Edited version: Thesis Overview

Software Parallelization and Distribution for Heterogeneous Multi-Core Embedded Systems

Von der Fakultät für Elektrotechnik und Informationstechnik
der Rheinisch–Westfälischen Technischen Hochschule Aachen
zur Erlangung des akademischen Grades
eines Doktors der Ingenieurwissenschaften
genehmigte Dissertation

vorgelegt von
Miguel Angel Aguilar Ulloa, M.Sc.
aus Cartago, Costa Rica

Berichter: Universitätsprofessor Dr. rer. nat. Rainer Leupers
Universitätsprofessor Dr.-Ing. Jeronimo Castrillon

Tag der mündlichen Prüfung: 28.11.2018

Dedicated to the memory
of my beloved mother

Acknowledgements

This dissertation is the result of my doctoral research work at the Institute for Communication Technologies and Embedded Systems (ICE) at the RWTH Aachen University. During the 6 years that I spent at ICE, I was supported by many great people. It is my pleasure to start this document by expressing my gratitude to them.

First, I would like to thank my advisor Professor Rainer Leupers for giving me the opportunity to join ICE. His excellent guidance and his trust on my work was decisive to successfully complete my doctoral degree. One of the most valuable lessons that I learned from him was to focus my research efforts on practical problems that matter for industry, instead of focusing on pure theoretical problems with limited applicability. I would like to also thank Professor Jeronimo Castrillon for serving as a reviewer of this dissertation. I have always admired his work, which was a model and a source of inspiration during my doctoral studies.

During my time at the ICE, I had the pleasure to work together with amazing colleagues and students that created a friendly and supportive environment. Special thanks go to María Auras-Rodríguez, Juan Eusse, Luis Murillo, Robert Bücs, Jan Weinstock, Dominik Šišejković and Diego Pala. I am grateful to them for their support in the ups and downs, great technical discussions, reviewing my publications and more important for making me feel like at home during these years. I also would like express my gratitude to the non-scientific staff at ICE, especially to Tanja Palmen and Elisabeth Böttcher, for helping me with many administrative matters.

I cannot thank enough Diego Pala, Thomas Grass, Maria Auras-Rodríguez, Robert Bücs and Farhad Merchant for proof-reading this dissertation. Their excellent feedback made possible to bring this document into its final state.

My deepest gratitude is for my mother. Without your dedication and sacrifices, I would have never been in the position to accomplish all what I have done. Loosing you during my doctoral studies was a very hard moment, but even during your last days you gave me the courage to complete this. Another extremely important person in my life is Lena. Along this journey, Lena has both unconditionally supported me during difficult times and celebrated my accomplishments as if they were her own. Thanks for making me feel Germany as my home.

Miguel Angel Aguilar Ulloa, January 2019

Contents

Chapter 1

Introduction

For many years during the *single-core era*, software developers took performance improvements for granted thanks to enhanced microarchitectures and increased clock frequencies in every new processor generation. This trend was enabled by the ever increasing number of transistors in integrated circuits, as described by Moore's Law [267]. This processor design paradigm was expected to last for many more years. In 2002, it was predicted that by 2010 processors would be running at 30 GHz [30, 73]. However, this was never possible due to power consumption and thermal issues associated with the increasing clock frequencies [303]. Then, in 2005 the end of the single-core era was described by Herb Sutter as "The free lunch is over" [280]. This crisis motivated a fundamental change in the processor design paradigm in which transistors are used to build architectures with multiple cores, instead of increasing the complexity and performance of a monolithic core. Figure 1.1 illustrates these trends, where after 2004 the number of cores started to grow, while the single-core performance, frequency and power consumption started to saturate [258]. This new processor paradigm brought new eras of computing known as the *homogeneous multi-core era* (or simply *multi-core era*) with the increase in the number of cores and then the *heterogeneous multi-core era* (or simply *heterogeneous era*) with the specialization of the cores [32, 128, 297].

The multi-core and heterogeneous eras impacted not only desktop and High Performance Computing (HPC) but also embedded computing. In the embedded domain, the underlying technologies of the systems evolved into complex heterogeneous Multi-Processor System-on-Chips (MPSoCs), which combine multiple cores of a variety of types (e.g., General Purpose Processors (GPPs), Digital Signal Processors (DSPs) and Graphics Processing Units (GPUs)) [142]. These MPSoCs are able to meet the demands of the embedded market that is continuously pushing for high performance at lower energy consumption and cost. Nowadays heterogeneous MPSoC are widely used in the design of devices from smartphones and tablets that provide a rich variety of services, to cars that are evolving towards autonomous supercomputers on wheels.

This evolution in the processor design paradigm also implied a major change in the programming paradigm from sequential to parallel. Tim Mattson, a renowned senior parallel computing scientist, argues that the advent of parallel programming is not due to an achievement of the software, but instead due to a failure of the hardware [117]. Parallel programming has shown to be a challenging task [46, 206, 320]. The current practice relies on a manual and error-prone program transformation process in which a large amount of legacy sequential software has to be migrated to parallel systems [90, 117, 144, 197, 324]. Therefore, there is an urgent need for solutions to succeed in this dramatic paradigm shift.

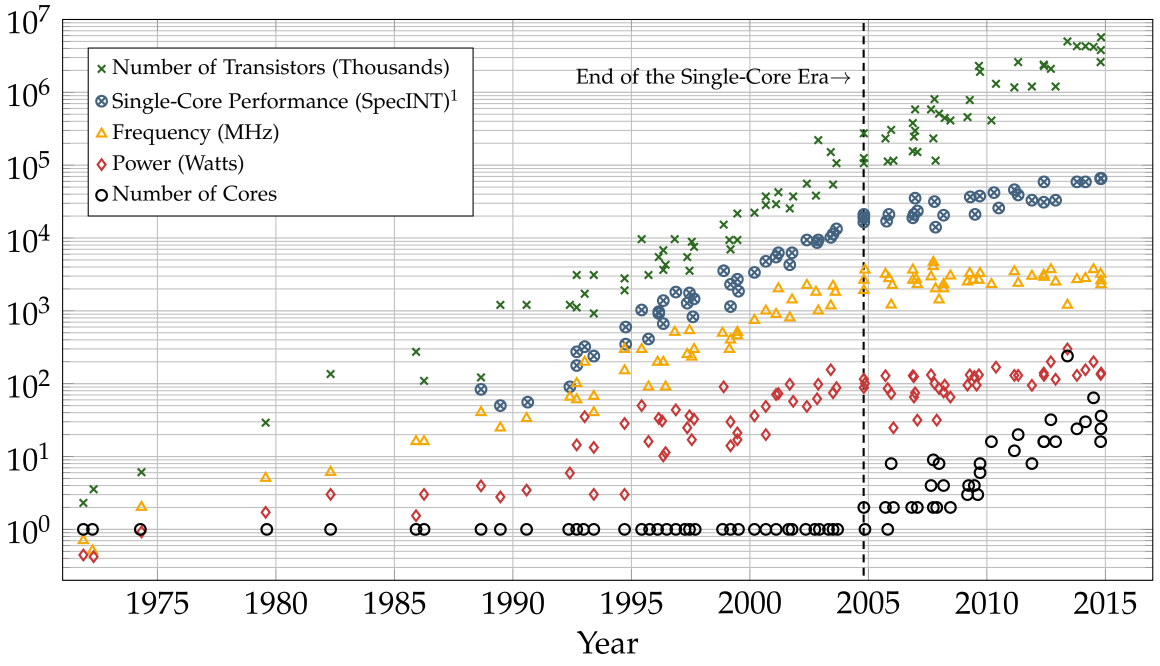


Figure 1.1: 40 Years of Microprocessor Trend Data (Data from: [258])

This thesis aims at addressing the challenges posed by the multi-core and heterogeneous eras with focus on the embedded domain, to relieve developers from the hectic and error-prone manual process of software optimization for heterogeneous multi-core systems. For this purpose, in this thesis a tool flow is proposed based on novel compiler technologies to automatically optimize legacy sequential programs for a proper parallel execution. This tool flow builds a model of the programs, which is then analyzed by multiple heuristics to identify software parallelization and distribution opportunities. These optimization opportunities are then realized by generating parallel code in multiple state-of-the-art programming paradigms, which allows to apply the tool flow to a wide range of relevant commercial platforms.

The remainder of this chapter is organized as follows. Section 1.1 discusses key aspects of the multi-core and heterogeneous eras. The list of requirements that a tool for software parallelization and distribution should meet is discussed in Section 1.2. Section 1.3 presents an overview of the proposed tool flow. The contributions made in this thesis are stated in Section 1.4. Finally, Section 1.5 concludes with a summary of the chapter and outlines the rest of the document.

1.1 The Challenge: Entering a Heterogeneous Parallel Universe

This section discusses key aspects of the multi-core and heterogeneous eras with emphasis on the embedded domain. In addition, the challenge of legacy sequential code, as well as the details of the current parallel programming practice are described.

[1] The SPECInt is a benchmark specification for evaluating integer CPU performance [71]

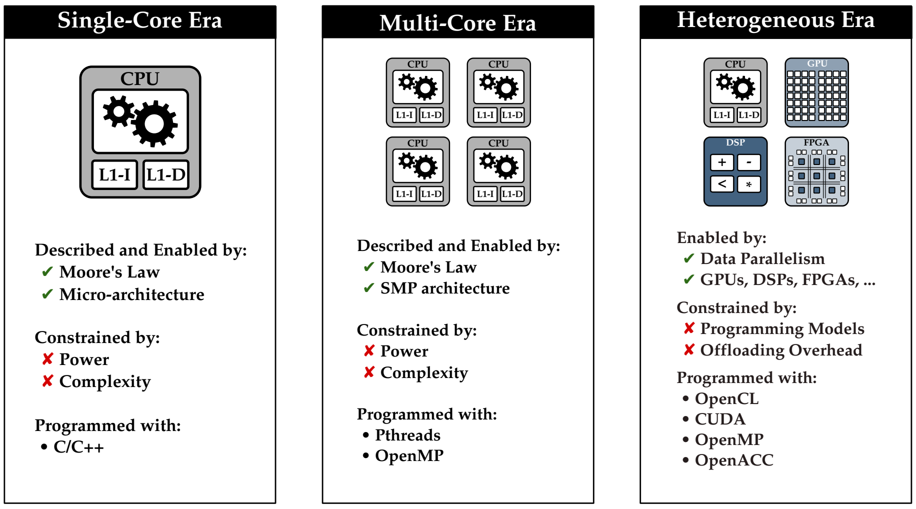


Figure 1.2: Eras of Processor Design Paradigms (Adapted from [32])

1.1.1 From the Single-Core to the Multi-Core & Heterogeneous Eras

Figure 1.2 summarizes key aspects of the single-core, multi-core and heterogeneous eras, such as the enablers, constrainers and their programming paradigms. As previously discussed, the single-core era was described by Moore's Law, and enabled by advances in microarchitecture technologies and increasing clock frequencies. In addition, currently it is possible to exploit close to the full hardware potential offered by a single-core. In terms of programming paradigms, a large amount programs were developed in languages, such as, C and C++, which resulted in what is known today as *legacy sequential software* [209]. However, the single-core era hit a limit often referred as the *power wall* [303], which opened the doors to parallel computing.

Similarly to the single-core era, the multi-core era is still described by Moore's Law, but also enabled by Symmetric Multiprocessing (SMP) architectures. An SMP system involves two or more tightly coupled homogeneous cores connected to a single shared-memory, which is controlled by one single Operating System (OS). In the embedded domain, the use of homogeneous MPSoCs gained acceptance as the underlying technology to design multi-core devices. On the one hand, MPSoCs provide a proper trade-off between performance, power consumption and cost. On the other hand, MPSoCs enable a component reuse strategy known as *platform-based design* [266], which helps to reduce the Non-Recurring Engineering (NRE) costs and to meet the strict time-to-market requirements. Figure 1.3 shows the most recent prediction of the number cores in MPSoCs, published by the International Technology Roadmap for Semiconductors (ITRS) in 2015 [138]. According to ITRS, by 2027 it is expected that MPSoCs are going to incorporate over 300 cores. However, currently there is still a significant gap between the attainable performance of these platforms and the actual performance that the current software is able to achieve on them [52]. This gap can be closed by means of *software parallelization* with the extraction of multiple parallel patterns, which is addressed in Chapter 4.

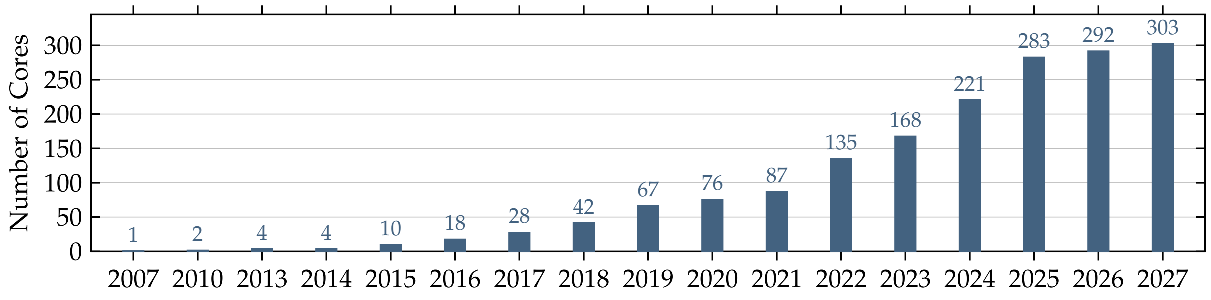


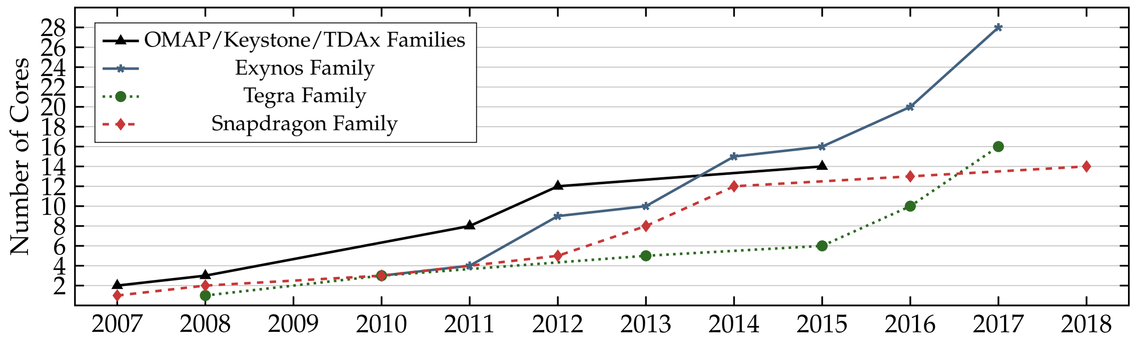
Figure 1.3: ITRS 2.0 Prediction of the Core Count in MPSoCs (Data from: [138])

In terms of programming paradigms, it is worth to mention two widely used examples in the multi-core era: POSIX Threads (Pthreads) [1] and Open Multi-Processing (OpenMP) [36]. Pthreads is a library-based programming paradigm for shared memory systems. It is a low level approach, as the developer has to explicitly perform thread management, workload partitioning and synchronization. The accesses to critical sections (shared data) have to be carefully designed to avoid *data races* and *deadlocks* by means of mutual exclusion (mutex). OpenMP is an industry standard programming model also for shared memory systems based on compiler directives, originally designed for homogeneous multi-core platforms. The use of compiler directives is a high-level approach that requires minimal source code modifications in contrast to Pthreads. In addition, the OpenMP runtime system takes care of the thread management. Although Pthreads and OpenMP initially targeted desktop computing and HPC, both have been used in the embedded domain as well [80, 277, 300].

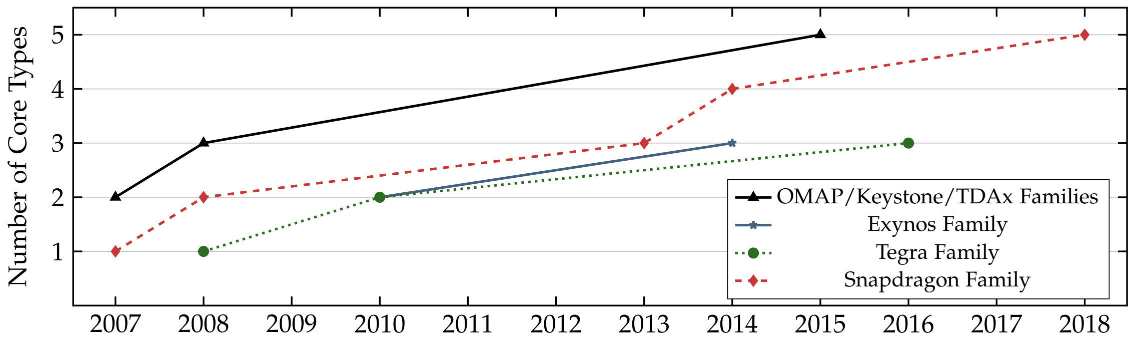
The ever increasing requirements of modern applications pushed yet another major change in the processor design paradigm known as the *heterogeneous era* [32, 255, 322]. Nowadays, mobile devices such as smartphones and tablets provide a wide range of features beyond calling and texting, which includes video & audio processing, emailing, gaming and navigation among others. These features are enabled by applications that are diverse in nature. Therefore, this brought a need for specialization by means of *heterogeneous computing*, where computationally intensive workloads can be more efficiently processed in terms of performance and power consumption on specialized cores. Although the use of these type of cores have been around for many years, it was until late 2000s when they started to become widely accessible to the developers as programmable cores [223]. This is exemplified by the advent of General-Purpose computing on Graphics Processing Units (GPGPU). The heterogeneity can be manifested in multiple forms: *(i)* multiple cores with the same Instruction Set Architecture (ISA) running at different clock frequencies, *(ii)* multiple cores with different ISAs, and *(iii)* programmable cores combined with Field-Programmable Gate Arrays (FPGAs). The complexity introduced in this era by the diversity of the cores, poses new challenges both in terms of hardware and software [322]. In addition, the performance gap between the hardware platforms and the current software has grown in contrast to the multi-core era. This gap can be closed by means of *software distribution* based on accelerator offloading, which is addressed in Chapter 5.

It is worth mentioning that in 2012 the Heterogeneous System Architecture (HSA) foundation appeared as a major effort to simplify this new era of heterogeneous computing through standardization [129]. HSA is a consortium composed of various semiconductor companies, Intellectual Property (IP) providers, tool providers, software vendors, and academic institutions. The HSA foundation strives for improving heterogeneous computing by providing specifications for multiple aspects of the systems including the platform architecture, programming model, runtime system, tooling and multi-vendor compatibility. These specifications are already in practice in multiple domains from embedded to HPC. The Bifrost microarchitecture implemented in the Mali-G71 GPU from ARM [260] and the recent Exynos 8895 MPSoC from Samsung [265, 271] are examples of embedded platforms compliant with the HSA 1.1 hardware specification.

The heterogeneous era also brought new programming paradigms, mainly following a host-centric model in which host cores offload workloads described as computational *kernels* to accelerators [78]. About a decade ago in 2007, the first version of the parallel programming paradigm called Compute Unified Device Architecture (CUDA) was released by NVIDIA [214]. The main goal of this paradigm is to allow the use of NVIDIA GPUs for general purpose computing. CUDA provides an Application Programming Interface (API) that allows to write both the code in the host side, as well as the kernel code for the GPU side. Besides the API, CUDA is also accompanied by a wide ecosystem of tools and specialized libraries for multiple domains [213]. Another programming paradigm for heterogeneous systems called Open Compute Language (OpenCL) was released for the first time in 2009 [276]. Similar to CUDA, OpenCL provides an API to program both the host side code and the kernel code. However, in contrast to CUDA, OpenCL is an open industry standard supported on a variety of platforms from different vendors. In addition, OpenCL supports devices beyond GPUs (e.g., DSPs, FPGAs and other accelerators). Both CUDA and OpenCL are classified as *low-level* paradigms, which imply a significant programming effort. Therefore, *high-level* paradigms based on compiler directives emerged as an alternative to program heterogeneous systems with less effort. In 2011, the first specification of Open Accelerators (OpenACC) was released [225]. OpenACC is a open industry standard managed by a consortium composed of industry and academic members. It aims to be a performance portable model to program accelerators based on compiler directives, which allow to offload both data and computations to accelerators. In 2013, OpenMP also entered the heterogeneous era with the introduction of the *accelerator model* as part of the OpenMP 4.0 specification [226]. Similarly to OpenACC, the accelerator model allows to offload data and computation by means of compiler directives. It is worth mentioning that in the academia, data-flow Models of Computation (MoCs) gained acceptance, since they are suitable to describe embedded streaming programs on heterogeneous systems [52, 116]. These MoCs describe programs as a network of autonomous processes that exchange data through FIFO channels. Prominent examples of these MoCs are Kahn Process Network (KPN) [99] and Synchronous Data Flow (SDF) [168].



(a) Multi-Core Trend



(b) Heterogeneous Trend

Figure 1.4: Trends in Commercial MPSoC Families (Data from: [51, 52, 283, 285, 288, 312, 313, 314, 315])

To exemplify the impact of the multi-core and heterogeneous eras in the embedded domain, the evolution in terms of the number and diversity of programmable cores in various commercial[2] MPSoC families is presented in Figure 1.4. The first trend is composed of three MPSoC families of Texas Instruments (TI): Open Multimedia Applications Platform (OMAP), Keystone [283] and TDAx [288]. These families are presented here as a single trend line, since together they represent the overall evolution of the MPSoC strategy of TI. The trend begins in 2007 with the OMAP family, starting with the dual-core OMAP1 platform, as shown in Figure 1.4a. The main focus of the OMAP family was smartphones and tablets. However, in 2012 TI decided to leave the mobile market [45]; thus, bringing this family to an end. In 2011, before announcing this market shift, TI already made an important step by introducing the Keystone family to target new markets [283]. In its first generation, this family provided homogeneous platforms with up to 8 C66x DSPs [289]. Then, in its second generation, the Keystone family introduced heterogeneous platforms with up to 12 cores: 4 ARM Cortex-A15 cores and 8 C66x DSP cores [285]. Another major step by TI was the introduction of the TDAx family in 2015 to target Advanced Driver-Assistance Systems (ADAS). The most relevant aspect of this family is its heterogeneity, as Figures 1.4b and 1.5 show. This platform has up to 14 cores of 5 different types: 2 ARM Cortex-A15 cores, 4 ARM Cortex-M4 cores, 2 C66x DSP cores, a dual-core SGX544 GPU and 4 Embedded Vision Engine (EVE) cores [287].

[2] Company, product and brand names used in this thesis may be trademarks or registered trademarks of their respective owners

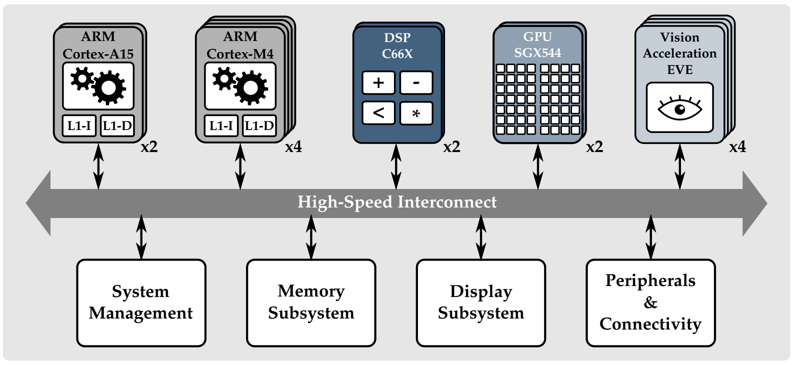


Figure 1.5: TDA2SX: A Highly Heterogeneous MPSoC [287]

Another prominent family of MPSoCs is Snapdragon from Qualcomm [248]. The core count trend of this family has grown since 2007 from 1 core in the Snapdragon S1 platform to 13 cores in Snapdragon 845 platform in 2018 [313], as Figure 1.4a shows. In terms of heterogeneity, this family reached 5 different core types in a single MPSoC with the Snapdragron 845 [247]. This MPSoC has 4+4 Kryo cores (custom Cortex-A75 and Cortex-A55), 1 Adreno 630 GPU, a Hexagon 685 DSP with 4 cores and 1 Secure Processing Unit (SPU). The Exynos family from Samsung is another example of widely used MPSoCs in the mobile market [312]. The core count trend of this family presents the most dramatic growth, starting in 2010 with 3 cores in the Exynos 3 platforms to 28 cores in the Exynos 9 platforms in 2017 [312], as Figure 1.4a shows. The Exynos 9 MPSoC combines 4 M1 "Mongoose" cores, 4 ARM Cortex-A53 and a Mali-G71 MP20 GPU with 20 cores. Finally, the last family considered here is Tegra from NVIDIA, which targets a variety of domains including mobile, gaming and automotive [222]. Recently, this family has gained a strong relevance due to its use in the *deep learning* domain [215]. The core count trend of the Tegra platforms goes from 1 core in the Tegra APX 2500 platform in 2008 to 16 cores in the Xavier platform announced in 2017 [217], as Figure 1.4a shows. The latest available platform of this family is the Tegra X2, which combines 2 Denver2 cores, 4 ARM Cortex-A57 and a Pascal GPU with 8 Streaming Multiprocessors (SMs). The Drive PX 2 platform for autonomous cars is an example of a system that incorporates 2 Tegra X2 MPSoCs together with 2 discrete Pascal GPUs [216]. The previous trends strongly suggest that embedded devices will keep evolving towards highly parallel and heterogeneous systems, not only due to the increasing number and diversity of cores within a single MPSoC, but also due to the use of multiple MPSoCs within a single system.

1.1.2 Current Programming Practice: Legacy Sequential Code

Parallel programming has shown to be a challenging task, as for humans it is more natural to *think sequentially*. Moreover, many generations of developers have been trained to design and program sequentially. However, still nowadays there is an open debate about how and when universities should teach to *think parallel* [117, 146, 191, 200, 205]. As previously discussed, multiple paradigms have been developed to address the issue of parallel programming in the multi-core and heterogeneous eras.

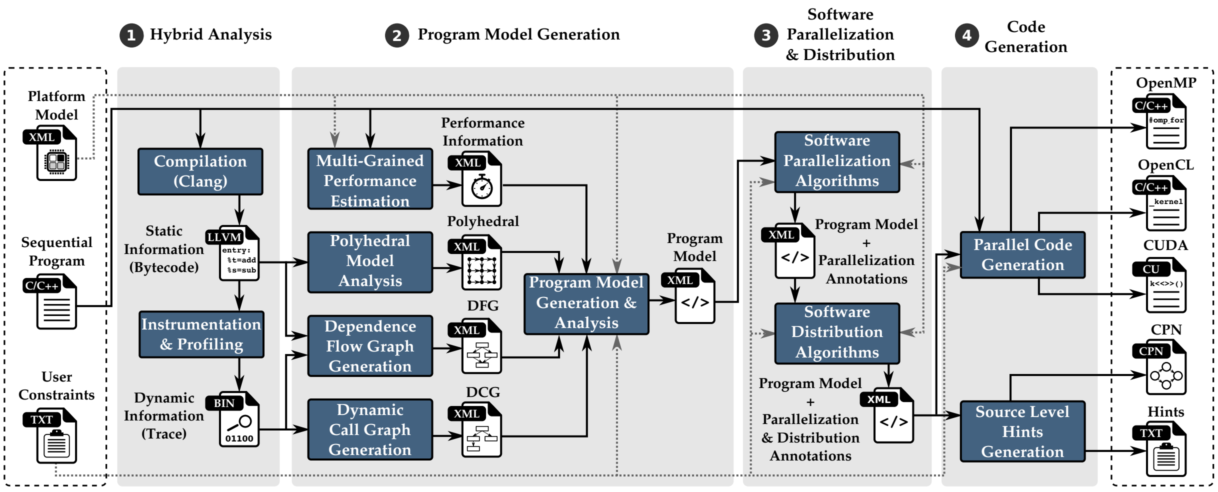
Despite these efforts for providing a convenient programming paradigm, developers still have the cumbersome task of writing efficient and correct parallel code. This task is even more challenging considering that the current practice for software development relies on program transformation of existing legacy code instead of designing everything from scratch [144]. It has been estimated that the amount of legacy code in new developments exceeds the new code by a factor of 100 to 1 or even 1000 to 1 [90]. Moreover, developers are not going to adopt completely new parallel languages to reimplement the huge amount of existing sequential code [117]. Instead, developers have to incrementally optimize this legacy code (mostly written in C/C++ in the embedded domain [47]) for an efficient execution on modern heterogeneous multi-core systems [197]. This is an extremely error-prone and time-consuming task in which developers have to perform multiple manual steps:

- ***Getting Familiar with the Legacy Source Code***: The first step is to understand the sequential code to be optimized. This task is especially challenging when the code was written by someone else, which is typically the case. Therefore, developers have to follow multiple strategies to tackle unknown legacy code [209].
- ***Identifying Computationally Intensive Code Regions***: To achieve a profitable optimization of the legacy code, developers have to focus on optimizing computationally intensive sections of the programs. This can be achieved by using profiling and performance estimation tools available for the platform of interest [51, 52].
- ***Understanding Data Dependencies***: Developers have to identify and understand data dependencies in the sequential code to preserve the functional correctness of the program when parallelism is extracted [199, 319]. This is one of the most challenging steps for developers, especially in "spaghetti code" [307]. In addition, other parallelization inhibitors should be identified, including functions with side effects or unstructured code (e.g., `goto`, `continue` and `break` statements) [16].
- ***Identifying Parallelization Opportunities***: This is a key step in which developers have to identify and select the most profitable parallelization opportunities within computationally intensive code sections. The most prominent parallel patterns include Data Level Parallelism (DLP), Pipeline Level Parallelism (PLP), Task Level Parallelism (TLP) and Recursive Level Parallelism (RLP) [10, 7, 51, 52, 70, 158].
- ***Identifying Software Distribution Opportunities***: For heterogeneous multi-core systems, developers have to identify code regions that are good candidates to be offloaded to accelerators (e.g., GPUs or DSPs). For this purpose, developers have to make sure that: *(i)* a code region exhibits a significantly higher performance when is executed on a given accelerator than on the host cores, and *(ii)* the offloading overhead does not outweight the benefit of offloading a code region.
- ***Writing the Parallel Code***: The final step is to realize the identified software parallelization and distribution opportunities by means of parallel paradigms available on the target platform. Here developers have to perform various degrees of code transformations and refactorizations according to the parallel paradigm. This task by itself has proved to be extremely challenging [104, 149, 211, 279].

1.2 The Solution: Tools for Software Parallelization and Distribution

The most prominent solution to help developers in the process of evolving legacy sequential code into the parallel space is the development of frameworks for automated software parallelization and distribution [73, 78, 117, 131, 199, 261]. Previous research efforts have left valuable observations and techniques from which it is possible to derive the following set of requirements that an effective tool flow in the embedded domain should meet:

- ***Coding Style and User Constraints:*** Multiple existing parallelization frameworks impose restrictions to the type of code that they can handle [31, 161]. However, the tools should be able to handle a wide variety of source code without imposing important restrictions that can limit the applicability of these frameworks. Moreover, tools should focus on relevant programming languages according to the domain of interest (e.g., C and C++ in the embedded domain [47]). In addition, developers should be allowed to have some degree of control to guide the analyses being performed by the tools (e.g., by means of user constraints to configure the analysis).
- ***Platform Model:*** To achieve the best results in the embedded domain, it is important that the tools are aware of the characteristics of the underlying platforms. For example, in terms of the number and types of cores, communication costs and task creation overhead [10, 51, 52, 70, 141]. This issue has motivated the emergence of industry standards, such as the *Software-Hardware Interface for Multi-many-core* (SHIM) specification [204]. SHIM is a standard from the *Multicore Association* (MCA) for abstracting hardware properties that are key to enable multi-core tools.
- ***Profile-Driven Analysis:*** Traditionally, state-of-the-art parallelizing compilers relied only on *static analysis*. However, it has been observed that this approach often failed to extract parallelism in languages like C/C++, as they allow the use of pointers, dynamic memory allocation and indirect function calls [294]. To overcome these issues, multiple authors agreed that *dynamic analysis* can be used as either an alternative or a complement to static analysis [148, 162, 294].
- ***Profitability Analysis:*** A key aspect for a profitable parallelization is performance information for two main reasons: *(i)* it allows to identify computationally intensive code sections, and *(ii)* it allows to perform a cost-benefit analysis to evaluate the potential of a given optimization opportunity. Multiple existing tools [120, 294] make use of static information for hotspot identification and cost-benefit analysis. However, this approach might result in missing profitable optimization opportunities or even in a slow down [296]. For this reason, more accurate performance estimation techniques at various program granularities are necessary.
- ***Forms of Parallelism:*** Early works focused on extraction of DLP from loops in which each iteration is independent from the others [155]. While DLP is abundant in scientific applications, studies like [139, 153] show that in the embedded domain additional forms of parallelism should be explored (e.g., TLP, PLP or RLP).



1 Hybrid Analysis
2 Program Model Generation
3 Software Parallelization & Distribution
4 Code Generation
Platform Model
XML
Sequential Program
C/C++
User Constraints
TXT
Compilation (Clang)
Static Information (Bytecode)
LLVM
Instrumentation & Profiling
Dynamic Information (Trace)
BIN
01100
Multi-Grained Performance Estimation
Performance Information
Polyhedral Model Analysis
Polyhedral
Dependence Flow Graph Generation
DFG
Dynamic Call Graph Generation
DCG
Program Model Generation & Analysis
Program Model
Software Parallelization Algorithms
Program Model + Parallelization Annotations
Software Distribution Algorithms
Program Model + Parallelization & Distribution Annotations
Parallel Code Generation
Source Level Hints Generation
OpenMP
OpenCL
CUDA
CPN
Hints

Figure 1.6: Tool Flow Overview

- ***Heterogeneity:*** The current practice in terms of software distribution in heterogeneous systems is based on a host-centric model, where host cores (e.g., CPUs) offload code sections and data to specialized cores (e.g., GPUs or DSPs). Therefore, together with the extraction of multiple forms of parallelism, tools should be able to automatically identify the best core types to offload computationally intensive code sections, which typically exhibit abundant DLP [189, 233].
- ***Source Level Hints:*** High-level information should be presented to the developers in the form of intuitive source level hints [51, 52, 135, 141]. This information allows developers to get a general understanding about the characteristics of the applications and to assess its optimization potential.
- ***Parallel Programming Paradigms:*** The usefulness of software tools highly depends on the platforms to which they can be applied. Therefore, tools should be able to realize software optimization opportunities on multiple relevant parallel programming paradigms. In the embedded domain, industry paradigms such as OpenMP, OpenCL and CUDA have gained strong acceptance and currently are being supported in a wide variety of platforms [160, 277, 286, 300].

1.3 Overview of the Proposed Tool Flow

By taking the requirements described in the previous section into account, in this thesis a tool flow for software parallelization and distribution for heterogeneous multi-core embedded systems is proposed, as shown in Figure 1.6. This tool flow was developed in the context of the framework called MPSoC Application Programming Studio (MAPS) [51, 52, 53, 54, 269] of the RWTH Aachen University. Section 2.1.2.5 describes how this thesis contributes to the existing facilities of the MAPS framework. The proposed tool flow takes as inputs a sequential C/C++ program, a model of the target platform and constraints provided by developers. The tool flow itself is composed of four phases as explained in the following. During the first phase a hybrid program analysis (❶ in Figure 1.6) takes place, which collects *static* and *dynamic* information. While the static analysis gathers compile time information, such as the complete control flow, variable declarations and memory accesses; the dynamic analysis gathers runtime information such as a list of executed functions, basic block execution count and memory accesses involving pointers or dynamically allocated memory. The dynamic information is obtained by instrumenting the program and executing it to generate a trace file. A Program Model (PM) is generated during the following phase (❷ in Figure 1.6). This model describes the input program in terms of performance information, a graph that expresses the calling relationships among functions in a given profiling run, and an Intermediate Representation (IR) that describes control and data dependencies among code statements, as well as the hierarchy of code regions. Afterwards, the PM is analyzed in first place by heuristics that perform the software parallelization in which multiple forms of parallelism are extracted, followed by heuristics that perform the software distribution in which code regions are selected for accelerator offloading (❸ in Figure 1.6). The results of this phase are stored in the PM in the form of annotations, which are later used during the code generation phase. Finally, during the last phase (❹ in Figure 1.6) information in the form of source level hints is presented to developers to give a general understanding of the characteristics of the program and its optimization potential. In addition, during this phase is where the parallel code in multiple paradigms is generated (i.e., OpenMP, OpenCL and CUDA and CPN [269]). The details of the proposed tool flow shown in Figure 1.6 and its evaluation are presented in the following chapters.

1.4 Contributions

Having introduced the tool flow in the previous section, it is now possible to precisely describe the contributions of this thesis. These contributions can be found in multiple phases of the proposed tool flow, from the PM and its analysis, to the realization of the identified optimization opportunities. The major contributions are outlined in the following:

- ***Program Model (Chapter 3):*** In thesis, it is proposed a unified representation of the program [5, 6, 10], which includes all the information required for an effective software parallelization and software distribution for heterogeneous multi-core embedded systems. Furthermore, this thesis contributes with techniques to model and analyze challenging language constructs (e.g, `while` loops [9]), which are not typically supported by existing tools, thus missing important optimization opportunities.
- ***Multi-Grained Performance Estimation (Chapter 3):*** The selection of code granularity is a major issue in frameworks for software parallelization and distribution, as it has a direct impact on the form and degree of parallelism that can be exploited. Typical granularities include: statement, basic block, loops, function and arbitrary code blocks. Therefore, software parallelization and distribution frameworks re-

quire performance information at these granularities. This thesis contributes with a flexible approach to provide performance information at multiple granularities [8].

- ***Software Parallelization Heuristics (Chapter 4):*** In this thesis, heuristics are proposed to extract four different high level forms of parallelism from legacy sequential programs, namely, TLP, DLP, PLP and RLP [5, 6, 7, 10].
- ***Software Distribution Heuristics (Chapter 5):*** Together with the heuristics for extraction of parallelism, this thesis also contributes with heuristics for automated accelerator offloading of computationally intensive code regions in heterogeneous systems [11, 14].
- ***Parallel Code Generation (Chapter 6):*** This thesis contributes with code generation techniques, which allow to realize software parallelization and distribution opportunities using state-of-the-art parallel programming paradigms [5, 7, 10, 14]. This enables the applicability of the proposed tool flow to a wide variety of relevant commercial heterogeneous embedded multi-core platforms.
- ***Optimization of Parallel Code (Chapter 6):*** Although the main focus of this work is to optimize sequential applications, this thesis also contributes with techniques to further optimize existing parallel code, in particular, code annotated with OpenMP compiler directives [13]. This input OpenMP code to be further optimized can be either generated by the proposed tool flow, or manually parallelized code.
- ***Applicability to Commercial Platforms (Chapter 7):*** The applicability of the proposed technologies is evaluated on relevant commercial embedded platforms, such as Android devices [5, 10] and multi-core DSP platforms [9, 12, 14].

1.5 Synopsis and Outline

This chapter started by describing the dramatic shift in the paradigm of processor design from the single-core era to the multi-core and heterogeneous eras. To exemplify this in the context of embedded systems, the evolution of multiple commercial families of MPSoCs was presented. The current practice and challenges of parallel programming were also discussed in detail, as well as the need for software parallelization and distribution tools. Finally, a brief overview of the proposed tool flow was given together with a description of the key contributions of this thesis.

The remainder of this thesis is organized as follows. A review on the previous research work relevant to this thesis is presented in Chapter 2. Chapter 3 discusses the details of the Program Model. The proposed software parallelization techniques are presented in Chapter 4, while the software distribution techniques are presented in Chapter 5. The details of the code generation phase are discussed in Chapter 6. The experimental evaluation is detailed in Chapter 7. Finally, the summary, conclusions and the outlook of this thesis are presented in Chapter 8.

Chapter 2

Related Work

Techniques for automated software optimization for multi-core systems are not new. Early works on parallelization focused solely on loop parallelism (also known as DOALL) [155], where each iteration in a loop is independent from the others. Classical examples of this are the Stanford University Intermediate Format (SUIF) [318] and Polaris [33, 34] frameworks, which have been often referred as *first generation tools* [261]. This chapter aims at presenting the features and limitations of existing academic and commercial frameworks for software parallelization and distribution, which are the most relevant to this thesis. The presentation of the frameworks is organized in multiple categories. Some frameworks might simultaneously fall into multiple categories. Therefore, they are either discussed in their most relevant category, or their discussion is divided across multiple categories.

This chapter is structured as follows. Section 2.1 discusses multiple parallelization frameworks driven by profiling information and by the extraction of parallel patterns. Then, the frameworks with support for heterogeneous systems are presented in Section 2.2. Finally, Section 2.3 closes this chapter with a summary of the presented frameworks, which shows in perspective the proposed tool flow in this thesis.

2.1 Software Parallelization

This section presents the most relevant parallelization frameworks to this thesis. First, profile-driven frameworks are discussed, followed by pattern-driven techniques. The coarse-grained parallel patterns considered in this section are DLP, PLP, TLP and RLP.

2.1.1 Profile-Driven Parallelization

The use of dynamic information for software parallelization has been identified as an effective way to overcome the traditional limitations of static techniques when it comes to analyze pointers, dynamic allocated memory, function pointers among others [239]. This information is typically gathered at runtime by means of program instrumentation and profiling [86]. The use of dynamic information has been proposed either as a replacement or a complement to static information to enable a *hybrid analysis* [259]. This section reviews multiple frameworks whose main contribution lies on profiling techniques for software parallelization. It is worth mentioning that these profile-driven frameworks typically focus only on parallelism discovery, without providing facilities for automated parallel code generation.

One of the earliest profile-driven frameworks for software parallelization is Embla [89, 187], which was developed at the Swedish Institute of Computer Science. Embla is a simple tool that records and reports to developers relevant dynamic data dependencies among statements (e.g., Read-After-Write (RAW), Write-After-Write (WAR) and Write-After-Write (WAW)). For this purpose, it uses Valgrind [208] as the Dynamic Binary Instrumentation (DBI) tool. However, this framework fully relies on the developers to manually identify and correctly implement the most promising parallelization opportunities. Alchemist [323] is a framework developed at Purdue University for dependence distance profiling in C and C++ programs also based on Valgrind [208]. This framework works at various language construct granularities (e.g., loops or functions). The primary focus of Alchemist is to provide high-level recommendations about dependencies among the language constructs that might prevent parallelization (i.e., RAW, WAW and WAR). To distinguish among the different instances of a given construct, this framework builds an execution index tree by using a post-dominator analysis [155]. However, this framework does not provide facilities for code generation, which implies still a significant effort for developers.

Prospector [162, 163] is another profile-driven framework for extraction of DLP from the Georgia Institute of Technology. This framework is implemented on top of the Pin tool [186] for DBI. It is based on a profiling technique developed by the same research group that developed a framework called SD^3 [164]. The most interesting aspect of SD^3 is that its main focus is to reduce both the runtime and memory overhead of the profiling process. On the one hand, to optimize the runtime overhead this framework parallelizes the profiling process itself. On the other hand, to reduce the memory overhead SD^3 takes advantage of stride patterns to compress the memory access information; then, it derives the dependency information from the compressed format itself. However, similar to Alchemist, Prospector only provides high-level parallelization hints without any code generation support. The Parwiz [157] framework. jointly developed at INRIA and the Université de Strasbourg, uses DBI built on top of Pin for identifying DLP. This framework aims at different use cases, including identification of parallel loops and transformation of loop nests to enable vectorization. Parwiz achieves this by building an execution tree that contains multiple types of nodes to enable the dependence analysis. The ACCESS nodes are key, as they represent individual memory accesses from which data dependencies are derived. Although Parwiz is able to distinguish dependencies that can be handled with privatization, it is not able to detect reduction operations. Finally, Parwiz incorporates static analysis to reduce the profiling overhead. An important concern of tools based on DBI, such as Embla, Alchemist, Prospector and Parwiz, is the degree of accuracy at which they can provide high-level readable information to developers [140].

Profiling technologies from the single-core era have also inspired frameworks for software parallelization. Kremlin [96] is one example inspired by `gprof` [118], which was developed at the University of California, San Diego. This framework aims at helping developers for both parallelism discovery and planning (i.e., implementation). The key contribution of this framework is the introduction of a Hierarchical Critical Path Analysis (HCPA), as an extension to the traditional Critial Path Analysis (CPA) [154]. The goal of the HCPA is to model dependencies across nested regions in a program (e.g., nested loops) to enable the extraction of DLP. Along with the HCPA, this framework introduces a metric to quantify the parallelization potential of a given region called *self-parallelism*, which is inspired by the *self-time* metric used in traditional profilers such as `gprof`. In addition, Kremlin includes facilities for parallelism planning, which provide suggestions on how to parallelize the programs with OpenMP. However, developers still have the challenging task to manually refactor and implement the parallelism. Kismet [141] is a tool built on top of Kremlin that aims at providing speedup estimates for the parallelization opportunities identified in sequential programs. This framework has two major components: *(i)* a self-parallelism profiler, which extends the one introduced in Kremlin and *(ii)* a speedup predictor, which is the main contribution of Kismet. To estimate the parallel speedup, the tool makes use of platform-independent parallelization information provided by the self-parallelism profiler and platform specific details, such as number of available cores and parallelization overhead. The authors of Kismet made clear that this tool does not provide suggestions on how to refactor the program for parallelization. Then, this task is left to developers, which is an error-prone and time-consuming process, as discussed in Chapter 1.

Profile-driven tools have also emerged in the industry. Threading Advisor is a tool for thread design and prototyping included in the Intel Advisor Tool Suite [135]. This tool is structured as workflows in which developers have to follow multiple steps. The Threading Advisor workflow starts with the survey step, where it profiles the input program to identify hotspots. Then, with this information developers are required to add annotations in code sections that they consider good candidates for parallelization. This implies that the actual extraction of parallelism is performed manually by developers. Afterwards, it follows the suitability step, where using the annotations the tool is able to estimate the scalability of the parallelization opportunities by using different number of threads. Finally, Threading Advisor performs a dependence analysis to identify potential data races and deadlocks. The main drawback of this tool is that it requires significant manual intervention by the developers, as it neither automatically identifies parallel patterns nor provides code generation facilities. Prism [75] is an example of a commercial tool based on DBI, which was developed by Critical Blue. Prism profiles the application to obtain data dependency and performance characterizations. Using this information, it is possible to detect hotspots, identify data dependencies and study the parallelization scalability based on a *what-if* analysis. However, Prism requires that developers manually suggest the potential parallelism. Additionally, this tool provides other facilities beyond multicore optimization, including binary translation, cache optimization and software security. Similar to this thesis, Prism has been applied to Android, but only for cache optimization [74]. Intel Thread Advisor and Prism are similar tools in terms of their workflow and the analyses provided. However, they heavily rely on insights provided by developers due to the interactive approach of this tools, i.e., they do not automatically extract parallel patterns from the sequential code, nor generate the parallel code.

2.1.2 Pattern-Driven Parallelization

The use of design patterns is a widely accepted practice for software development in which recurring problems in a given context are solved by reusing well-known solutions [95]. This approach has been also applied to parallel programming. The taxonomy known as Our Pattern Language (OPL) organizes parallel patterns in multiple abstraction layers [195]. In particular, the *parallel algorithm strategy patterns* layer is relevant for software parallelization, since it defines multiple patterns that can be extracted from sequential programs. The most prominent patterns considered in this thesis are Data Level Parallelism (DLP), Pipeline Level Parallelism (PLP), Task Level Parallelism (TLP) and Recursive Level Parallelism (RLP). Extraction of patterns has been identified as an effective strategy for software parallelization since early works, such as the framework called Parallelize Automatically by Pattern Matching (PARAMAT) [156]. This section reviews frameworks relevant for this thesis whose main contribution lies on the automatic identification of parallel patterns from sequential code. Firstly, frameworks that focus on one specific pattern are presented, followed by frameworks that provide support for multiple patterns.

2.1.2.1 Data Level Parallelism (DLP)

DLP is one of the most scalable parallel patterns [196], which is typically found in scientific and multimedia applications. It is defined as a pattern where a data set is split into smaller blocks to which the same computation is simultaneously applied by multiple parallel tasks. DLP is one of the most studied patterns in the domain of software parallelization tools. This thesis also proposes an approach for extraction of DLP, which is described in Section 4.2. In the following, the most relevant frameworks for the extraction of DLP are discussed.

Cetus [77], developed at Purdue University, is a source-to-source compiler infrastructure written in Java with support for building automatic parallelization tools. This framework provides three fundamental fully static analyses for loop parallelization (i.e., extraction of DLP), namely variable privatization, reduction variable recognition and induction variable substitution. In addition to the parallelization support, Cetus includes other general compiler facilities, including array section and points-to analyses. Moreover, Cetus provides code generation facilities, which annotate the outermost parallel loops with OpenMP pragmas. A similar source-to-source parallelizing compiler is autoPar [179], which is built on top of the ROSE compiler infrastructure [250], developed at the Lawrence Livermore National Laboratory. This is a fully static framework that focus on array-based loops. autoPar first normalizes and identifies candidate loops for parallelization. Then, for each candidate it performs the following steps: liveness and dependence analyses, classification of loop variables (i.e., auto-scoping), elimination of dependencies associated with the auto-scoped variables, and finally insertion of OpenMP pragmas. In addition, autoPar provides supports to parallelize loops in C++ using the Standard Template Library (STL) [72]. However, autoPar do not perform any cost-benefit analysis to reason about the benefit of parallelizing a given loop. Overall the main concern about Cetus and autoPar is that they rely on fully static analysis, which presents important limitations for software parallelization, as it was discussed in Section 2.1.1. Instead, Tournavitis et al. [296] presented a profile-driven holistic approach for extraction of DLP, which was developed at the University of Edinburgh. This approach makes use of static and dynamic analyses to identify control and data dependencies, which are implemented on top of the Compiler System (CoSy) framework [22]. Then, profitable parallel loops are identified using machine learning, based on both static and dynamic program features (e.g., instruction count and memory accesses). Finally, the selected loops are annotated with OpenMP pragmas, including the scheduling policy. However, the approach used to select the OpenMP scheduling policy does not take load balancing into account. This thesis addresses the selection of the OpenMP scheduling policy to achieve a proper load balancing, as it is discussed in Section 6.3.3.

A popular static approach for extraction of DLP is the use of the *polyhedral model* (also known as *polytope model*) [172]. This model is a mathematical framework for transformation, parallelization and data locality optimization of loop nests. The applicability of the polyhedral model is limited to a subset of statically predictable loop nests known as Static Control Parts (SCoP) (also called Static Affine Nest Loop Programs (SANLP)). A SCoP consists of a set of statements enclosed in loops in which data dependencies have to be statically computable, and where loops bounds, array accesses and expressions in conditions must be affine expressions of the enclosing loops. These restrictions significantly limit the applicability of the polyhedral model. Nevertheless, multiple frameworks rely on this model and there are works that have proposed approaches to relax its restrictions to some extent [31]. PLUTO [38, 39] is a source-to-source parallelizing compiler, jointly developed by Ohio and Louisiana State Universities, which is based on the polyhedral model. The main focus of this tool is the parallelization and locality optimization of affine nested loops. The analysis in PLUTO is enabled by polyhedral libraries, such as Integer Set Library (ISL) [304] and PolyLib [316]. OpenMP code generation facilities are also provided in PLUTO.

Par4All [237] is a source-to-source compiler based on the polyhedral model jointly developed by SILKAN, MINES ParisTech and Institut TÉLÉCOM/TÉLÉCOM Bretagne/HPCAS. Par4All is built on top of the Parallelization Infrastructure for Parallel Systems (PIPS). Initially, this framework supported OpenMP code generation, and later it was extended to CUDA and OpenCL to address the heterogeneous era. However, the development of this framework is not active anymore. Polly [120, 293] is a recent framework for IR-level polyhedral optimizations jointly developed by INRIA and University of Passau. This framework eventually became part of the Low Level Virtual Machine (LLVM) compiler infrastructure and it is currently supported by an active community of developers. Polly was inspired by a former polyhedral framework called GRAPHITE [242], which was integrated in GNU Compiler Collection (GCC), being one of the earliest efforts to incorporate polyhedral analyses and transformations into production compilers. Polly provides facilities for loop optimization by means of coarse-grained and fined-grained parallelization (i.e., vectorization), and for data locality optimizations by means of loop transformations. In addition, support for GPGPU was recently incorporated [61]. It is worth mentioning that the community behind Polly has made important efforts to relax the traditional limitations of the polyhedral model (e.g., supporting reduction operations [79]); thus, improving the applicability of this framework. Polly is a complementary framework to the work done in this thesis. Therefore, its static analysis facilities have been integrated in the proposed tool flow, as discussed in Section 4.2.2.1.

The polyhedral model has also been used to derive parallel data-flow MoCs from SANLP in C streaming applications [292]. PNgen [306] is a tool for this purpose, which is part of the Daedalus design flow developed at Leiden University [274]. This tool transforms SANLPs into the Polyhedral Process Network (PPN) MoC [305], which is a network of processes that exchange data tokens through First-In First-Out (FIFO) channels. The PPN MoC is a special case of KPN that is statically analyzable and allows to perform algebraic transformations according to the polyhedral model. The resultant PPN is described in the Extensible Markup Language (XML) format, which is further processed by the rest of the Daedalus design flow.

`While` loops have been also identified as a challenge for software parallelization, since the iteration space is unknown at compile time. Early works on `while` loop parallelization focused on generalizing the polyhedral model to support loop nests that contain this type of loops. Lengauer et al. [171] proposed at the University of Passau a conservative extension to the polyhedral model in which the execution space and the termination condition are precisely scanned at runtime. Rauchwerger et al. [253] at the University of Illinois at Urbana-Chapaign proposed a speculative technique targeting loops containing linked lists. The main drawback of the previous approaches is their runtime overhead, which limits the effectiveness of these techniques. Parallelization of `while` loops is also considered in this thesis, as discussed in Chapter 3.

In the commercial domain it is also possible to find tools that aim at helping developers to identify DLP exploitation opportunities. Parallware [19] is a tool based on LLVM to assist in parallelization of scientific applications developed by Appentra. The technology behind this tool was originally developed at the University of A Coruña [20]. The parallelization approach of this framework is based on a hierarchical classification in which first the code is split into small kernels, and then the data dependencies among kernels are analyzed to identify parallelism. Parallware leverages multiple classical static compiler analyses, including array analysis, variable scoping and interprocedural analysis. This tool parallelizes loops either with OpenMP or OpenACC. The main drawback of Parallware is that it requires manual code refactoring to make it manageable by this tool, e.g., by removing global variables, structs and enums [188]. Compaan Hotspot Parallelizer [62] is another commercial tool developed by Compaan Design, which is a spin-off of Leiden University. This tool is built on top of the CoSy compiler framework [22, 198, 202]. Compaan derives a KPN specification from SANLPs in C programs using the polyhedral model. Then, the KPN specification is mapped on platforms with GPPs and FPGAs using POSIX Threads (Pthreads) and the VHSIC Hardware Description Language (VHDL), respectively. However, Compaan does not use performance information to evaluate the potential of the parallelization opportunities. It is worth mentioning that the original academic version of the tool takes MATLAB programs as an input [161, 275].

2.1.2.2 Pipeline Level Parallelism (PLP)

In PLP a given computation within a loop body is broken into a sequence of processes (i.e., *pipeline stages*), which follow a *producer-consumer* relationship. This is a relevant pattern in embedded systems, since many applications in this domain present a streaming-based structure [291]. In these applications, there are serially dependent tasks, such as audio and video encoding and decoding. An interesting characteristic of PLP is that it can be applied to loops with loop-carried dependencies, which prevent the exploitation of DLP. Therefore, PLP complements DLP for loop parallelization. This thesis also addresses the extraction of PLP in Section 4.3. This section provides an overview of existing approaches for extraction of PLP.

One of the earliest efforts to exploit PLP in loops is an approach called Decoupled Software Pipelining (DSWP), which was introduced by Rangan et al. [252] at Princeton University. DSWP specifically targets the optimization of Recursive Data Structures (RDS), such as linked lists, trees and graphs. It works by dividing RDS loops into threads for the traversal code (critical path) and for the actual computation (off-critical path). Then, these threads execute in parallel in a pipelined fashion. Ottoni et al. [232] proposed an approach to automatically extract DSWP. This approach is based on a clustering algorithm that tries to find Strongly Connected Components (SCCs) in a Program Dependence Graph (PDG) [91], which represents the program at the low-level instruction granularity. The proposed heuristic tries to balance the pipeline stages by estimating the cycles of each SCC. The approach is implemented on top of the back-end of the Illinois Microarchitecture Project using Algorithms and Compiler Technology (IMPACT) framework [24]. Vachharajani et al. [298] proposed an extension to DSWP to support Thread Level Speculation (TLS). The idea of this approach is to improve load balancing by speculating over infrequent dependencies to avoid restricting instructions to one single thread. Raman et al. [251] further extended DSWP by introducing a technique called Parallel Stage Decoupled Software Pipelining (PS-DSWP). The goal of this technique is to improve the scalability of DSWP by replicating pipeline stages with no loop-carried dependencies to exploit DLP. The previous DSWP approaches are integrated in a compiler framework called VELOCITY [43]. In addition, further improvements and frameworks based on DSWP have been proposed [130, 175, 181]. In contrast to VELOCITY, the approach presented in this thesis for extraction of PLP is not limited to RDS.

Parallelization approaches that require significant involvement by developers have been also proposed for PLP extraction. Thies et al. [291] presented an approach to extract PLP based on a semi-automatic profiling technique. In this approach, developers have to manually group and annotate statements in pipeline stages (similar to Intel Threading Advisor [135] discussed in Section 2.1.1). Then, using dynamic analysis, it is possible to build a stream graph, which is presented to developers to understand the performance of the current pipeline configuration. If this configuration is not satisfactory, developers have to iteratively refine the boundaries of the pipeline stages. A similar annotation-based approach called Paralax [299], which was developed at Queen's University of Belfast and Ghent University. The idea behind Paralax is that developers can help compilers to close semantic gaps by providing annotations with information that can not be inferred statically, such as function properties, memory access and liveness information of variables and data structures. Using these annotations Paralax first performs a dependency analysis and then extracts pipeline configurations from only outermost loops using a DSWP algorithm. The resulting pipeline configurations are implemented using Pthreads. However, the main disadvantage of the previous two approaches is that the extraction of PLP is not automated. Tournavitis [295] proposed a semi-automatic profile-driven approach for PLP identification at the University of Edinburgh. This approach performs a hierarchical extraction of PLP that allows to identify pipeline configurations that span multiple levels in a loop nest. Additionally, pipeline stages without inner loop-carried dependencies are replicated in a similar fashion to PS-DSWP [251]. This hierarchical pipeline extraction technique operates on the PDG [91], and it is implemented on top of the CoSy framework [22]. The code generation takes place directly in the IR of CoSy and the execution is enabled by a dedicated runtime system.

Geuns et al. [98] proposed a method for parallelizing `while` loops jointly developed by Eindhoven University of Technology, NXP Semiconductors and University of Twente. This approach consists in creating one task for each function within the loop body. The communication among tasks is performed through circular buffers with overlapping windows. Besides being restricted to `while` loops with function calls, one important limitation of this approach is that programs have to be written in the Single Assigment (SA) form [17]. This implies an additional effort for developers, as they have to manually perform data dependency analysis first and then source code transformations to the SA form. Cordes [66] proposed a method at TU Dortmund for PLP extraction in embedded systems based on Integer Linear Programming (ILP). The ILP formulation extracts pipeline configurations from the PDG [91], which is augmented with performance information used to control the granularity of the stages. This approach is implemented on top of the MACC framework [245] and it uses MPSoC Parallelization Assist (MPA) [28] as the code generator, which also provides a dedicated runtime system to create tasks and synchronization primitives. Cordes later proposed a multi-objective PLP extraction method based on Genetic Algorithms (GAs) [63]. The idea of this approach is to optimize sequential applications not only for execution time but also for energy or communication overhead. Unfortunately, these techniques proposed by Cordes were not evaluated on realistic commercial embedded platforms using state-of-the-art parallel programming paradigms. Furthermore, the main drawback of these approaches is their long analysis times, which might limit the applicability of these techniques in large production programs. Unlike these ILP and GA based approaches, in this thesis it was opted for effective faster heuristics. Moreover, the proposed tool flow was evaluated on commercial embedded platforms, e.g., Android devices.

2.1.2.3 Task Level Parallelism (TLP)

In contrast to loop parallelism, TLP is a more irregular pattern in which multiple tasks perform different computations on the same or on different data sets. Typically, these tasks are composed by basic blocks, language constructs (i.e., independent `if` blocks or loops executed in parallel) or function calls. Therefore, the extraction of TLP deals with multiple granularities. This pattern is also considered in this thesis, as Section 4.4 describes. The rest of this section discusses the most relevant approaches to this thesis for extraction of TLP.

Cordes et at. [68] proposed an automatic TLP extraction approach for embedded systems that is based on ILP. In this approach, an augmented version of the Hierarchical Tasks Graph (HTG) [100] is used as the IR, which allows to model the communication between the multiple levels of hierarchy by adding extra nodes for this purpose. This extension to the HTG makes this IR more suitable for hierarchical parallelization techniques. The ILP technique proposed by Cordes operates on the augmented HTG to control the granularity of the generated tasks. However, one disadvantage of the HTG is that it can not model unstructured code (i.e., it contains `break`, `return` or `goto` statements). This thesis also takes advantage of the program hierarchy using another IR called Dependence Flow Graph (DFG) [145, 241], which is also able to model unstructured code, as discussed in Section 3.4. Cordes et at. [67] later proposed a multi-objective TLP extraction approach based on GA. However, the main concern of ILP and GA based approaches is their long analysis times, as discussed in the previous section.

The extraction of TLP has been also studied in the automotive domain. Kehr et al. [151, 152, 236] proposed approaches for extraction of TLP at multiple granularities in legacy automotive software described in Automotive Open System Architecture (AUTOSAR) [25]. This work was jointly developed by DENSO Automotive, Barcelona Supercomputing Center and Ilmenau University of Technology. AUTOSAR is a standard for the software architecture of automotive applications. In this standard, applications are described as a set of elementary code sections called *runnables* and a set of tasks, which in turn are clusters of runnables. In [236], the authors first proposed a framework called RunPar to explore the parallelization at runnable level. RunPar allows to map runnables within a single task to multi-core Electronic Control Units (ECUs). RunPar relies on a static timing analysis tool called Open Toolbox for Adaptive Worst-Case Execution Time Analysis (OTAWA) [29]. This tool provides the Worst-Case Execution Time (WCET) estimates required to make the mapping decisions. Later, the authors explored the parallelization of AUTOSAR applications at the task level [152]. This was achieved by introducing a new concept called Timed Implicit Communication (TIC), which allows to decouple the task communication between producers and consumers. Subsequently, Kehr et al. [151] proposed a new concept called *SuperTask* with the aim of maximizing the amount of runnable level parallelism in AUTOSAR applications. Compared the previous approaches, the techniques proposed in this thesis aim at a general applicability without being restricted to one particular application domain.

2.1.2.4 Recursive Level Parallelism (RLP)

Divide-and-Conquer (DaC) algorithms are an important class of design paradigms, with a high degree of parallelization potential used to solve a vast set of problems in multiple application domains [195, 272]. These algorithms are typically implemented in programs with multiple recursion in which functions contain two or more self-invocations. This implementation strategy allows to recursively break problems into smaller coordinated sub-problems that are easier to solve. Provided that these sub-problems are independent, it is possible to exploit a scalable form of nested parallelism called RLP. Therefore, multiple research efforts have been directed towards exploiting parallelism from this class of programs in terms of language support, runtime systems and compilers. Recent examples of research works for language support and runtime systems are C++11 templates for DaC algorithms [76] and runtime techniques to control the task granularity [92], respectively. This thesis concerns compiler technologies for automated extraction of RLP, as discussed in Section 4.5. The rest of this section reviews relevant frameworks to extract RLP from sequential code.

Multiple research efforts have addressed the parallelization of recursive programs. The compiler proposed by Rugina et al. [257] is an early work on the parallelization of DaC algorithms developed at Massachusetts Institute of Technology (MIT). This approach exploits the fact that DaC algorithms can be decomposed in sub-problems that access disjoint array regions. This framework relies on multiple analyses from the SUIF compiler infrastructure [318] to statically reason about the mutual independence of recursive call-sites. Finally, this framework generates parallel code using the Cilk programming paradigm [35], which was originally developed at MIT and later acquired by Intel [137]. Gupta et al. [121] proposed an automatic parallelization approach of recursive procedures developed by IBM and Mobious Management Systems. In this approach, compile time analysis is complemented by a runtime system to perform a speculative parallelization to address spurious data dependencies. The approach is build on top of the Toronto Portable Optimizer (TPO) [166]. However, the authors do not discuss the overhead introduced by the runtime speculation that could limit the effectiveness of the approach. AutopaR [147] is another source-to-source framework for RLP extraction developed at Bilkent University. This tool performs static analysis on GCC and its main goal is to parallelize recursive calls in C code with OpenMP pragmas using parallel sections. However, the authors do not clarify how the mutual independence of the recursive call site is verified. Furthermore, AutopaR does not address the selection of a proper task granularity, which is fundamental to achieve a profitable RLP extraction [2, 92].

Interactive frameworks that require important user intervention for parallelizing DaC programs have been also proposed. The source-to-source compiler called Recursive Programs Automatically Parallelized (REAPAR) [244] is one example, which was developed by abaXX Technology and GINIT. In this framework the independence of the recursive call-sites is assumed and not verified, leaving this challenging responsibility to developers. REAPAR performs a profiling run of the input program to collect information that allows to select a proper parallelization strategy. Then, a code generator based on a Perl script that uses pattern matching to add the threading code. Unfortunately, besides the lack of dependency analysis, REAPAR imposes multiple restrictions to the input source code that it can handle, which further limits its applicability. In addition, the robustness of the Perl-based code generator is not clear, in contrast to a typical compiler frontend. Huckleberry [60] is a framework for parallel code generation from recursive programs targeting distributed memory multi-core systems, which was developed at Columbia University. The input programs accepted by this framework must be written using an API called *partition*, which implies an important manual effort when existing recursive code has to be ported to this API. Then, Huckleberry takes the input specification together with a model of the platform to generate the parallel code using the Cell Software Development Kit (SDK) to target the QS20 Cell Blade platform [207]. Similarly, Ariadne [193] is a framework in which the developer has to insert directives to tell the compiler where and how to parallelize recursive programs. This framework was jointly developed by the University of Ioannina and ETH Zurich. Ariadne produces parallel code in Pthreads, OpenMP, Cilk or in a model called Self-adaptive Virtual Processor (SVP) [143]. Compared to the previous works, the approach proposed in this thesis for RLP extraction, automatically verifies mutual independence of recursive call-sites and performs a profitability analysis to select a proper parallelization strategy to achieve a good load balancing and a low task management overhead.

2.1.2.5 Multiple Patterns

In general, DLP has been one of the most studied patterns in the domain of frameworks for software parallelization. However, as discussed in the previous sections, other prominent parallel patterns can be extracted from sequential code to maximize its optimization opportunities for multi-core systems. Therefore, this section reviews parallelization approaches that consider more than one parallel pattern.

One example of these frameworks is the MPSoC Application Programming Studio (MAPS) [4, 51, 52, 53, 269], developed at the RWTH Aachen University. This is the most relevant work to this thesis, as the tool flow proposed in this thesis originates from the MAPS project. This framework was introduced by Ceng et at. [53, 54] based on a clustering algorithm to extract TLP from sequential code called Constrained Agglomerative Hierarchical Clustering (CACH). The granularity at which the CACH algorithm extracts tasks is called Coupled Block (CB), which is a schedulable and tightly coupled code section. This algorithm operates on an IR called Weighted Statement Control Data Flow Graphs (WSCDFG), which is built using static and dynamic analysis. The WSCDFG also includes performance information annotations. Unfortunately, the performance estimation technique used in this framework is based on a very simple and inaccurate approach that relies on cost tables provided by a platform model. In addition, parallel code generation has to be manually performed by developers. Later, Castrillon [51, 52] proposed extensions to MAPS that lead to evolve this framework into two main tool flows: a *sequential flow* for parallelism extraction and a *parallel flow* for mapping KPN applications. The sequential flow is the one relevant to this thesis. This tool flow is an extension to the initial parallelization tool flow proposed in [53] in which pattern-driven parallelization heuristics were incorporated to extract TLP, DLP and PLP. Also, a limited experimental code generator was added just for verification purposes to emit a combination of Pthread and Message Passing Interface (MPI) code, together with hints to help in the manual migration of the input C program into a CPN parallel specification [269]. However, the proposed pattern-driven heuristics present important limitations that impact their effectiveness. The DLP is not able to identify private variables and reduction patterns present in many loops, which results in missing many profitable optimization opportunities. In addition, the arrays are seen by the DLP heuristic as monolithic objects for which it is not possible to analyze their iteration space in detail. Therefore, it is only possible to apply this heuristic to trivial loops. Regarding the PLP heuristic, this technique is not able to exploit multi-level pipelines, nor stage replication. Finally, the TLP heuristic is based on a simple As Soon As Possible (ASAP) scheduling, which only operates on the first level of the functions (i.e., it is not able to extract TLP in nested code regions). Furthermore, the sequential flow of MAPS is not able to decide which core type is more suitable for a given code region (i.e., is not able to optimize code for heterogeneous platforms). Finally, this framework was only evaluated with synthetic benchmarks and platforms. Compared to the sequential flow of MAPS, in this thesis more effective heuristics are proposed; new patterns are targeted; an accurate multi-grained performance estimation approach is proposed; heterogeneous platforms are supported by means of heuristics for automatic accelerator offloading; and automatic code generation facilities are provided for multiple programming paradigms that allow to apply the proposed tool flow to a wide range of commercial embedded devices.

Edler Von Koch [84] proposed an approach for the detection of algorithmic skeletons in sequential code developed at the University of Edinburgh. Algorithmic skeletons [59] are concrete implementations for a given domain, language, model or platform that enable the realization of high-level parallel patterns (e.g., DLP and TLP) [103, 262]. The approach is based on the commutativity analysis of code regions to detect algorithmic skeletons. The key idea behind this approach is to reorder code regions (e.g., function calls, loops, among others) and verify if the output of the program is still correct. If this is case, the regions are commutative, and therefore concurrent. However, the commutativity property does not guarantee parallelism and it can not be applied to extract PLP. The skeleton detection is based on two phases: *(i)* the skeleton candidates are statically detected and *(ii)* the commutativity of the regions is evaluated at runtime. The proposed approach only produces a report of the detected commutative regions. It is left to developers or subsequent tools to make conclusions about the actual parallelism opportunities and to generate the parallel code. Besides the previous limitations, the authors acknowledge that dynamic community testing of code regions is to some extend a brute-force approach, which implies various issues. One fundamental issue is the risk of combinatorial explosion due to the potentially large number of possible code permutations. In addition, compared to this approach, in this thesis the focus is on detecting high level parallel patterns rather than on specific pattern implementations.

Sambamba [278] is a framework for on-line adaptive parallelization developed at Saarland University. This framework is divided into two components: a compile time part that finds the best parallelization candidate for each function in a program using PDG [91] as the IR and ILP as the analysis approach; and a runtime part that continuously collects dynamic information, such as the load of the system and the utilization of the task queues, to adaptively decide which version of each function to execute (i.e., sequential or parallel). The parallel version is speculatively executed. Sambamba accepts C and C++ programs, implicitly exploits DLP and TLP and is implemented on top of LLVM. Just-In-Time (JIT) is used to compile and attach the selected version to a running program. Sambamba presents various limitations: it is based on a flow-insensitive analysis that results in inaccuracies when verifying the independence of memory accesses; the dependence analysis used in this framework can not handle properly regular data structures (e.g., arrays) and recursive functions.

DiscoPop [176, 177] is a parallelization framework for homogeneous platforms jointly developed by the German Research School for Simulation Sciences, RWTH Aachen University and TU Darmstadt. This framework is divided into three phases. In the first phase, information about control and data flow of the program is extracted using static and dynamic analyses. Then, during the second phase parallelism is extracted using the concept of Computational Units (CUs) as the minimum granularity for building tasks, which is a similar concept to CBs introduced in the MAPS framework [53] previously discussed. A CU is a set of instructions that follow a read-compute-write pattern. DiscoPop builds a CU graph using the concept of CUs and the information about data dependencies among them. Based on the CU graph, this framework applies techniques (e.g., SCCs) to extract multiple forms of parallelism. Finally, DiscoPop generates a report in which the parallelization opportunities are ranked using instruction coverage (a simplistic performance model), local speedup, and CU imbalance as metrics. However, it is the responsibility of the developers to interpret the report and implement the parallel code. Compared to DiscoPop, the tool flow proposed in this thesis evaluates the parallelization opportunities not only locally but also globally on the scale of the whole program. In addition, the approach proposed in thesis supports heterogeneous platforms and provides facilities for automatic code generation.

One example of a commercial tool supporting multiple patterns is Pareon [301] from Vector Fabrics. This tool follows an interactive approach similar to the Threading Advisor from Intel. Pareon is based on a three-step process: *(i)* insight, *(ii)* investigate and *(iii)* implement. In the insight step information about performance, dependencies and memory accesses in C/C++ programs is presented to the developers. Then, the parallelization opportunities following multiple patterns and its performance impact are interactively identified by the developers. Finally, during the implementation step, Pareon provides *recipes* to help the developers in the process of manually parallelizing the programs. Pareon is complemented with a C-based library called vfTasks [302], which allows to implement parallel tasks. Unfortunately, Pareon is not commercially available anymore [87].

2.2 Software Distribution

The need for specialization in the embedded domain motivated the introduction of heterogeneous platforms in which computationally intensive workloads can be more efficiently processed by dedicated cores. However, heterogeneity further increases the programming complexity of embedded systems. While software parallelization in the multi-core era is still an open research issue as previously discussed in this chapter, in the heterogeneous era new frameworks and techniques for software distribution are required to address the introduced challenges. In the embedded domain, one form of software distribution is the *mapping and scheduling* of parallel dataflow MoCs, which has been an active research area for many years. In dataflow MoCs, programs are described as a network of processes that communicate through FIFO channels. Two prominent examples of these MoCs are KPN [99] and SDF [168]. There is a multitude of frameworks for mapping and scheduling of dataflow MoCs [27, 48, 50, 51, 52, 101, 238, 240, 290]. However, these frameworks assume that the input program is already parallelized in one of these dataflow MoCs. Techniques for mapping and scheduling of dataflow MoCs are out of the scope of this thesis. Instead, the focus of the proposed tool flow here is to provide general techniques for software distribution starting from sequential programs, which is a less studied area. The approach proposed in this thesis for software distribution is presented in Chapter 5. The rest of this section describes relevant approaches for software distribution.

Cordes [70] proposed parallelization approaches to exploit PLP and TLP on heterogeneous MPSoCs, which were developed at TU Dortmund. This author proposed two different approaches for extraction of PLP for heterogeneous systems [64, 65]. One is a single-objective approach based on ILP in which the program is modeled using the PDG IR [91]. The other one is a multi-objective approach based on GAs in which the program is also described using the PDG IR. Similarly, Cordes [69] proposed a single-objective and a multi-objective approach for exploitation of TLP on heterogeneous platforms based on ILP and GAs, respectively. As previously discussed, the major concern of this approach is their long execution time, which limits their applicability. Moreover, their evaluation was performed on a synthetic platform, which is based on processors of the same ISA but running at different frequencies.

Multiple speculative approaches have been also proposed to exploit DLP on GPUs. Paragon [264] is a framework to run possibly data parallel loops in sequential programs on GPUs, jointly developed by the University of Michigan and Microsoft Research. This framework is divided into one offline static compilation phase and one runtime kernel management phase. During the offline phase possible data-parallel loops are identified and CUDA code is generated for them. Then, during the runtime phase the candidate loops are speculatively executed on a GPU using a kernel management unit. If a data dependency violation is detected at runtime, then the execution of the loop is transferred to the Central Processing Unit (CPU). Similarly, Wang et al. [310] proposed a tool flow for speculative loop execution on GPUs, which was developed by the Lancaster University and University of Edinburgh. This tool flow is divided into a compile time phase and a runtime phase. At compile time potentially parallel loops are detected using static and offline profiling dependence analyses. For the detected candidates OpenCL code is generated. Then, at runtime, data dependencies are checked to detect violations. A competitive scheduling scheme is used to recover from dependence violations in which a sequential version of the program is simultaneously executed on a single CPU. If a violation is detected, the speculative version on the GPU is aborted and the final result is taken from the sequential version running on the CPU. However, the previous speculative frameworks present various limitations. During the detection of loop candidates these frameworks do not perform a cost-benefit analysis to guarantee a profitable execution on GPUs. Moreover, the offloading overhead is not considered to select candidate loops. The authors in [310] confirm this argument, since they report slowdowns using the previous frameworks. In addition, another disadvantage of speculation based approaches is their associated runtime overhead. Wang et al. [310] report a speculation overhead from 15% to 60%, with an average of 28% across the benchmarks considered. Compared to the previous frameworks, the software distribution techniques proposed in this thesis perform cost-benefit analyses to ensure a profitable execution on heterogeneous systems. Furthermore, these techniques perform off-line analysis, and thus avoiding expensive runtime overheads.

Optimally Scheduled Advanced Multiprocessor (OSCAR) [125, 150] is a parallelizing compiler for low power multi-core systems developed at Waseda University, which has been deployed in the industry [231]. The key idea behind OSCAR is to decompose a program into coarse grained code regions called *macro-tasks* (e.g., basic blocks, loops or functions) from which a graph is built, which is later analyzed in order to discover DLP or TLP. For the identified parallelization opportunities, OSCAR generates an intermediate parallel code in the so-called OSCAR API, which is in turn translated into runtime library calls (e.g., Pthreads) or into OpenMP directives. In addition, this compiler also takes advantage of the idle times to reduce power using techniques, such as clock gating and Dynamic Voltage and Frequency Scaling (DVFS). OSCAR was initially developed for homogeneous SMPs [150] and then it was extended to target heterogeneous systems [125]. However, OSCAR presents two important limitations in terms of productivity: *(i)* the input code must be manually re-written in *Parallelizable C* [192], which is a set of coding rules to make the code friendly to the compiler and *(ii)* for heterogeneous platforms developers have to manually insert hint directives to instruct the compiler to which accelerator a given code region should be offloaded. In contrast to OSCAR, the tool flow proposed in this thesis does not require any type of code refactoring of the input program and for heterogeneous platforms is able to automatically select a proper accelerator for a given code region.

It is worth mentioning that multiple frameworks have been proposed to migrate parallel code written for homogeneous systems (e.g. OpenMP) into heterogeneous systems. OpenMP extended for CUDA (OpenMPC) [170] is a framework to translate OpenMP into CUDA. The main goal of this framework is to provide a programming interface that abstracts the complexity of CUDA using high-level OpenMP compiler directives. For this purpose, OpenMPC proposes additional directives and environment variables to extend OpenMP for CUDA-specific optimizations. Similarly, Wang et al. [309] proposed a framework to translate existing OpenMP into OpenCL to target GPU based platforms. This approach makes use of machine learning to select loops that are good candidates for GPU offloading and loops that should stay parallelized with OpenMP on the host multi-cores. Unlike these approaches where developers need to identify software parallelization and distribution opportunities, the tool flow proposed in this thesis performs these optimizations without user assistance.

2.3 Synopsis

This chapter presented a review of related work in the area of automatic software parallelization and distribution. A summary of frameworks and approaches that take as input sequential programs is presented in Table 2.1, which considers multiple aspects and features to compare them. The first aspect is whether the tool is academic or commercial. As previously discussed, some of the frameworks considered in this chapter started as academic projects and then eventually evolved into commercial products. The *domain* to which the frameworks are targeting is also presented. The column *basis* refers to the framework upon which each approach is built. Typically, the basis is a well-established compiler framework (e.g., LLVM) or a profiler (e.g., Valgrind). The column *platform model* indicates whether or not frameworks use a model of the target platform to tailor the optimizations. The column *program analysis* refers to the approach used to extract information about programs, which could be static, dynamic or hybrid. The table also presents whether or not frameworks use *performance information* to perform a cost-benefit analysis of the identified optimization opportunities to assess its potential. The column *main approach* describes the key method used to discover parallel patterns and to distribute code regions on heterogeneous platforms. In addition, the table presents which frameworks have support for heterogeneous platforms. Finally, the last column enumerates the programming paradigms for which frameworks are able to generate parallel code either automatically or semi-automatically by means of high-level hints for developers. However, some frameworks do not provide code generation facilities, then they are marked with *none* in this column. Based on the discussion presented in this chapter and the summary presented in Table 2.1 it is possible to draw multiple conclusions regarding existing work on software parallelization and distribution of legacy sequential code for heterogeneous multi-core systems:

- The majority of the frameworks target the HPC domain and thus do not consider particular characteristics of embedded devices. Moreover, only few frameworks make use of a platform model to tailor the software parallelization and distribution optimizations to a particular target system.
- Multiple approaches impose strict restrictions on the input source code to make it easier to handle. However, this strongly limits the applicability of these approaches. Furthermore, in some cases this implies an error-prone manual refactoring of the sequential programs to enable them for a given framework.

- The use of dynamic information for program analysis is a widely accepted technique to replace or complement static information, which is more conservative and presents important limitations particularly for software parallelization. Dynamic analysis enables more optimistic and effective parallelization approaches.
- Few frameworks use accurate performance information to enable a cost-benefit analysis to assess the potential of the discovered optimization opportunities.
- The majority of the frameworks focus on one specific parallel pattern (e.g., DLP). However, software parallelization frameworks should explore multiple forms of parallelism to increase its applicability and effectiveness.
- Although we are currently in the heterogeneous era as discussed in Chapter 1, the majority of the frameworks focus only software parallelization approaches targeting homogeneous platforms. Moreover, the few frameworks that address heterogeneity focus only on GPUs. However, in the embedded domain frameworks should be able to support an increasing diversity of processing elements beyond GPUs (e.g., DSPs).
- There is still a large number of frameworks that follow interactive and semi-automatic approaches that leave significant productivity gaps in the process of software parallelization and distribution, as they have to be addressed manually by developers. These gaps include manually extracting parallel patterns, selecting regions to be offloaded to accelerators in heterogeneous platforms and generating the parallel code.
- Some frameworks use speculation as their main method for software parallelization and distribution. However, this approach implies a significant overhead that outweighs the performance improvements achieved by the optimizations. This is even more critical in the embedded domain, especially for real-time systems that must meet strict deadlines. Therefore, runtime optimization approaches such as speculation are not well suited for this domain.
- Most of the frameworks targeting the embedded domain are evaluated only on synthetic platforms. However, for a solid assessment of the applicability, frameworks should be evaluated on real commercial platforms in which the non-idealities of parallel and heterogeneous systems come into play.

The tool flow proposed in this thesis takes into account the previous observations. It integrates key aspects for an effective software parallelization and distribution into a single unified framework in which *(i)* a platform model allows to tailor the optimizations to the target system; *(ii)* the program analysis is based on both static and dynamic information; *(iii)* performance information is used to assess the potential of optimizations; *(iv)* four different parallel patterns are extracted; *(v)* heterogeneous platforms are supported; and *(vi)* parallel code is generated in widely used programming paradigms. In addition, the tool flow has been evaluated on relevant commercial platforms. This tool flow is discussed in detail through the following chapters.

Table 2.1: Summary of Frameworks for Software Parallelization and Distribution

Framework or Author	Type	Domain	Basis	Input	Platform Model	Program Analysis	Performance Analysis	Main Approach	Patterns	Heterogeneity	Programming Paradigms
Alchemist [323]	Academic	HPC	Valgrind	C/C++	✗	Dynamic	✗	Profiling	DLP^1/TLP	✗	None
Ariadne [193]	Academic	HPC	-	C	✗	None	✗	Annotations	RLP	✗	OpenMP/Pthreads/Cilk/SVP
autoPar [179]	Academic	HPC	ROSE	C/C++	✗	Static	✗	Pattern	DLP	✗	OpenMP
AutopaR [147]	Academic	HPC	GCC	C	✗	Static	✗	Pattern	RLP	✗	OpenMP
Cetus [77]	Academic	HPC	Cetus	C	✗	Static	✓	Pattern	DLP	✗	OpenMP
Compaan [62, 198]	Commercial	Embedded	CoSy	C	✓	Static	✗	Polyhedral	DLP	✓	Pthreads/VHDL
Cordes [66, 63]	Academic	Embedded	MACC	C	✓	Hybrid	✓	ILP/GA	PLP/TLP	✓	MPA
DiscoPop [177, 176]	Academic	HPC	LLVM	C	✗	Hybrid	✓	Pattern	DLP/PLP/TLP	✗	None
Edler Von Koch [84]	Academic	HPC	LLVM	C/C++	✗	Hybrid	✗	Commutativity	DLP/TLP	✗	None
Embla [89, 187]	Academic	HPC	Valgrind	C	✗	Dynamic	✗	Profiling	DLP/TLP	✗	None
Geuns [98]	Academic	Embedded	-	C	✗	Static	✗	Pattern	PLP	✗	None
GRAPHITE [242]	Academic	HPC	GCC	C	✗	Static	✗	Polyhedral	DLP	✗	None
Gupta [121]	Academic	HPC	TPO	C	✗	Hybrid	✗	Speculation	RLP	✗	None
Huckleberry [60]	Academic	HPC	-	C	✓	None	✗	Annotations	RLP	✗	Cell SDK
Intel Advisor [135]	Commercial	HPC	-	C/C++/C#	✗	Dynamic	✓	Annotations	DLP	✗	None
Kehr [151, 152, 236]	Academic	Automotive	OTAWA	C	✓	Static	✓	Pattern	TLP	✗	None
Kismet [141]	Academic	HPC	LLVM	C	✓	Dynamic	✓	Profiling	DLP	✗	None
Kremlin [96]	Academic	HPC	LLVM	C	✗	Dynamic	✓	Profiling	DLP	✗	OpenMP^2
Lengauer [171]	Academic	HPC	-	C	✗	Hybrid	✗	Polyhedral	DLP	✗	None
MAPS (Seq. flow) [51]	Academic	Embedded	LLVM	C	✓	Hybrid	✓	Pattern	DLP/PLP/TLP	✗	CPN,MPI
OSCAR [150, 125]	Commercial	Embedded	-	C/Fortran	✗	Hybrid	✗	Pattern	DLP/TLP	✓	OpenMP/Pthreads
Par4All [237]	Academic	HPC	PIPS	C^3/Fortran	✗	Static	✗	Polyhedral	DLP	✓	OpenMP/CUDA/OpenCL
Paragon [264]	Academic	HPC	Cetus	C	✗	Hybrid	✗	Speculation	DLP	✓	CUDA
Paralax [299]	Academic	HPC	LLVM	C	✗	Static	✓	Annotations	PLP	✗	Pthread
Parallware [19]	Commercial	HPC	LLVM	C	✗	Static	✗	Kernel Extraction	DLP	✓	OpenMP/OpenACC
Pareon [301]	Commercial	Embedded	-	C/C++	✗	Dynamic	✓	Profiling	DLP/PLP	✗	vfTasks
Parwiz [157]	Academic	HPC	Pin	C	✗	Hybrid	✗	Profiling	DLP	✗	None
PLUTO [38, 39]	Academic	HPC	-	C	✗	Static	✗	Polyhedral	DLP	✗	OpenMP
PNgen [306]	Academic	Embedded	SUIF	C	✗	Static	✗	Polyhedral	DLP	✗	PPN XML
Polly [120, 293]	Academic	HPC	LLVM	C/C++	✗	Static	✗	Polyhedral	DLP	✓	OpenMP/OpenCL/CUDA
Prism [75]	Commercial	Embedded	-	C/C++	✗	Hybrid	✓	Profiling	DLP	✗	None
Prospector [163, 164]	Academic	HPC	Pin	C	✗	Dynamic	✗	Profiling	DLP	✗	None
Rauchwerger [253]	Academic	HPC	-	Fortran	✗	Hybrid	✓	Speculation	DLP	✗	None
REAPAR [244]	Academic	HPC	-	C	✗	Dynamic	✓	Pattern	RLP	✗	Threads
Rugina [257]	Academic	HPC	SUIF	C	✗	Static	✗	Pattern	RLP	✗	Cilk
Sambamba [278]	Academic	HPC	LLVM	C/C++	✗	Hybrid	✓	Adaptive	DLP/TLP	✗	Binary
Thies [291]	Academic	HPC	Valgrind	C	✓	Dynamic	✓	Annotation	PLP	✗	None
Tournavitis [296, 295]	Academic	HPC	CoSy	C	✗	Hybrid	✓	Machine Learning	DLP/PLP	✗	OpenMP
VELOCITY [43]	Academic	HPC	IMPACT	C	✗	Hybrid	✓	Pattern	PLP	✗	None
Wang [310]	Academic	HPC	LLVM	C	✗	Hybrid	✗	Speculation	DLP	✓	OpenCL
This Thesis	Academic	Embedded	LLVM	C/C++	✓	Hybrid	✓	Pattern	DLP/PLP/TLP/RLP	✓	OpenMP/OpenCL/CUDA/CPN

[1] Parallel patterns have to be manually extracted using the data dependence provided by the framework
[2] Manual code generation assisted by suggestions of the framework
[3] The framework imposes restrictions to the code that it can handle (e.g., it must be in the form of SCoPs/SANLPs)

Bibliography

[1] "Standard for Information Technology–Portable Operating System Interface (POSIX(R)) Base Specifications, Issue 7," *IEEE Std 1003.1, 2013 Edition (incorporates IEEE Std 1003.1-2008, and IEEE Std 1003.1-2008/Cor 1-2013)*, pp. 1–3906, April 2013.

[2] U. A. Acar, A. Charguéraud, and M. Rainey, "Oracle scheduling: Controlling granularity in implicitly parallel languages," *SIGPLAN Not.*, vol. 46, no. 10, pp. 499–518, Oct. 2011. [Online]. Available: http://doi.acm.org/10.1145/2076021.2048106

[3] Agam Shah, "Nvidia powers up drones, robots with Jetson TX1 board," [Online] Available https://www.computerworld.com/article/3003989/robotics/nvidia-powers-up-drones-robots-with-jetson-tx1-board.html (accessed 01/2018).

[4] M. Aguilar, R. Jimenez, R. Leupers, and G. Ascheid, "Improving performance and productivity for software development on TI Multicore DSP platforms," in *2014 6th European Embedded Design in Education and Research Conference (EDERC)*, Sept 2014, pp. 31–35.

[5] M. A. Aguilar, J. F. Eusse, P. Ray, R. Leupers, G. Ascheid, W. Sheng, and P. Sharma, "Parallelism extraction in embedded software for Android devices," in *2015 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, July 2015, pp. 9–17.

[6] M. A. Aguilar and R. Leupers, "Unified identification of multiple forms of parallelism in embedded applications," in *2015 International Conference on Parallel Architecture and Compilation (PACT)*, Oct 2015, pp. 482–483.

[7] M. A. Aguilar, R. Leupers, G. Ascheid, and J. F. Eusse, "Extraction of recursion level parallelism for embedded multicore systems," in *2017 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, July 2017.

[8] M. A. Aguilar, A. Aggarwal, A. Shaheen, R. Leupers, G. Ascheid, J. Castrillon, and L. Fitzpatrick, "Multi-grained Performance Estimation for MPSoC Compilers: Work-in-progress," in *Proceedings of the 2017 International Conference on Compilers, Architectures and Synthesis for Embedded Systems Companion*, ser. CASES '17. New York, NY, USA: ACM, 2017, pp. 14:1–14:2.

[9] M. A. Aguilar, J. F. Eusse, R. Leupers, G. Ascheid, and M. Odendahl, "Extraction of Kahn Process Networks from While Loops in Embedded Software," in *Proceedings of the 2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conf on Embedded Software and Systems*, ser. HPCC-CSS-ICESS '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 1078–1085.

[10] M. A. Aguilar, J. F. Eusse, P. Ray, R. Leupers, G. Ascheid, W. Sheng, and P. Sharma, "Towards Parallelism Extraction for Heterogeneous Multicore Android Devices," *International Journal of Parallel Programming*, Dec 2016. [Online]. Available: https://doi.org/10.1007/s10766-016-0479-5

[11] M. A. Aguilar and R. Leupers, "Towards Effective Parallelization and Accelerator Offloading for Heterogeneous Multicore Embedded Systems," in *Proceedings of the 2016 International Summer School on Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems (ACACES)*, July 2016, pp. 121–124.

[12] M. A. Aguilar, R. Leupers, G. Ascheid, and N. Kavvadias, "A Toolflow for Parallelization of Embedded Software in Multicore DSP Platforms," in *Proceedings of the 18th International Workshop on Software and Compilers for Embedded Systems*, ser. SCOPES '15. New York, NY, USA: ACM, 2015, pp. 76–79.

[13] M. A. Aguilar, R. Leupers, G. Ascheid, N. Kavvadias, and L. Fitzpatrick, "Schedule-aware Loop Parallelization for Embedded MPSoCs by Exploiting Parallel Slack," in *Proceedings of the Conference on Design, Automation & Test in Europe*, ser. DATE '17. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2017, pp. 1237–1240.

[14] M. A. Aguilar, R. Leupers, G. Ascheid, and L. G. Murillo, "Automatic Parallelization and Accelerator Offloading for Embedded Applications on Heterogeneous MPSoCs," in *Proceedings of the 53rd Annual Design Automation Conference*, ser. DAC '16. New York, NY, USA: ACM, 2016, pp. 49:1–49:6.

[15] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2006.

[16] S. M. Alnaeli, A. Alali, and J. I. Maletic, "Empirically examining the parallelizability of open source software system," in *Proceedings of the 2012 19th Working Conference on Reverse Engineering*, ser. WCRE '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 377–386.

[17] B. Alpern, M. N. Wegman, and F. K. Zadeck, "Detecting equality of variables in programs," in *Proceedings of the 15th ACM SIGPLAN-SIGACT Symposium on*

Principles of Programming Languages, ser. POPL '88. New York, NY, USA: ACM, 1988, pp. 1–11. [Online]. Available: http://doi.acm.org/10.1145/73560.73561

[18] S. Amarasinghe, M. Gordon, R. Soulé, and E. Wong, "StreamIt Benchmarks," [Online] Available http://groups.csail.mit.edu/cag/streamit/shtml/benchmarks.shtml/ (accessed 02/2018).

[19] Appentra, "Parallware: Novel LLVM-Based Software Technology to Assist in Parallelization of Scientific Codes with OpenMP and OpenACC," [Online] Available https://www.appentra.com/technology/ (accessed 08/2017).

[20] M. Arenaz, J. Tourino, and R. Doallo, "Compiler support for parallel code generation through kernel recognition," in *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.*, April 2004, pp. 79–.

[21] ARM, "ARM big.LITTLE," [Online] Available https://developer.arm.com/technologies/big-little (accessed 08/2017).

[22] Associated Computer Experts, "CoSy compiler development system," [Online] Available http://www.ace.nl/compiler/cosy.html (accessed 08/2017).

[23] ASUS, "Nexus 7 (2013)," [Online] Available http://www.asus.com/Tablets_Mobile/Nexus_7_2013/ (accessed 01/2018).

[24] D. I. August, D. A. Connors, S. A. Mahlke, J. W. Sias, K. M. Crozier, B.-C. Cheng, P. R. Eaton, Q. B. Olaniran, and W. M. W. Hwu, "Integrated predicated and speculative execution in the impact epic architecture," in *Proceedings. 25th Annual International Symposium on Computer Architecture (Cat. No.98CB36235)*, Jun 1998, pp. 227–237.

[25] AUTOSAR Consortium, "AUTOSAR Standard," [Online] Available https://www.autosar.org/ (accessed 08/2017).

[26] E. Ayguadé, B. Blainey, A. Duran, J. Labarta, F. Martínez, X. Martorell, and R. Silvera, "Is the schedule clause really necessary in OpenMP?" in *WOMPAT'03*, Berlin, Heidelberg.

[27] I. Bacivarov, W. Haid, K. Huang, and L. Thiele, *Methods and Tools for Mapping Process Networks onto Multi-Processor Systems-On-Chip*. Boston, MA: Springer US, 2010, pp. 1007–1040. [Online]. Available: https://doi.org/10.1007/978-1-4419-6345-1_35

[28] R. Baert, E. Brockmeyer, S. Wuytack, and T. J. Ashby, "Exploring Parallelizations of Applications for MPSoC Platforms Using MPA," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '09. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2009, pp. 1148–1153. [Online]. Available: http://dl.acm.org/citation.cfm?id=1874620.1874898

[29] C. Ballabriga, H. Cassé, C. Rochange, and P. Sainrat, "OTAWA: an open toolbox for adaptive WCET analysis," in *Software Technologies for Embedded and Ubiquitous Systems - 8th IFIP WG 10.2 International Workshop, SEUS 2010, Waidhofen/Ybbs, Austria, October 13-15, 2010. Proceedings*, 2010, pp. 35–46. [Online]. Available: https://doi.org/10.1007/978-3-642-16256-5_6

[30] Barry Pangrle, "Favorite Forecast Fallacies," [Online] Available https://semiengineering.com/favorite-forecast-fallacies/ (accessed 08/2017).

[31] M.-W. Benabderrahmane, L.-N. Pouchet, A. Cohen, and C. Bastoul, "The polyhedral model is more widely applicable than you think," in *Proceedings of the 19th Joint European Conference on Theory and Practice of Software, International Conference on Compiler Construction*, ser. CC'10/ETAPS'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 283–303.

[32] P. Blinzer, "The heterogeneous system architecture: It's beyond the GPU," in *2014 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV)*, July 2014, pp. iii–iii.

[33] B. Blume, R. Eigenmann, K. Faigin, J. Grout, J. Hoeflinger, D. Padua, P. Petersen, B. Pottenger, L. Rauchwerger, P. Tu, and S. Weatherford, "Polaris: The next generation in parallelizing compilers," in *Proceedings of the Workshop on Languages and Compilers for Parallel Computing*. Springer-Verlag, Berlin/Heidelberg, 1994, pp. 10–1.

[34] W. Blume, R. Eigenmann, K. Faigin, J. Grout, J. Hoeflinger, D. Padua, P. Petersen, W. Pottenger, L. Rauchwerger, P. Tu, and S. Weatherford, "Effective automatic parallelization with polaris," *International Journal of Parallel Programming*, 1995.

[35] R. D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, and Y. Zhou, "Cilk: An efficient multithreaded runtime system," *SIGPLAN Not.*, vol. 30, no. 8, pp. 207–216, Aug. 1995. [Online]. Available: http://doi.acm.org/10.1145/209937.209958

[36] O. A. R. Board, "OpenMP," [Online] Available http://www.openmp.org (accessed 08/2017).

[37] B. Boissinot, "Towards an SSA based compiler back-end: Some interesting properties of SSA and its extensions," Ph.D. dissertation, 2010.

[38] U. Bondhugula, "PLUTO - An automatic parallelizer and locality optimizer for affine loop nests," [Online] Available http://pluto-compiler.sourceforge.net/ (accessed 08/2017).

[39] U. Bondhugula, A. Hartono, J. Ramanujam, and P. Sadayappan, "A practical automatic polyhedral parallelizer and locality optimizer," *SIGPLAN Not.*,

vol. 43, no. 6, pp. 101–113, Jun. 2008. [Online]. Available: http://doi.acm.org/10.1145/1379022.1375595

[40] M. Boyer, "CUDA Kernel Overhead," [Online] Available https://www.cs.virginia.edu/~mwb7w/cuda_support/kernel_overhead.html (accessed 12/2017).

[41] M. Boyer, "Memory Transfer Overhead," [Online] Available https://www.cs.virginia.edu/~mwb7w/cuda_support/memory_transfer_overhead.html (accessed 12/2017).

[42] R. P. Brent, "The parallel evaluation of general arithmetic expressions," *J. ACM*, vol. 21, no. 2, pp. 201–206, Apr. 1974. [Online]. Available: http://doi.acm.org/10.1145/321812.321815

[43] M. J. Bridges, "The velocity compiler: Extracting efficient multicore execution from legacy sequential codes," Ph.D. dissertation, Princeton, NJ, USA, 2008, aAI3332403.

[44] J. M. Cardoso, J. G. F. Coutinho, and P. C. Diniz, ""Targeting heterogeneous computing platforms"," in *Embedded Computing for High Performance*, J. M. Cardoso, J. G. F. Coutinho, and P. C. Diniz, Eds. Boston: Morgan Kaufmann, 2017, pp. 227 – 254.

[45] S. Carew, "Texas Instruments eyes shift away from wireless," [Online] Available https://www.reuters.com/article/texasinstruments-wireless-idUSL1E8KP5FN20120925?irpc=932 (accessed 08/2017).

[46] S. Cass, "Multicore Processors Create Software Headaches," [Online] Available https://www.technologyreview.com/s/418580/multicore-processors-create-software-headaches/ (accessed 08/2017).

[47] S. Cass, "The top programming languages 2017 (embedded)," [Online] Available http://spectrum.ieee.org/computing/software/the-2017-top-programming-languages (Accessed 7/2017).

[48] J. Castrillon, R. Leupers, and G. Ascheid, "MAPS: Mapping Concurrent Dataflow Applications to Heterogeneous MPSoCs," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 1, pp. 527–545, Feb 2013.

[49] J. Castrillon, W. Sheng, R. Jessenberger, L. Thiele, L. Schorr, B. Juurlink, M. Alvarez-Mesa, A. Pohl, V. Reyes, and R. Leupers, "Multi/many-core programming: Where are we standing?" in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2015, pp. 1708–1717.

[50] J. Castrillon, A. Tretter, R. Leupers, and G. Ascheid, "Communication-aware mapping of KPN applications onto heterogeneous MPSoCs," in *DAC Design Automation Conference 2012*, June 2012, pp. 1262–1267.

[51] J. Castrillon, "Programming Heterogeneous MPSoCs: Tool Flows to Close the Software Productivity Gap," Ph.D. dissertation, RWTH Aachen Univeristy, Chair for Software for Systems on Silicon, April 2013.

[52] J. Castrillon and R. Leupers, *Programming Heterogeneous MPSoCs: Tool Flows to Close the Software Productivity Gap.* Springer, 2014.

[53] J. Ceng, J. Castrillon, W. Sheng, H. Scharwächter, R. Leupers, G. Ascheid, H. Meyr, T. Isshiki, and H. Kunieda, "MAPS: An Integrated Framework for MPSoC Application Parallelization," in *Proceedings of the 45th Annual Design Automation Conference*, ser. DAC '08. New York, NY, USA: ACM, 2008, pp. 754–759. [Online]. Available: http://doi.acm.org/10.1145/1391469.1391663

[54] J. Ceng, "A Methodology for Efficient Multiprocessor System-on-Chip Software Development," Ph.D. dissertation, RWTH Aachen Univeristy, Chair for Software for Systems on Silicon, April 2011.

[55] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S. H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *2009 IEEE International Symposium on Workload Characterization (IISWC)*, Oct 2009, pp. 44–54.

[56] S. Che, J. W. Sheaffer, M. Boyer, L. G. Szafaryn, L. Wang, and K. Skadron, "A characterization of the Rodinia benchmark suite with comparison to contemporary CMP workloads," in *Workload Characterization (IISWC), 2010 IEEE International Symposium on*, Dec 2010, pp. 1–11.

[57] S. J. Cho, S. H. Yun, and J. W. Jeon, "A roofline model based on working set size for embedded systems," *IEICE Electronics Express*, vol. 11, no. 15, pp. 20 140 560–20 140 560, 2014.

[58] CNN Tech, "How android beat the iphone to world domination," [Online] Available http://money.cnn.com/2017/06/28/technology/business/android-iphone-world-domination/index.html (accessed 01/2018).

[59] M. Cole, *Algorithmic Skeletons: Structured Management of Parallel Computation.* Cambridge, MA, USA: MIT Press, 1991.

[60] R. L. Collins, B. Vellore, and L. P. Carloni, "Recursion-driven parallel code generation for multi-core platforms," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '10. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2010, pp. 190–195. [Online]. Available: http://dl.acm.org/citation.cfm?id=1870926.1870972

[61] P. Community, "Polly - GPGPU Code Generation," [Online] Available http://polly.llvm.org/documentation/gpgpucodegen.html (accessed 03/2018).

[62] Compaan Design BV, "Compaan Design: C-to-Dataflow Compiler," [Online] Available http://www.compaandesign.com/ (accessed 08/2017).

[63] D. Cordes, M. Engel, P. Marwedel, and O. Neugebauer, "Automatic extraction of multi-objective aware pipeline parallelism using genetic algorithms," in *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, ser. CODES+ISSS '12. New York, NY, USA: ACM, 2012, pp. 73–82. [Online]. Available: http://doi.acm.org/10.1145/2380445.2380463

[64] D. Cordes, M. Engel, P. Marwedel, and O. Neugebauer, "Automatic extraction of multi-objective aware pipeline parallelism using genetic algorithms," in *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, ser. CODES+ISSS '12. New York, NY, USA: ACM, 2012, pp. 73–82.

[65] D. Cordes, M. Engel, O. Neugebauer, and P. Marwedel, "Automatic extraction of pipeline parallelism for embedded heterogeneous multi-core platforms," in *Proceedings of the 2013 International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, ser. CASES '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 4:1–4:10. [Online]. Available: http://dl.acm.org/citation.cfm?id=2555729.2555733

[66] D. Cordes, A. Heinig, P. Marwedel, and A. Mallik, "Automatic extraction of pipeline parallelism for embedded software using linear programming," in *Proceedings of the 2011 IEEE 17th International Conference on Parallel and Distributed Systems*, ser. ICPADS '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 699–706.

[67] D. Cordes and P. Marwedel, "Multi-objective aware extraction of task-level parallelism using genetic algorithms," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '12. San Jose, CA, USA: EDA Consortium, 2012, pp. 394–399. [Online]. Available: http://dl.acm.org/citation.cfm?id=2492708.2492808

[68] D. Cordes, P. Marwedel, and A. Mallik, "Automatic parallelization of embedded software using hierarchical task graphs and integer linear programming," in *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, ser. CODES/ISSS '10. New York, NY, USA: ACM, 2010, pp. 267–276. [Online]. Available: http://doi.acm.org/10.1145/1878961.1879009

[69] D. Cordes, O. Neugebauer, M. Engel, and P. Marwedel, "Automatic Extraction of Task-Level Parallelism for Heterogeneous MPSoCs," in *Proceedings of the 2013 42Nd International Conference on Parallel Processing*, ser. ICPP '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 950–959.

[70] D. A. Cordes, "Automatic parallelization for embedded multi-core systems using high-level cost models," Ph.D. dissertation, TU Dortmund University, 2013.

[71] S. P. E. Corporation, "LTE Uplink Receiver PHY Benchmark," [Online] Available https://www.spec.org/cpu2006/ (accessed 01/2018).

[72] cplusplus.com, "Standard Containers," [Online] Available http://www.cplusplus.com/reference/stl/ (accessed 08/2017).

[73] R. Cringley, "Parallel universe," [Online] Available https://www.technologyreview.com/s/411425/parallel-universe/ (accessed 08/2017).

[74] Critical Blue, "Embedded WebKit - Case Study," [Online] Available https://www.prism-services.io/pdf/webkit.pdf (accessed 08/2017).

[75] Critical Blue, "Prism," [Online] Available https://www.prism-services.io/index.html (accessed 08/2017).

[76] M. Danelutto, T. De Matteis, G. Mencagli, and M. Torquati, "A divide-and-conquer parallel pattern implementation for multicores," in *Proceedings of the 3rd International Workshop on Software Engineering for Parallel Systems*, ser. SEPS 2016. New York, NY, USA: ACM, 2016, pp. 10–19.

[77] C. Dave, H. Bae, S. J. Min, S. Lee, R. Eigenmann, and S. Midkiff, "Cetus: A source-to-source compiler infrastructure for multicores," *Computer*, vol. 42, no. 12, pp. 36–42, Dec 2009.

[78] J. Diaz, C. Muñoz-Caro, and A. Niño, "A Survey of Parallel Programming Models and Tools in the Multi and Many-Core Era," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 8, pp. 1369–1386, Aug 2012.

[79] J. Doerfert, K. Streit, S. Hack, and Z. Benaissa, "Polly's polyhedral scheduling in the presence of reductions," in *Proc. 5rd International Workshop on Polyhedral Compilation Techniques (IMPACT)*, Amsterdam, Netherlands, 2015.

[80] Doug Abbott, "Effective use of Pthreads in embedded Linux designs," [Online] Available http://www.embedded.com/design/programming-languages-and-tools/4431425/Effective-use-of-Pthreads-in-embedded-Linux-designs--Part-1---The-multitasking-paradigm (accessed 08/2017).

[81] A. Duran, X. Teruel, R. Ferrer, X. Martorell, and E. Ayguade, "Barcelona OpenMP Tasks Suite: A Set of Benchmarks Targeting the Exploitation of Task Parallelism in OpenMP," in *2009 International Conference on Parallel Processing*, Sept 2009, pp. 124–131.

[82] A. Duran, X. Teruel, R. Ferrer, X. Martorell, and E. Ayguade, "Barcelona OpenMP tasks suite: A set of benchmarks targeting the exploitation of task parallelism in OpenMP," in *Proc. of ICPP*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 124–131.

[83] Eclipse Foundation, "Eclipse," [Online] Available www.eclipse.org (accessed 02/2018).

[84] T. J. K. Edler Von Koch, "Automated detection of structured coarse-grained parallelism in sequential legacy applications," Ph.D. dissertation, University of Edinburgh, November 2014.

[85] J. F. Eusse, C. Williams, L. G. Murillo, R. Leupers, and G. Ascheid, "Pre-architectural performance estimation for asip design based on abstract processor models," in *2014 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV)*, July 2014, pp. 133–140.

[86] J. F. Eusse, C. Williams, and R. Leupers, "Coex: A novel profiling-based algorithm/architecture co-exploration for asip design," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 8, no. 3, pp. 17:1–17:16, May 2015.

[87] FaillissementsDossier, "Bankruptcy terminated Vector Fabrics B.V." [Online]. Available: http://www.faillissementsdossier.nl/en/bankruptcy/1198991/vector-fabrics-b-v.aspx

[88] K. Fatahalian, J. Sugerman, and P. Hanrahan, "Understanding the Efficiency of GPU Algorithms for Matrix-matrix Multiplication," in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, ser. HWWS '04. New York, NY, USA: ACM, 2004, pp. 133–137. [Online]. Available: http://doi.acm.org/10.1145/1058129.1058148

[89] K.-F. Faxén, K. Popov, S. Jansson, and L. Albertsson, "Embla - data dependence profiling for parallel programming," in *Proceedings of the 2008 International Conference on Complex, Intelligent and Software Intensive Systems*, ser. CISIS '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 780–785. [Online]. Available: http://dx.doi.org/10.1109/CISIS.2008.52

[90] M. Feathers, *Working Effectively with Legacy Code*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2004.

[91] J. Ferrante, K. J. Ottenstein, and J. D. Warren, "The program dependence graph and its use in optimization," *ACM Trans. Program. Lang. Syst.*, vol. 9, no. 3, pp. 319–349, Jul. 1987. [Online]. Available: http://doi.acm.org/10.1145/24039.24041

[92] A. Fonseca and B. Cabral, *Evaluation of Runtime Cut-off Approaches for Parallel Programs*. Cham: Springer International Publishing, 2017, pp. 121–134. [Online]. Available: https://doi.org/10.1007/978-3-319-61982-8_13

[93] I. Foster, *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

[94] K. Fürlinger and M. Gerndt, "ompP: A profiling tool for OpenMP," in *IWOMP'05/IWOMP'06*, Berlin, Heidelberg, 2008.

[95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

[96] S. Garcia, D. Jeon, C. M. Louie, and M. B. Taylor, "Kremlin: Rethinking and rebooting gprof for the multicore age," *SIGPLAN Not.*, vol. 46, no. 6, pp. 458–469, Jun. 2011. [Online]. Available: http://doi.acm.org/10.1145/1993316.1993553

[97] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990.

[98] S. Geuns, M. Bekooij, T. Bijlsma, and H. Corporaal, "Parallelization of while loops in nested loop programs for shared-memory multiprocessor systems," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2011.

[99] Gilles Kahn, "The Semantics of a Simple Language for Parallel Programming," in *IFIP Congress 74*, North Holland, Amsterdam, 1974, pp. 471–475.

[100] M. Girkar and C. D. Polychronopoulos, "The hierarchical task graph as a universal intermediate representation," *International Journal of Parallel Programming*, vol. 22, no. 5, pp. 519–551, Oct 1994. [Online]. Available: https://doi.org/10.1007/BF02577777

[101] A. Goens, R. Khasanov, J. Castrillon, M. Hähnel, T. Smejkal, and H. Härtig, "Tetris: A multi-application run-time system for predictable execution of static mappings," in *Proceedings of the 20th International Workshop on Software and Compilers for Embedded Systems*, ser. SCOPES '17. New York, NY, USA: ACM, 2017, pp. 11–20. [Online]. Available: http://doi.acm.org/10.1145/3078659.3078663

[102] T. Gonzalez, O. H. Ibarra, and S. Sahni, "Bounds for lpt schedules on uniform processors," *SIAM*, vol. 6, no. 1, pp. 155–166, 1977.

[103] H. González-Vélez and M. Leyton, "A survey of algorithmic skeleton frameworks: High-level structured parallel programming enablers," *Softw. Pract. Exper.*, vol. 40, no. 12, pp. 1135–1160, Nov. 2010. [Online]. Available: http://dx.doi.org/10.1002/spe.v40:12

[104] R. Gonçalves, M. Amaris, T. Okada, P. Bruel, and A. Goldman, "OpenMP is Not as Easy as It Appears," in *2016 49th Hawaii International Conference on System Sciences (HICSS)*, Jan 2016, pp. 5742–5751.

[105] Google, "Android," [Online] Available https://www.android.com/ (accessed 01/2018).

[106] Google, "Android Build System," [Online] Available https://developer.android.com/studio/build/index.html (accessed 01/2018).

[107] Google, "Android: Canvas and Drawables," [Online] Available http://developer.android.com/guide/topics/graphics/2d-graphics.html (accessed 01/2018).

[108] Google, "Android Studio and SDK Tools," [Online] Available https://developer.android.com/studio/index.html (accessed 01/2018).

[109] Google, "ART and Dalvik," [Online] Available https://source.android.com/devices/tech/dalvik/index.html (accessed 01/2017).

[110] Google, "Create WebP Images," [Online] Available https://developer.android.com/studio/write/convert-webp.html (accessed 01/2018).

[111] Google, "Java Native Interface," [Online] Available http://developer.android.com/training/articles/perf-jni.html (accessed 01/2018).

[112] Google, "KitKat 4.4," [Online] Available https://www.android.com/versions/kit-kat-4-4/ (accessed 01/2018).

[113] Google, "libwebp API Documentation," [Online] Available https://developers.google.com/speed/webp/docs/api (accessed 01/2018).

[114] Google, "Native Development Kit," [Online] Available http://developer.android.com/ndk/guides/concepts.html (accessed 01/2018).

[115] Google, "WebP: A new image format for the Web," [Online] Available https://developers.google.com/speed/webp/ (accessed 01/2018).

[116] M. I. Gordon, W. Thies, and S. Amarasinghe, "Exploiting coarse-grained task, data, and pipeline parallelism in stream programs," in *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XII. New York, NY, USA: ACM, 2006, pp. 151–162. [Online]. Available: http://doi.acm.org/10.1145/1168857.1168877

[117] G. Goth, "Entering a parallel universe," *Communications of the ACM*, vol. 52, no. 9, pp. 15–17, Sep. 2009. [Online]. Available: http://doi.acm.org/10.1145/1562164.1562171

[118] S. L. Graham, P. B. Kessler, and M. K. Mckusick, "Gprof: A call graph execution profiler," *SIGPLAN Not.*, vol. 17, no. 6, pp. 120–126, Jun. 1982. [Online]. Available: http://doi.acm.org/10.1145/872726.806987

[119] S. Grauer-Gray, L. Xu, R. Searles, S. Ayalasomayajula, and J. Cavazos, "Autotuning a high-level language targeted to GPU codes," in *2012 Innovative Parallel Computing (InPar)*, May 2012, pp. 1–10.

[120] T. Grosser, A. Groesslinger, and C. Lenngauer, "Polly - performing polyhedral optimizations on a low-level intermediate representation," *Parallel Processing Letters*, vol. 22, no. 04, p. 1250010, 2012.

[121] M. Gupta, S. Mukhopadhyay, and N. Sinha, "Automatic parallelization of recursive procedures," *Int. J. Parallel Program.*, vol. 28, no. 6, pp. 537–562, Dec. 2000. [Online]. Available: http://dx.doi.org/10.1023/A:1007560600904

[122] H. Halawa, H. A. Abdelhafez, A. Boktor, and M. Ripeanu, *NVIDIA Jetson Platform Characterization*. Cham: Springer International Publishing, 2017, pp. 92–105. [Online]. Available: https://doi.org/10.1007/978-3-319-64203-1_7

[123] C. Hammer and G. Snelting, "Flow-sensitive, context-sensitive, and object-sensitive information flow control based on program dependence graphs," *Int. J. Inf. Secur.*, vol. 8, no. 6, pp. 399–422, Oct. 2009. [Online]. Available: http://dx.doi.org/10.1007/s10207-009-0086-1

[124] M. Harris, "Unified Memory for CUDA Beginners," [Online] Available https://devblogs.nvidia.com/unified-memory-cuda-beginners/ (accessed 02/2018).

[125] A. Hayashi, Y. Wada, T. Watanabe, T. Sekiguchi, M. Mase, J. Shirako, K. Kimura, and H. Kasahara, *Parallelizing Compiler Framework and API for Power Reduction and Software Productivity of Real-Time Heterogeneous Multicores*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 184–198. [Online]. Available: https://doi.org/10.1007/978-3-642-19595-2_13

[126] Y. He, C. E. Leiserson, and W. M. Leiserson, "The cilkview scalability analyzer," in *Proceedings of the Twenty-second Annual ACM Symposium on Parallelism in Algorithms and Architectures*, ser. SPAA '10. New York, NY, USA: ACM, 2010, pp. 145–156. [Online]. Available: http://doi.acm.org/10.1145/1810479.1810509

[127] G. Hegde, Siddhartha, N. Ramasamy, and N. Kapre, "Caffepresso: An optimized library for deep learning on embedded accelerator-based platforms," in *Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, ser. CASES '16. New York, NY, USA: ACM, 2016, pp. 14:1–14:10. [Online]. Available: http://doi.acm.org/10.1145/2968455.2968511

[128] M. D. Hill and M. R. Marty, "Retrospective on Amdahl's Law in the Multicore Era," *Computer*, vol. 50, no. 6, pp. 12–14, 2017.

[129] HSA Foundation, "Heterogeneous System Architecture (HSA)," [Online] Available http://www.hsafoundation.com/ (accessed 03/2018).

[130] J. Huang, A. Raman, T. B. Jablin, Y. Zhang, T.-H. Hung, and D. I. August, "Decoupled software pipelining creates parallelization opportunities," in *Proceedings of the 8th Annual IEEE/ACM International Symposium on Code Generation and Optimization*, ser. CGO '10. New York, NY, USA: ACM, 2010, pp. 121–130. [Online]. Available: http://doi.acm.org/10.1145/1772954.1772973

[131] W.-m. Hwu, K. Keutzer, and T. G. Mattson, "The concurrency challenge," *IEEE Des. Test*, vol. 25, no. 4, pp. 312–320, Jul. 2008. [Online]. Available: http://dx.doi.org/10.1109/MDT.2008.110

[132] IDC, "Smartphone OS Market Share, 2017 Q1," [Online] Available https://www.idc.com/promo/smartphone-market-share/os (accessed 1/2018).

[133] A. Ilic, F. Pratas, and L. Sousa, "Beyond the roofline: Cache-aware power and energy-efficiency modeling for multi-cores," *IEEE Transactions on Computers*, vol. 66, no. 1, pp. 52–58, Jan 2017.

[134] A. Ilic, F. Pratas, and L. Sousa, "Cache-aware roofline model: Upgrading the loft," *IEEE Comput. Archit. Lett.*, vol. 13, no. 1, pp. 21–24, Jan. 2014. [Online]. Available: http://dx.doi.org/10.1109/L-CA.2013.6

[135] Intel, "Intel Advisor," [Online] Available https://software.intel.com/en-us/intel-advisor-xe (accessed 08/2017).

[136] Intel, "Intel Advisor Roofline," [Online] Available https://software.intel.com/en-us/articles/intel-advisor-roofline (accessed 12/2017).

[137] Intel, "Intel Cilk Plus," [Online] Available https://software.intel.com/en-us/intel-cilk-plus-support (accessed 08/2017).

[138] International Technology Roadmap for Semiconductors (ITRS), "System integration roadmaps in itrs 2.0, 2015 edition," [Online] Available http://www.itrs2.net/itrs-reports.html (Accessed 7/2017).

[139] M. Islam, "On the limitations of compilers to exploit thread-level parallelism in embedded applications," in *Computer and Information Science, 2007. ICIS 2007. 6th IEEE/ACIS International Conference on*, July 2007, pp. 60–66.

[140] T. Janjusic and K. Kavi, "Hardware and application profiling tools," in *Advances in Computers*, A. Hurson, Ed. Elsevier, 2014, vol. 92, ch. 3, pp. 105–160.

[141] D. Jeon, S. Garcia, C. Louie, and M. B. Taylor, "Kismet: Parallel speedup estimates for serial programs," *SIGPLAN Not.*, vol. 46, no. 10, pp. 519–536, Oct. 2011. [Online]. Available: http://doi.acm.org/10.1145/2076021.2048108

[142] A. A. Jerraya and W. Wolf, *Multiprocessor Systems-on-chips*, ser. Electronics & Electrical. Morgan Kaufmann, 2005. [Online]. Available: https://books.google.de/books?id=7i9Z69lrYBoC

[143] C. Jesshope, M. Lankamp, and L. Zhang, "The Implementation of an SVP Many-core Processor and the Evaluation of Its Memory Architecture," *SIGARCH Comput. Archit. News*, vol. 37, no. 2, pp. 38–45, Jul. 2009. [Online]. Available: http://doi.acm.org/10.1145/1577129.1577136

[144] R. E. Johnson, “Software development is program transformation,” in *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, ser. FoSER ’10. New York, NY, USA: ACM, 2010, pp. 177–180.

[145] R. C. Johnson, “Efficient program analysis using dependence flow graphs,” Ph.D. dissertation, Cornell University, August 1994.

[146] A. f. C. M. A. Joint Task Force on Computing Curricula and I. C. Society, *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. New York, NY, USA: ACM, 2013, 999133.

[147] M. E. Kalender, C. Mergenci, and O. Ozturk, “Autopar: An automatic parallelization tool for recursive calls,” in *2014 43rd International Conference on Parallel Processing Workshops*, Sept 2014, pp. 159–165.

[148] I. Karkowski and H. Corporaal, “Overcoming the limitations of the traditional loop parallelization,” *Future Gener. Comput. Syst.*, vol. 13, no. 4-5, pp. 407–416, Mar. 1998.

[149] A. Karpov, “32 OpenMP traps for C++ developers,” [Online] Available https://software.intel.com/en-us/articles/32-openmp-traps-for-c-developers (accessed 08/2017).

[150] H. Kasahara, M. Obata, and K. Ishizaka, *Automatic Coarse Grain Task Parallel Processing on SMP Using OpenMP*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 189–207. [Online]. Available: https://doi.org/10.1007/3-540-45574-4_13

[151] S. Kehr, M. Panić, E. Quiñones, B. Böddeker, J. B. Sandoval, J. Abella, F. J. Cazorla, and G. Schäfer, “Supertask: Maximizing runnable-level parallelism in autosar applications,” in *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*, ser. DATE ’16. San Jose, CA, USA: EDA Consortium, 2016, pp. 25–30. [Online]. Available: http://dl.acm.org/citation.cfm?id=2971808.2971815

[152] S. Kehr, E. Quiñones, B. Böddeker, and G. Schäfer, “Parallel execution of autosar legacy applications on multicore ecus with timed implicit communication,” in *Proceedings of the 52Nd Annual Design Automation Conference*, ser. DAC ’15. New York, NY, USA: ACM, 2015, pp. 42:1–42:6. [Online]. Available: http://doi.acm.org/10.1145/2744769.2744889

[153] A. Kejariwal, A. V. Veidenbaum, A. Nicolau, M. Girkarmark, X. Tian, and H. Saito, “Challenges in exploitation of loop parallelism in embedded applications,” in *Proceedings of the 4th International Conference on Hardware/Software Codesign and System Synthesis*, ser. CODES+ISSS ’06. New York, NY, USA: ACM, 2006, pp. 173–180.

[154] J. E. Kelley, Jr and M. R. Walker, “Critical-path planning and scheduling,” in *Papers Presented at the December 1-3, 1959, Eastern Joint IRE-AIEE-ACM Computer*

Conference, ser. IRE-AIEE-ACM '59 (Eastern). New York, NY, USA: ACM, 1959, pp. 160–173. [Online]. Available: http://doi.acm.org/10.1145/1460299.1460318

[155] K. Kennedy and J. R. Allen, *Optimizing Compilers for Modern Architectures: A Dependence-based Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002.

[156] C. W. Kessler, "Pattern-driven automatic parallelization," *Sci. Program.*, vol. 5, no. 3, pp. 251–274, Aug. 1996. [Online]. Available: http://dx.doi.org/10.1155/1996/406379

[157] A. Ketterlin and P. Clauss, "Profiling data-dependence to assist parallelization: Framework, scope, and optimization," in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-45. Washington, DC, USA: IEEE Computer Society, 2012, pp. 437–448. [Online]. Available: http://dx.doi.org/10.1109/MICRO.2012.47

[158] K. Keutzer and T. Mattson, "Our pattern language (OPL). a pattern language for parallel programming," [Online] Available http://parlab.eecs.berkeley.edu/wiki/patterns/patterns (accessed 07/2017).

[159] Khronos, "The OpenCL specification. version 1.1," [Online] Available https://www.khronos.org/registry/cl/specs/opencl-1.1.pdf (accessed 02/2018).

[160] Khronos Group, "OpenCL embedded boards comparison 2015," https://www.khronos.org/news/events/opencl-embedded-boards-comparison-2015. Visited on Mar. 2017.

[161] B. Kienhuis, E. Rijpkema, and E. Deprettere, "Compaan: deriving process networks from Matlab for embedded signal processing architectures," in *Proceedings of the Eighth International Workshop on Hardware/Software Codesign. CODES 2000 (IEEE Cat. No.00TH8518)*, May 2000, pp. 13–17.

[162] M. Kim, "Dynamic program analysis algorithms to assist parallelization," Ph.D. dissertation, Atlanta, GA, USA, 2012.

[163] M. Kim, H. Kim, and C.-K. Luk, "Prospector: Discovering parallelism via dynamic data-dependence profiling," in *Proceedings of the 2nd USENIX Workshop on Hot Topics in Parallelism, HOTPAR*, vol. 10, 2010, pp. 395–416.

[164] M. Kim, N. B. Lakshminarayana, H. Kim, and C.-K. Luk, "Sd3: An efficient dynamic data-dependence profiling mechanism," *IEEE Trans. Comput.*, vol. 62, no. 12, pp. 2516–2530, Dec. 2013. [Online]. Available: http://dx.doi.org/10.1109/TC.2012.182

[165] E. Konstantinidis and Y. Cotronis, "A quantitative roofline model for GPU kernel performance estimation using micro-benchmarks and hardware metric profiling," *Journal of Parallel and Distributed Computing*, vol. 107, no. Supplement C, pp. 37 – 56, 2017.

[166] R. Koo and F. Gandolfi, "Compiler business value," IBM, Tech. Rep. [Online]. Available: ttps://www.ibm.com/developerworks/rational/cafe/docBodyAttachments/2880-102-1-4641/RAW14079-USEN-00.pdf

[167] C. Lee, "UTDSP Benchmark," [Online] Available http://www.eecg.toronto.edu/~corinna/DSP/infrastructure/UTDSP.html (accessed 09/2017), 1998.

[168] E. Lee and D. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *Computers, IEEE Transactions on*, vol. C-36, no. 1, pp. 24–35, Jan 1987.

[169] E. A. Lee and T. M. Parks, "Readings in hardware/software co-design," G. De Micheli, R. Ernst, and W. Wolf, Eds. Norwell, MA, USA: Kluwer Academic Publishers, 2002, ch. Dataflow Process Networks, pp. 59–85. [Online]. Available: http://dl.acm.org/citation.cfm?id=567003.567010

[170] S. Lee and R. Eigenmann, "OpenMPC: Extended OpenMP Programming and Tuning for GPUs," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–11. [Online]. Available: https://doi.org/10.1109/SC.2010.36

[171] C. Lengauer and M. Griebl, "On the parallelization of loop nests containing while loops," in *Parallel Algorithms/Architecture Synthesis, 1995. Proceedings., First Aizu International Symposium on*, Mar 1995, pp. 10–18.

[172] C. Lengauer, "Loop parallelization in the polytope model," in *Proceedings of the 4th International Conference on Concurrency Theory*, ser. CONCUR '93. London, UK, UK: Springer-Verlag, 1993, pp. 398–416.

[173] R. Leupers, M. A. Aguilar, J. Castrillon, and W. Sheng, "Software compilation techniques for heterogeneous embedded multi-core systems," S. S. Bhattacharyya, E. F. Deprettere, R. Leupers, and J. Takala, Eds. Springer, 2018.

[174] R. Leupers, M. A. Aguilar, J. F. Eusse, J. Castrillon, and W. Sheng, *MAPS: A Software Development Environment for Embedded Multicore Applications*. Dordrecht: Springer Netherlands, sep 2017.

[175] F. Li, A. Pop, and A. Cohen, "Advances in Parallel-Stage Decoupled Software Pipelining Leveraging Loop Distribution, Stream-Computing and the SSA Form," in *WIR 2011: Workshop on Intermediate Representations*. Chamonix, France: Florent Bouchez and Sebastian Hack and Eelco Visser, Apr. 2011, pp. pp.29–36, 8 pages Categories and Subject Descriptors D.3.4 [Programming

Languages]: Processors-Compilers, Optimization. [Online]. Available: https://hal-mines-paristech.archives-ouvertes.fr/hal-00744090

[176] Z. Li, "Discovery of potential parallelism in sequential programs," Ph.D. dissertation, October 2016.

[177] Z. Li, R. Atre, Z. U. Huda, A. Jannesari, and F. Wolf, "Unveiling parallelization opportunities in sequential programs," *Journal of Systems and Software*, vol. 117, p. 282–295, Jul. 2016.

[178] C. Liao *et al.*, "Early experiences with the OpenMP accelerator model," in *OpenMP in the Era of Low Power Devices and Accelerators*. Springer, 2013, pp. 84–98. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-40698-0_7

[179] C. Liao, D. J. Quinlan, J. J. Willcock, and T. Panas, "Semantic-aware automatic parallelization of modern applications using high-level abstractions," *International Journal of Parallel Programming*, vol. 38, no. 5, pp. 361–378, Oct 2010. [Online]. Available: https://doi.org/10.1007/s10766-010-0139-0

[180] T. Ligocki, "Empirical Roofline Tool," [Online] Available https://bitbucket.org/berkeleylab/cs-roofline-toolkit (accessed 12/2017).

[181] X. Liu, R. Zhao, L. Han, and P. Liu, "An Automatic Parallel-Stage Decoupled Software Pipelining Parallelization Algorithm Based on OpenMP," in *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, July 2013, pp. 1825–1831.

[182] LLVM Foundation, "clang: a C language family frontend for LLVM," [Online] Available https://clang.llvm.org/ (accessed 02/2018).

[183] LLVM Foundation, "LLVM Bitcode File Format," [Online] Available https://llvm.org/docs/BitCodeFormat.html (accessed 9/2017).

[184] Y. J. Lo, S. Williams, B. Van Straalen, T. J. Ligocki, M. J. Cordery, N. J. Wright, M. W. Hall, and L. Oliker, *Roofline Model Toolkit: A Practical Tool for Architectural and Program Analysis*. Cham: Springer International Publishing, 2015, pp. 129–148.

[185] U. Lopez-Novoa, A. Mendiburu, and J. Miguel-Alonso, "A survey of performance modeling and simulation techniques for accelerator-based computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 1, pp. 272–281, Jan 2015.

[186] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: Building customized program analysis tools with dynamic instrumentation," *SIGPLAN Not.*, vol. 40, no. 6, pp. 190–200, Jun. 2005. [Online]. Available: http://doi.acm.org/10.1145/1064978.1065034

[187] J. Mak, K.-F. Faxén, S. Janson, and A. Mycroft, "Estimating and exploiting potential parallelism by source-level dependence profiling," in *Proceedings of the 16th International Euro-Par Conference on Parallel Processing: Part I*, ser. EuroPar'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 26–37. [Online]. Available: http://dl.acm.org/citation.cfm?id=1887695.1887700

[188] Manuel Arenaz, "A Success Case using Parallware: The NAS Parallel Benchmark EP," [Online] Available openmpcon.org/wp-content/uploads/openmpcon2015-manuel-arenaz-appentra.pdf (accessed 08/2017).

[189] A. Marongiu, A. Capotondi, G. Tagliavini, and L. Benini, "Simplifying many-core-based heterogeneous soc programming with offload directives," *IEEE Transactions on Industrial Informatics*, vol. 11, no. 4, pp. 957–967, Aug 2015.

[190] A. Marongiu *et al.*, "Simplifying many-core-based heterogeneous SoC programming with offload directives," *IEEE Trans. on Ind. Informatics*, vol. 11, no. 4, pp. 957–967, Aug 2015.

[191] A. Marowka, "Think parallel: Teaching parallel programming today," *IEEE Distributed Systems Online*, vol. 9, no. 8, pp. 1–1, Aug 2008.

[192] M. Mase, Y. Onozaki, K. Kimura, and H. Kasahara, "Parallelizable C and its performance on low power high performance multicore processors," in *Proc. of 15th Workshop on Compilers for Parallel Computing*, vol. 2011, 2010.

[193] A. Mastoras and G. Manis, "Ariadne - directive-based parallelism extraction from recursive functions," *J. Parallel Distrib. Comput.*, vol. 86, no. C, pp. 16–28, Dec. 2015. [Online]. Available: http://dx.doi.org/10.1016/j.jpdc.2015.07.009

[194] Matthew Humphries, "Nintendo Switch Uses a Standard Tegra X1 Processor," [Online] Available http://uk.pcmag.com/news/88426/nintendo-switch-uses-a-standard-tegra-x1-processor (accessed 01/2018).

[195] T. Mattson, B. Sanders, and B. Massingill, *Patterns for Parallel Programming*, 1st ed. Addison-Wesley Professional, 2004.

[196] M. McCool, J. Reinders, and A. Robison, *Structured Parallel Programming: Patterns for Efficient Computation*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann, 2012.

[197] A. Meade, J. Buckley, and J. J. Collins, "Challenges of evolving sequential to parallel code: An exploratory review," in *Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th Annual ERCIM Workshop on Software Evolution*, ser. IWPSE-EVOL '11. New York, NY, USA: ACM, 2011, pp. 1–5. [Online]. Available: http://doi.acm.org/10.1145/2024445.2024447

[198] S. Meijer, S. van Haastregt, D. Nadezhkin, and B. Kienhuis, "Kahn Process Network IR Modeling for Multicore Compilation," Leiden Institute

of Advanced Computer Science, Niels Bohrweg 1, 2333 CA, Leiden, Netherlands, Tech. Rep. MSU-CSE-06-2. [Online]. Available: liacs.leidenuniv.nl/~kienhuisacj/ftp/technicalreport2.pdf

[199] S. P. Midkiff, *Automatic Parallelization: An Overview of Fundamental Compiler Techniques*, ser. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2012. [Online]. Available: https://doi.org/10.2200/S00340ED1V01Y201201CAC019

[200] R. Miller, "Sequential programming considered harmful?" [Online] Available http://spectrum.ieee.org/at-work/education/sequential-programming-considered-harmful (Accessed 8/2017).

[201] G. Mitra *et al.*, "Implementation and Optimization of the OpenMP Accelerator Model for the TI Keystone II Architecture," in *Using and Improving OpenMP for Devices and More*. Springer, 2014, pp. 202–214. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-11454-5_15

[202] K. Morris, "Compaan Design releases HotSpot Parallelizer for ISO C," [Online] Available https://www.eejournal.com/article/20100518-02/ (accessed 08/2017).

[203] S. S. Muchnick, *Advanced Compiler Design and Implementation*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997.

[204] Multicore Association, "Software-hardware interface for multi-many-core (SHIM) specification," [Online] Available http://www.multicore-association.org (accessed 07/2017).

[205] R. Muresano, D. Rexachs, and E. Luque, "Learning parallel programming: a challenge for university students," *Procedia Computer Science*, vol. 1, no. 1, pp. 875 – 883, 2010, iCCS 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050910000979

[206] L. G. Murillo, S. Wawroschek, J. Castrillon, R. Leupers, and G. Ascheid, "Automatic detection of concurrency bugs through event ordering constraints," in *Proceedings of the Conference on Design, Automation & Test in Europe*, ser. DATE '14. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2014, pp. 282:1–282:6.

[207] A. K. Nanda, J. R. Moulic, R. E. Hanson, G. Goldrian, M. N. Day, B. D. D'Arnora, and S. Kesavarapu, "Cell/b.e. blades: Building blocks for scalable, real-time, interactive, and digital media servers," *IBM J. Res. Dev.*, vol. 51, no. 5, pp. 573–582, Sep. 2007. [Online]. Available: http://dx.doi.org/10.1147/rd.515.0573

[208] N. Nethercote and J. Seward, "Valgrind: A framework for heavyweight dynamic binary instrumentation," *SIGPLAN Not.*, vol. 42, no. 6, pp. 89–100, Jun. 2007. [Online]. Available: http://doi.acm.org/10.1145/1273442.1250746

[209] A. Nicolaou, "Eight strategies for tackling legacy code you didn't write," [Online] Available https://www.fastcompany.com/3029446/eight-strategies-for-tackling-legacy-code-you-didnt-write (Accessed 7/2017).

[210] V. P. Nikolskiy, V. V. Stegailov, and V. S. Vecher, "Efficiency of the Tegra K1 and X1 systems-on-chip for classical molecular dynamics," in *2016 International Conference on High Performance Computing Simulation (HPCS)*, July 2016, pp. 682–689.

[211] NVIDIA, "CUDA C Best Practices Guide," [Online] Available http://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html#axzz4pAtErnjo (accessed 08/2017).

[212] NVIDIA, "CUDA C Programming Guide," [Online] Available http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html (accessed 02/2018).

[213] NVIDIA, "CUDA Ecosystem," [Online] Available https://developer.nvidia.com/tools-ecosystem (accessed 08/2017).

[214] NVIDIA, "CUDA Zone," [Online] Available https://developer.nvidia.com/cuda-zone (accessed 08/2017).

[215] NVIDIA, "Deep Learning," [Online] Available https://www.nvidia.com/en-us/deep-learning-ai/developer/ (accessed 08/2017).

[216] NVIDIA, "Drive PX: Scalable Supercomputer for Autonomous Driving," [Online] Available http://www.nvidia.com/object/drive-px.html (accessed 08/2017).

[217] NVIDIA, "Introducing Xavier, the NVIDIA AI Supercomputer for the Future of Autonomous Transportation," [Online] Available https://blogs.nvidia.com/blog/2016/09/28/xavier/ (accessed 08/2017).

[218] NVIDIA, "Maxwell architecture," [Online] Available https://developer.nvidia.com/maxwell-compute-architecture (Accessed 12/2017).

[219] NVIDIA, "Nvidia jetson," [Online] Available http://www.nvidia.com/object/embedded-systems-dev-kits-modules.html (Accessed 7/2017).

[220] NVIDIA, "Profiler User's Guide," [Online] Available http://docs.nvidia.com/cuda/profiler-users-guide/index.html (accessed 01/2018).

[221] NVIDIA, "Shield: The Ultimate Tablet For Gamers," [Online] Available https://www.nvidia.com/en-us/shield/tablet/ (accessed 01/2018).

[222] NVIDIA, “Tegra,” [Online] Available http://www.nvidia.com/object/tegra.html (accessed 08/2017).

[223] Oak Ridge National Laboratory, “History of CUDA, OpenCL, and the GPGPU,” [Online] Available https://www.olcf.ornl.gov/kb_articles/history-of-the-gpgpu/ (accessed 08/2017).

[224] G. Ofenbeck, R. Steinmann, V. C. Cabezas, D. G. Spampinato, and M. Püschel, “Applying the roofline model,” in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2014, pp. 76 – 85.

[225] OpenACC-Standard.org, “OpenACC,” [Online] Available https://www.openacc.org/ (accessed 08/2017).

[226] OpenMP Architecture Review Board, “OpenMP Application Programming Interface. Version 4.0,” [Online] Available www.openmp.org/wp-content/uploads/OpenMP4.0.0.pdf (accessed 08/2017), July 2013.

[227] OpenMP Review Board, “OpenMP Application Program Interface. Version 3.0,” [Online] Available http://www.openmp.org/wp-content/uploads/spec30.pdf (accessed 2/2018).

[228] OpenMP Review Board, “OpenMP Application Program Interface. Version 3.1,” [Online] Available www.openmp.org/wp-content/uploads/OpenMP3.1.pdf (accessed 1/2018).

[229] OpenMP Review Board, “OpenMP Application Program Interface. Version 4.5,” [Online] Available http://www.openmp.org/wp-content/uploads/openmp-4.5.pdf (accessed 09/2017).

[230] OpenMP Review Board, “OpenMP C and C++ Application Program Interface Version 1.0,” [Online] Available http://www.openmp.org/wp-content/uploads/cspec10.pdf (accessed 2/2018).

[231] Oscar Technology Corporation, “Oscar.” [Online]. Available: http://www.oscartech.jp/en/

[232] G. Ottoni, R. Rangan, A. Stoler, and D. I. August, “Automatic thread extraction with decoupled software pipelining,” in *Proceedings of the 38th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 38. Washington, DC, USA: IEEE Computer Society, 2005, pp. 105–118. [Online]. Available: http://dx.doi.org/10.1109/MICRO.2005.13

[233] G. Ozen, E. Ayguadé, and J. Labarta, *On the Roles of the Programmer, the Compiler and the Runtime System When Programming Accelerators in OpenMP*. Cham: Springer International Publishing, 2014, pp. 215–229.

[234] G. Ozen *et al.*, "On the roles of the programmer, the compiler and the runtime system when programming accelerators in OpenMP," in *Using and Improving OpenMP for Devices, Tasks and More*. Springer, 2014, pp. 215–229. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-11454-5_16

[235] D. Padua, Ed., *Whole Program Analysis*. Boston, MA: Springer US, 2011, pp. 2159–2159. [Online]. Available: https://doi.org/10.1007/978-0-387-09766-4_2164

[236] M. Panić, S. Kehr, E. Quiñones, B. Boddecker, J. Abella, and F. J. Cazorla, "Runpar: An allocation algorithm for automotive applications exploiting runnable parallelism in multicores," in *Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis*, ser. CODES '14. New York, NY, USA: ACM, 2014, pp. 29:1–29:10. [Online]. Available: http://doi.acm.org/10.1145/2656075.2656096

[237] Par4All Members, "Par4All," [Online] Available http://par4all.github.io/ (accessed 08/2017).

[238] M. Pelcat, K. Desnos, J. Heulot, C. Guy, J. F. Nezan, and S. Aridhi, "Preesm: A dataflow-based rapid prototyping framework for simplifying multicore dsp programming," in *2014 6th European Embedded Design in Education and Research Conference (EDERC)*, Sept 2014, pp. 36–40.

[239] P. M. Petersen, "Evaluation of programs and parallelizing compilers using dynamic analysis techniques," Ph.D. dissertation, Champaign, IL, USA, 1993, uMI Order No. GAX93-14926.

[240] A. D. Pimentel, C. Erbas, and S. Polstra, "A systematic approach to exploring embedded system architectures at multiple abstraction levels," *IEEE Trans. Comput.*, vol. 55, no. 2, pp. 99–112, Feb. 2006. [Online]. Available: http://dx.doi.org/10.1109/TC.2006.16

[241] K. Pingali, M. Beck, R. Johnson, M. Moudgill, and P. Stodghill, "Dependence flow graphs: An algebraic approach to program dependencies," in *Proceedings of the 18th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL '91. New York, NY, USA: ACM, 1991, pp. 67–78. [Online]. Available: http://doi.acm.org/10.1145/99583.99595

[242] S. Pop, A. Cohen, C. Bastoul, S. Girbal, G.-A. Silber, and N. Vasilache, "Graphite: Polyhedral analyses and optimizations for gcc," in *Proceedings of the 2006 GCC Developers Summit*, 2006, p. 2006.

[243] L.-N. Pouchet, "PolyBench/C: The Polyhedral Benchmark Suite," [Online] Available http://web.cs.ucla.edu/~pouchet/software/polybench/ (accessed 01/2018).

[244] L. Prechelt and S. U. Hánssgen, "Efficient parallel execution of irregular recursive programs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 2, pp. 167–178, Feb. 2002. [Online]. Available: http://dx.doi.org/10.1109/71.983944

[245] R. Pyka, F. Klein, P. Marwedel, and S. Mamagkakis, "Versatile System-level Memory-aware Platform Description Approach for Embedded MPSoCs," *SIGPLAN Not.*, vol. 45, no. 4, pp. 9–16, Apr. 2010. [Online]. Available: http://doi.acm.org/10.1145/1755951.1755891

[246] Qualcomm, "Qualcomm snapdragon mobile platform opencl general programming and optimization," [Online] Available https://developer.qualcomm.com/qfile/33472/80-nb295-11_a.pdf (accessed 1/2018).

[247] Qualcomm, "Snapdragon 845 Mobile Platform," [Online] Available https://www.qualcomm.com/products/snapdragon-845-mobile-platform (accessed 03/2018).

[248] Qualcomm, "Snapdragon Processors," [Online] Available https://www.qualcomm.com/products/snapdragon (accessed 08/2017).

[249] Qualcomm, "Snapdragon S4," [Online] Available https://www.qualcomm.com/products/snapdragon/processors/s4-s1 (accessed 01/2018).

[250] D. Quinlan, "Rose: Compiler support for object-oriented frameworks," *Parallel Processing Letters*, vol. 10, no. 02n03, pp. 215–226, 2000.

[251] E. Raman, G. Ottoni, A. Raman, M. J. Bridges, and D. I. August, "Parallel-stage decoupled software pipelining," in *Proceedings of the 6th Annual IEEE/ACM International Symposium on Code Generation and Optimization*, ser. CGO '08. New York, NY, USA: ACM, 2008, pp. 114–123. [Online]. Available: http://doi.acm.org/10.1145/1356058.1356074

[252] R. Rangan, N. Vachharajani, M. Vachharajani, and D. I. August, "Decoupled software pipelining with the synchronization array," in *Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 177–188. [Online]. Available: https://doi.org/10.1109/PACT.2004.14

[253] L. Rauchwerger and D. Padua, "Parallelizing while loops for multiprocessor systems," in *9th International Parallel Processing Symposium*, Apr 1995, pp. 347–356.

[254] L. Rauchwerger and D. Padua, "The lrpd test: Speculative run-time parallelization of loops with privatization and reduction parallelization," *SIGPLAN Not.*, vol. 30, no. 6, pp. 218–232, Jun. 1995. [Online]. Available: http://doi.acm.org/10.1145/223428.207148

[255] Rik Myslewski, "The 'third era' of app development will be fast, simple, and compact," [Online] Available http://www.theregister.co.uk/Print/2013/08/25/heterogeneous_system_architecture_deep_dive/ (accessed 08/2017).

[256] G. Roelofs, "Portable Network Graphics," [Online] Available http://www.libpng.org/pub/png/png.html (accessed 01/2018).

[257] R. Rugina and M. Rinard, "Automatic parallelization of divide and conquer algorithms," *SIGPLAN Not.*, vol. 34, no. 8, pp. 72–83, May 1999. [Online]. Available: http://doi.acm.org/10.1145/329366.301111

[258] K. Rupp, "40 years of microprocessor trend data," [Online] Available https://www.karlrupp.net/2015/06/40-years-of-microprocessor-trend-data/ (accessed 08/2017).

[259] S. Rus, L. Rauchwerger, and J. Hoeflinger, "Hybrid analysis: Static & dynamic memory reference analysis," *International Journal of Parallel Programming*, vol. 31, no. 4, pp. 251–283, Aug 2003. [Online]. Available: https://doi.org/10.1023/A:1024597010150

[260] Ryan Smith, "ARM Unveils Next Generation Bifrost GPU Architecture & Mali-G71: The New High-End Mali," [Online] Available http://www.anandtech.com/show/10375/arm-unveils-bifrost-and-mali-g71/4 (accessed 08/2017).

[261] S. Sah and V. G. Vaidya, "A Review of Parallelization Tools and Introduction to Easypar," *International Journal of Computer Applications*, vol. 56, no. 12, pp. 30–34, October 2012.

[262] T. Saidani, J. Falcou, C. Tadonki, L. Lacassagne, and D. Etiemble, "Algorithmic skeletons within an embedded domain specific language for the cell processor," in *2009 18th International Conference on Parallel Architectures and Compilation Techniques*, Sept 2009, pp. 67–76.

[263] N. Sakharnykh, "Beyond gpu memory limits with unified memory on pascal," [Online] Available https://devblogs.nvidia.com/beyond-gpu-memory-limits-unified-memory-pascal/ (accessed 02/2018).

[264] M. Samadi, A. Hormati, J. Lee, and S. Mahlke, "Paragon: Collaborative Speculative Loop Execution on GPU and CPU," in *Proceedings of the 5th Annual Workshop on General Purpose Processing with Graphics Processing Units*, ser. GPGPU-5. New York, NY, USA: ACM, 2012, pp. 64–73. [Online]. Available: http://doi.acm.org/10.1145/2159430.2159438

[265] Samsung, "Mobile Processor: Exynos 9 Series (8895)," [Online] Available http://www.samsung.com/semiconductor/minisite/Exynos/Solution/MobileProcessor/Exynos_9_Series_8895.html (accessed 08/2017).

[266] A. Sangiovanni-Vincentelli and G. Martin, "Platform-based design and software design methodology for embedded systems," *IEEE Design Test of Computers*, vol. 18, no. 6, pp. 23–33, Nov 2001.

[267] R. R. Schaller, "Moore's law: Past, present, and future," *IEEE Spectr.*, vol. 34, no. 6, pp. 52–59, Jun. 1997. [Online]. Available: http://dx.doi.org/10.1109/6.591665

[268] G. E. Schalnat, A. Dilger, J. Bowler, G. Randers-Pehrson, and et al., "libpng," [Online] Available http://libpng.org/pub/png/libpng.html (accessed 01/2018).

[269] W. Sheng, S. Schürmans, M. Odendahl, M. Bertsch, V. Volevach, R. Leupers, and G. Ascheid, "A compiler infrastructure for embedded heterogeneous MPSoCs," *Parallel Computing*, vol. 40, no. 2, pp. 51 – 68, 2014, special issue on programming models and applications for multicores and manycores. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167819113001452

[270] Silexica GmbH, "SLX Tool Suite," [Online] Available http://www.silexica.com (accessed 4/2018).

[271] R. Smith, "Samsung Announces Exynos 8895 SoC," [Online] Available https://www.anandtech.com/show/11149/samsung-announces-exynos-8895-soc-10nm (accessed 03/2018).

[272] M. Sottile, T. G. Mattson, and C. E. Rasmussen, *Introduction to Concurrency in Programming Languages*, 1st ed. Chapman & Hall/CRC, 2009.

[273] M. Spierings and R. van de Voort, "Embedded platform selection based on the Roofline model," Master's thesis, Eindhoven University of Technology, 2011.

[274] T. Stefanov, A. Pimentel, and H. Nikolov, *Daedalus: System-Level Design Methodology for Streaming Multiprocessor Embedded Systems on Chips*. Dordrecht: Springer Netherlands, 2017, pp. 1–36. [Online]. Available: https://doi.org/10.1007/978-94-017-7358-4_30-1

[275] T. Stefanov, C. Zissulescu, A. Turjan, B. Kienhuis, and E. Deprettere, "System Design Using Kahn Process Networks: The Compaan/Laura Approach," in *Proceedings of the Conference on Design, Automation and Test in Europe - Volume 1*, ser. DATE '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 10 340–. [Online]. Available: http://dl.acm.org/citation.cfm?id=968878.968962

[276] J. E. Stone, D. Gohara, and G. Shi, "OpenCL: A parallel programming standard for heterogeneous computing systems," *IEEE Des. Test*, vol. 12, no. 3, pp. 66–73, May 2010.

[277] E. Stotzer, "Towards using OpenMP in embedded systems," OpenMPCon: Developers Conference, September 2015.

[278] K. Streit, J. Doerfert, C. Hammacher, A. Zeller, and S. Hack, "Generalized task parallelism," *ACM Trans. Archit. Code Optim.*, vol. 12, no. 1, pp. 8:1–8:25, Apr. 2015. [Online]. Available: http://doi.acm.org/10.1145/2723164

[279] M. Süßand C. Leopold, "Common Mistakes in OpenMP and How to Avoid Them: A Collection of Best Practices," in *Proceedings of the 2005 and 2006 International Conference on OpenMP Shared Memory Parallel Programming*, ser. IWOMP'05/IWOMP'06. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 312–323. [Online]. Available: http://dl.acm.org/citation.cfm?id=1892830.1892863

[280] H. Sutter, "The free lunch is over: A fundamental turn toward concurrency in software," *Dr. Dobb's Journal*, vol. 30, no. 3, pp. 202–210, 2005.

[281] Texas Instruments, "Keystone architecture hyperlink. user's guide," [Online] Available www.ti.com/lit/sprugw8 (accessed 01/2018).

[282] Texas Instruments, "Keystone architecture multicore navigator. user's guide," [Online] Available www.ti.com/lit/ug/sprugr9h/sprugr9h.pdf (accessed 01/2018).

[283] Texas Instruments, "Keystone Multicore Devices," [Online] Available http://processors.wiki.ti.com/index.php/Multicore (accessed 08/2017).

[284] Texas Instruments, "KeyStone SOM SBC," [Online] Available http://processors.wiki.ti.com/index.php/KeyStone_SOM_SBC (accessed 08/2017).

[285] Texas Instruments, "Multicore DSP+ARM KeyStone II System-on-Chip (SoC)," [Online] Available http://www.ti.com/product/66AK2H12 (accessed 08/2017).

[286] Texas Instruments, "Software development kit for multicore DSP Keystone platform," http://www.ti.com/tool/bioslinuxmcsdk. Visited on Mar. 2017.

[287] Texas Instruments, "TDA2SX," [Online] Available http://www.ti.com/product/tda2sx (accessed 08/2017).

[288] Texas Instruments, "TDAx ADAS SoCs," [Online] Available https://www.ti.com/processors/automotive-processors/tdax-adas-socs/overview.html (accessed 08/2017).

[289] Texas Instruments, "TMS320C6678: Multicore Fixed and Floating-Point Digital Signal Processor," [Online] Available http://www.ti.com/product/tms320c6678 (accessed 08/2017).

[290] L. Thiele, I. Bacivarov, W. Haid, and K. Huang, "Mapping applications to tiled multiprocessor embedded systems," in *Seventh International Conference on Application of Concurrency to System Design (ACSD 2007)*, July 2007, pp. 29–40.

[291] W. Thies, V. Chandrasekhar, and S. Amarasinghe, "A Practical Approach to Exploiting Coarse-Grained Pipeline Parallelism in C Programs," in *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 40. Washington, DC, USA: IEEE Computer Society, 2007, pp. 356–369. [Online]. Available: http://dx.doi.org/10.1109/MICRO.2007.7

[292] W. Thies, M. Karczmarek, and S. P. Amarasinghe, "Streamit: A language for streaming applications," in *Proceedings of the 11th International Conference on Compiler Construction*, ser. CC '02. London, UK, UK: Springer-Verlag, 2002, pp. 179–196. [Online]. Available: http://dl.acm.org/citation.cfm?id=647478.727935

[293] Tobias Grosser, "Polly: LLVM Framework for High-Level Loop and Data-Locality Optimizations," [Online] Available https://polly.llvm.org/ (accessed 08/2017).

[294] G. Tournavitis, "Profile-driven parallelization of sequential programs," Ph.D. dissertation, University of Edinburgh, 2011.

[295] G. Tournavitis and B. Franke, "Semi-automatic extraction and exploitation of hierarchical pipeline parallelism using profiling information," in *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '10. New York, NY, USA: ACM, 2010, pp. 377–388. [Online]. Available: http://doi.acm.org/10.1145/1854273.1854321

[296] G. Tournavitis, Z. Wang, B. Franke, and M. F. O'Boyle, "Towards a holistic approach to auto-parallelization: Integrating profile-driven parallelism detection and machine-learning based mapping," *SIGPLAN Not.*, vol. 44, no. 6, pp. 177–187, Jun. 2009. [Online]. Available: http://doi.acm.org/10.1145/1543135.1542496

[297] P. Tröger, "The multi-core era - trends and challenges," *CoRR*, vol. abs/0810.5439, 2008.

[298] N. Vachharajani, R. Rangan, E. Raman, M. J. Bridges, G. Ottoni, and D. I. August, "Speculative decoupled software pipelining," in *Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques*, ser. PACT '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 49–59. [Online]. Available: https://doi.org/10.1109/PACT.2007.66

[299] H. Vandierendonck, S. Rul, and K. D. Bosschere, "The paralax infrastructure: Automatic parallelization with a helping hand," in *2010 19th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, Sept 2010, pp. 389–399.

[300] R. Vargas, E. Quinones, and A. Marongiu, "OpenMP and timing predictability: A possible union?" in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, ser. DATE '15. San Jose, CA, USA: EDA Consortium, 2015, pp. 617–620.

[301] Vector Fabrics, "New Pareon tool from Vector Fabrics smooths multicore software optimization." [Online]. Available: http://www.eejournal.com/industry_news/20120601-05/

[302] Vector Fabrics, "vfTasks parallelization library ." [Online]. Available: https://sourceforge.net/projects/vftasks/

[303] V. Venkatachalam and M. Franz, "Power reduction techniques for microprocessor systems," *ACM Comput. Surv.*, vol. 37, no. 3, pp. 195–237, Sep. 2005. [Online]. Available: http://doi.acm.org/10.1145/1108956.1108957

[304] S. Verdoolaege, *isl: An Integer Set Library for the Polyhedral Model*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 299–302. [Online]. Available: https://doi.org/10.1007/978-3-642-15582-6_49

[305] S. Verdoolaege, *Polyhedral Process Networks*. Boston, MA: Springer US, 2010, pp. 931–965. [Online]. Available: https://doi.org/10.1007/978-1-4419-6345-1_33

[306] S. Verdoolaege, H. Nikolov, and T. Stefanov, "Pn: A tool for improved derivation of process networks," *EURASIP J. Embedded Syst.*, vol. 2007, no. 1, pp. 19–19, Jan. 2007. [Online]. Available: http://dx.doi.org/10.1155/2007/75947

[307] N. Vlatko, "The dangers of spaghetti code," [Online] Available https://jaxenter.com/the-dangers-of-spaghetti-code-117807.html (accessed 08/2017).

[308] R. Vuduc, A. Chandramowlishwaran, J. Choi, M. Guney, and A. Shringarpure, "On the Limits of GPU Acceleration," in *Proceedings of the 2Nd USENIX Conference on Hot Topics in Parallelism*, ser. HotPar'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 13–13.

[309] Z. Wang, D. Grewe, and M. F. P. O'boyle, "Automatic and Portable Mapping of Data Parallel Programs to OpenCL for GPU-Based Heterogeneous Systems," *ACM Trans. Archit. Code Optim.*, vol. 11, no. 4, pp. 42:1–42:26, Dec. 2014. [Online]. Available: http://doi.acm.org/10.1145/2677036

[310] Z. Wang, D. Powell, B. Franke, and M. O'Boyle, *Exploitation of GPUs for the Parallelisation of Probably Parallel Legacy Code*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 154–173. [Online]. Available: https://doi.org/10.1007/978-3-642-54807-9_9

[311] Wayne Wolf, "Memory-oriented optimization techniques for dealing with performance bottlenecks: Part 1," [Online] Available https://www.embedded.com/print/4413343 (accessed 03/2018).

[312] Wikipedia, "Exynos," [Online] Available https://en.wikipedia.org/wiki/Exynos (Accessed 7/2017).

[313] Wikipedia, "List of qualcomm snapdragon systems-on-chip," [Online] Available https://en.wikipedia.org/wiki/List_of_Qualcomm_Snapdragon_systems-on-chip (Accessed 7/2017).

[314] Wikipedia, "Omap," [Online] Available https://en.wikipedia.org/wiki/OMAP (Accessed 7/2017).

[315] Wikipedia, "Tegra," [Online] Available https://en.wikipedia.org/wiki/Tegra (Accessed 7/2017).

[316] D. Wilde, V. Loechner, and T. Risset, "Polylib - A library of polyhedral functions," [Online] Available http://www.irisa.fr/polylib/ (accessed 08/2017).

[317] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, no. 4, pp. 65–76, Apr. 2009. [Online]. Available: http://doi.acm.org/10.1145/1498765.1498785

[318] R. P. Wilson, R. S. French, C. S. Wilson, S. P. Amarasinghe, J. M. Anderson, S. W. K. Tjiang, S.-W. Liao, C.-W. Tseng, M. W. Hall, M. S. Lam, and J. L. Hennessy, "Suif: An infrastructure for research on parallelizing and optimizing compilers," *SIGPLAN Not.*, vol. 29, no. 12, pp. 31–37, Dec. 1994.

[319] M. Wolfe and U. Banerjee, "Data dependence and its application to parallel processing," *International Journal of Parallel Programming*, vol. 16, no. 2, pp. 137–178, Apr 1987. [Online]. Available: https://doi.org/10.1007/BF01379099

[320] M. Wrinn, "Top 10 Challenges in Parallel Computing," [Online] Available http://www.drdobbs.com/top-10-challenges-in-parallel-computing/212700369 (accessed 08/2017).

[321] Xilinx, "ZYNQ MPSoC Family," [Online] Available https://www.xilinx.com/products/silicon-devices/soc.html (accessed 08/2017).

[322] M. Zahran, "Heterogeneous computing: Here to stay. hardware and software perspectives," *Queue*, vol. 14, no. 6, pp. 40:31–40:42, Dec. 2016. [Online]. Available: http://doi.acm.org/10.1145/3028687.3038873

[323] X. Zhang, A. Navabi, and S. Jagannathan, "Alchemist: A transparent dependence distance profiling infrastructure," in *Proceedings of the 7th Annual IEEE/ACM International Symposium on Code Generation and Optimization*, ser. CGO '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 47–58.

[324] S. Zhao, Y. Bian, and S. Zhang, "A review on refactoring sequential program to parallel code in multicore era," in *Proceedings of 2015 International Conference on Intelligent Computing and Internet of Things*, Jan 2015, pp. 151–154.