of 2Nd Workshop on General Purpose Processing on Graphics Processing Units*, GPGPU-2, pages 19–27, New York, NY, USA, 2009. ACM.

[112] Wai Teng Tang, Wen Jun Tan, R. KrishnaMoorthy, Yi Wen Wong, Shyh hao Kuo, R.S.M. Goh, S.J. Turner, and Weng-Fai Wong. Optimizing and Auto-Tuning Iterative Stencil Loops for GPUs with the In-Plane Method. In *IEEE 27th International Symposium on Parallel Distributed Processing (IPDPS), 2013*, pages 452–462. IEEE Press, May 2013.

[113] A. Tarakji, M. Marx, and S. Lankes. The development of a scheduling system GPUSched for graphics processing units. In *International Conference on High Performance Computing and Simulation (HPCS), 2013*, pages 566–575, July 2013.

[114] Jürgen Teich, Jörg Henkel, Andreas Herkersdorf, Doris Schmitt-Landsiedel, Wolfgang Schröder-Preikschat, and Gregor Snelting. Invasive computing: An overview. In *Multi-processor System-on-Chip*, pages 241–268. Springer, 2011.

[115] Philippe Tillet, Karl Rupp, and Siegfried Selberherr. An Automatic OpenCL Compute Kernel Generator for Basic Linear Algebra Operations. In *Proceedings of the 2012 Symposium on High Performance Computing*, HPC '12, pages 1–2, San Diego, CA, USA, 2012. Society for Computer Simulation International.

[116] Vasily Volkov and James W. Demmel. Benchmarking GPUs to Tune Dense Linear Algebra. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, SC '08, pages 31:1–31:11, Piscataway, NJ, USA, 2008. IEEE Press.

[117] John Robert Wernsing and Greg Stitt. Elastic Computing: A Framework for Transparent, Portable, and Adaptive Multi-core Heterogeneous Computing. *SIGPLAN Not.*, 45(4):115–124, April 2010.

[118] R. Clint Whaley and Antoine Petitet. Minimizing development and maintenance costs in supporting persistently optimized BLAS. *Software: Practice and Experience*, 35(2):101–121, February 2005.

[119] Stefan Wildermann, Tobias Ziermann, and Jürgen Teich. Game-theoretic Analysis of Decentralized Core Allocation Schemes on Many-core Systems. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '13, pages 1498–1503, San Jose, CA, USA, 2013. EDA Consortium.

[120] David H. Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992.

[121] Ch. Ykman-Couvreur, V. Nollet, F. Catthoor, and H. Corporaal. Fast Multidimension Multichoice Knapsack Heuristic for MP-SoC Runtime Management. *ACM Transactions on Embedded Computing Systems (TECS)*, 10(3):1–16, April 2011.

[122] Chantal Ykman-Couvreur, Prabhat Avasare, Giovanni Mariani, Gianluca Palermo, Cristina Silvano, and Vittorio Zaccaria. Linking run-time resource management of embedded multi-core platforms with automated design-time exploration. *Computers & Digital Techniques, IET*, 5(2):123–135, 2011.

[123] Chantal Ykman-Couvreur, Philipp A. Hartmann, Gianluca Palermo, Fabien Colas-Bigey, and Laurent San. Run-time Resource Management Based on Design Space Exploration. In *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, CODES+ISSS '12, pages 557–566, New York, NY, USA, 2012. ACM.

[124] Vittorio Zaccaria, Gianluca Palermo, Fabrizio Castro, Cristina Silvano, and Giovanni Mariani. Multicube explorer: An open source framework for design space exploration of chip multiprocessors. In *23rd International Conference on Architecture of Computing Systems (ARCS), 2010*, pages 1–7. VDE, 2010.

[125] Ke Zhang, Jiangbo Lu, and Gauthier Lafruit. Cross-based Local Stereo Matching Using Orthogonal Integral Images. *IEEE Transactions on Circuits and Systems for Video Technology*, 19(7):1073–1079, 2009.

[126] Ke Zhang, Jiangbo Lu, Qiong Yang, G. Lafruit, R. Lauwereins, and L. Van Gool. Real-Time and Accurate Stereo: A Scalable Approach With Bitwise Fast Voting on CUDA. *IEEE Transactions on Circuits and Systems for Video Technology*, 21(7):867–878, July 2011.