ADAPTIVE AND ENERGY-EFFICIENT MANAGEMENT FOR HETEROGENEOUS MULTI-CORE ARCHITECTURES

Dissertation

zur Erlangung des akademischen Grades
Doktor-Ingenieur (Dr.-Ing.)

vorgelegt an der
Technischen Universität Dresden
Fakultät Informatik

eingereicht von

Robert Khasanov

geboren am 16.01.1990 in Tschaikowski, UdSSR

Gutachter:

Prof. Dr.-Ing. Jeronimo Castrillon
Technische Universität Dresden

Prof. Dr.-Ing. Jürgen Teich
Friedrich-Alexander-Universität Erlangen-Nürnberg

Tag der Verteidigung:

25.03.2025

TECHNISCHE
UNIVERSITÄT
DRESDEN

ADAPTIVE AND ENERGY-EFFICIENT MANAGEMENT FOR HETEROGENEOUS MULTI-CORE ARCHITECTURES

ROBERT KHASANOV

June, 2025

Robert Khasanov: *Adaptive and Energy-Efficient Management for Heterogeneous Multi-Core Architectures,* Dissertation © 2025

ABSTRACT

The evolution of processor architectures has seen a significant shift over the past few decades, driven by increasing application demands. Historically, general-purpose computing systems focused on increasing clock speeds and later transitioned to multi-core architectures. At the same time, embedded systems focused on energy-efficient designs and shifted to heterogeneous architectures to handle increasingly complex applications. Over time, the boundaries between embedded and general-purpose systems began to blur: embedded systems integrated multitasking operating systems and started handling more dynamic workloads, while general-purpose systems have adopted energy-efficient principles. This convergence led to the emergence of *Heterogeneous Multi-core Architectures* (HMAs), which combines cores with different performance-energy characteristics but a shared Instruction Set Architecture (ISA), enhancing energy efficiency while allowing workloads to migrate between core types. Initially introduced in embedded systems, HMAs have since expanded to powerful desktop and server platforms, further blurring the boundaries between the two domains.

Dynamic and unpredictable workloads, now prevalent in both domains, demand flexibility and *adaptivity* in *resource management* and *application execution*. HMAs add complexity to these challenges: resource managers must account for heterogeneous cores when allocating resources, while applications must adapt not only to dynamically changing allocations but also to the heterogeneity of the assigned cores.

This thesis proposes a series of solutions to address these adaptivity challenges. At the resource management level, it builds upon Hybrid Application Mapping (HAM) methodologies, introducing novel algorithms for generating spatio-temporal mappings and leveraging domain-specific knowledge to enhance adaptivity in real-time systems. At the application level, this thesis introduces an extension to Kahn Process Networks (KPNs), improving adaptivity in dynamic and heterogeneous environments. Finally, it asserts that fully utilizing HMAs requires coordinated adaptivity between these two levels. This coordination is demonstrated with HARP, a novel resource management framework, which can also efficiently manage unforeseen applications, thereby extending its applicability to desktop and server systems. By addressing both embedded systems and general-purpose platforms featuring HMAs, this thesis optimizes the utilization of these architectures and improves energy efficiency across domains.

PUBLICATIONS

Several ideas, figures and arguments that are presented in this dissertation have been published in prior works. The following lists all the publications cited in this thesis that I co-authored:

[1] Till Smejkal, Robert Khasanov, Jeronimo Castrillon, and Hermann Härtig. *E-Mapper: Energy-Efficient Resource Allocation for Traditional Operating Systems on Heterogeneous Processors*. June 2024. arXiv: 2406.18980 [cs.OS].

[2] Robert Khasanov, Marc Dietrich, and Jeronimo Castrillon. "Flexible Spatio-Temporal Energy-Efficient Runtime Management." In: *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*. Jan. 2024, pp. 777–784. doi: 10.1109/ASP-DAC58780.2024.10473885.

[3] Robert Khasanov, Julian Robledo, Christian Menard, Andrés Goens, and Jeronimo Castrillon. "Domain-specific Hybrid Mapping for Energy-efficient Baseband Processing in Wireless Networks." In: *ACM Trans. Embed. Comput. Syst.* 20.5s (Oct. 2021). issn: 1539-9087. doi: 10.1145/3476991.

[4] Christian Menard, Andrés Goens, Gerald Hempel, Robert Khasanov, Julian Robledo, Felix Teweleitt, and Jeronimo Castrillon. "Mocasin——Rapid Prototyping of Rapid Prototyping Tools: A Framework for Exploring New Approaches in Mapping Software to Heterogeneous Multi-cores." In: *Proceedings of the 2021 Drone Systems Engineering and Rapid Simulation and Performance Evaluation: Methods and Tools Proceedings*. DroneSE and RAPIDO '21. Budapest, Hungary: Association for Computing Machinery, Jan. 2021, pp. 66–73. isbn: 9781450389525. doi: 10.1145/3444950.3447285.

[5] Robert Khasanov and Jeronimo Castrillon. "Energy-efficient Runtime Resource Management for Adaptable Multi-application Mapping." In: *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Mar. 2020, pp. 909–914. doi: 10.23919/DATE48585.2020.9116381.

[6] Robert Khasanov, Andrés Goens, and Jeronimo Castrillon. "Implicit Data-Parallelism in Kahn Process Networks: Bridging the MacQueen Gap." In: *Proceedings of the 9th Workshop and 7th Workshop on Parallel Programming and RunTime Management Techniques for Manycore Architectures and Design Tools and Architectures for Multicore Embedded Computing Platforms*. PARMA-DITAM '18. Manchester, United Kingdom: Association for Com-

puting Machinery, Jan. 2018, pp. 20–25. ISBN: 978-1-4503-6444-7. DOI: `10.1145/3183767.3183790`.

[7] Andrés Goens, Robert Khasanov, Jeronimo Castrillon, Marcus Hähnel, Till Smejkal, and Hermann 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*. SCOPES '17. Sankt Goar, Germany: Association for Computing Machinery, 2017, pp. 11–20. ISBN: 9781450350396. DOI: `10.1145/3078659.3078663`.

The following publications are co-authored by me but not cited in this thesis:

[1] Shaokai Lin, Tassilo Tanneberger, Jiahong Bi, Guangyu Feng, Yimo Xu, Julian Robledo, Robert Khasanov, and Jeronimo Castrillon. "Navigating Time and Energy Trade-Offs in Reactive Heterogeneous Systems." In: *IEEE Embedded Systems Letters* (2024), pp. 1–1. DOI: `10.1109/LES.2024.3469278`.

[2] Markus Hahnel, Frehiwot Melak Arega, Waltenegus Dargie, Robert Khasanov, and Jeronimo Castrillon. "Application interference analysis: Towards energy-efficient workload management on heterogeneous micro-server architectures." In: *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. 2017, pp. 432–437. DOI: `10.1109/INFCOMW.2017.8116415`.

[3] Andres Goens, Robert Khasanov, Jeronimo Castrillon, Simon Polstra, and Andy Pimentel. "Why Comparing System-Level MPSoC Mapping Approaches is Difficult: A Case Study." In: *2016 IEEE 10th International Symposium on Embedded Multicore-/Many-core Systems-on-Chip (MCSoC)*. 2016, pp. 281–288. DOI: `10.1109/MCSoC.2016.48`.

It takes a village to raise a child.

— African proverb

ACKNOWLEDGMENTS

First and foremost, I want to express my sincere gratitude to my supervisor Prof. Jeronimo Castrillon for his continuous support, insightful guidance, and encouragement throughout my Ph.D. journey. His mentorship helped me grow as a confident and independent researcher, and also taught me to focus on important problems that matter from a research and practical perspective. I am also grateful for his timely feedback, even on weekends, and for building a friendly and supportive atmosphere within the group.

Speaking of friendly atmosphere in the group, I owe a huge thank you to my current and former colleagues: Justus Adam, Jiahong Bi, Hasna Bouraoui, Alexander Brauckmann, Sebastian Ertel, Mees Frensel, Andrés Goens, Fazal Hameed, Gerald Hempel, Hamid Farzaneh, Clément Fournier, Karl Friebel, Sven Karol, Asif Khan, Nesrine Khouzami, Steffen Köhler, Guilherme Korol, Galina Kozyreva, João Paulo Cardoso de Lima, Christian Menard, Norman Rink, Julian Robledo, Lars Schütze, Anderson Faustino da Silva, and Felix Suchert. You all made this time unforgettable, from the insightful discussions to the fun team activities. And thank you for (patiently!) putting up with my reminders to give a tech talk. A special thanks to Conny Okuma, whose responsiveness and support with administrative matters were invaluable, as well as for urging us to participate in running trainings. I also enjoyed sharing skiing and snowboarding trips with you and other colleagues.

I also want to thank everyone who worked with me as a student. This includes Marc Dietrich, Hannes Frank, Karl Friebel, Dylan Gageot, Fabius Mayer-Uhma, and Tianrui Zheng. Working with you allowed me to explore side projects, many of which influenced my research vision. It was also rewarding to witness your efforts and growth as you overcame challenges in your work. I am happy to see some of you already becoming successful engineers and researchers, and in one case, a colleague.

During my work, I have closely collaborated with the Chair of Operating Systems. First and foremost, I am honored to have Prof. Hermann Härtig as my second advisor, and I am grateful to him for his valuable feedback on my status talk and the papers we worked on together with members of his chair. This collaboration included Marcus Hähnel and Till Smejkal, with whom I worked on the TETRiS project. Over time, this project evolved into HARP, presented in this thesis, on which I collaborated intensively with Till Smejkal. Till,

it was an amazing collaboration. Looking back at how this project evolved over time, especially in the last two years, it feels like we wrote multiple different papers.

I also want to thank the rest of my collaborators that I have not mentioned personally so far. While not all of these projects became part of this thesis, they influenced my broader research vision. Thanks to Markus Hähnel and Frehiwot Melak Arega for the insightful work on the application interference analysis. I thank Arka Maity, Nishant Budhev and Tulika Mitra for sharing their LTE traces with our chair. Thanks to Tung Doan for collaborating on the adaptive self-driving car control project, which resulted in a well-received demonstration. I thank also Shaokai Lin for inviting me to contribute to the work of integrating the Lingua Franca and Mocasin projects.

During my time at the chair, I had an opportunity to work with several tools and frameworks that were generously provided to our group. This includes the Silexica Tool Suite, which was valuable at the beginning of my research. My earliest prototype of Adaptive Process Networks was implemented within this framework.

The work conducted in this thesis would not have been possible without funding. My Ph.D. studies were initially funded by the German Research Foundation (DFG) in the Collaborative Research Center 912 “Highly Adaptive Energy-Efficient Computing” (HAEC), and later by Federal Ministry of Education and Research of Germany (BMBF) within the programme of “Souverän. Digital. Vernetzt.” (6G-life). Both projects also provided an excellent environment for collaboration, which was invaluable throughout my Ph.D. studies.

Finally, and most importantly, I want to thank my friends and family, who constantly reminded me about other aspects of life beyond work. Here in Dresden, I have found friends who have accompanied me through these nine years — thank you for making this one of the best times of my life. To my friends from Russia, whom I met during my time at MIPT and even earlier during my school years, I am grateful that we maintain our friendships. Even though we are spread around the world, I am glad that we occasionally have the chance to visit each other.

I would not have been able to write this thesis without the tremendous support of my family. My parents, grandparents, little brother, aunt, and cousins — it is difficult to express how thankful I am for your constant encouragement and for always being there for me, even from afar. To my mother, I am especially grateful for nurturing my curiosity and giving me the freedom to pursue my passions.

Robert Khasanov, December 2024

CONTENTS

1 Introduction 1
  1.1 Heterogeneous Multi-Core Architectures 2
  1.2 Need for Adaptivity 4
    1.2.1 Adaptivity at the Resource Management Level . 5
    1.2.2 Adaptivity at the Application Level 6
  1.3 Contributions of This Thesis 7
  1.4 Synopsis and Outline 10
2 Foundations of Application Mapping onto HMAs 11
  2.1 Preliminaries 11
    2.1.1 Mapping, Scheduling, and Spatio-Temporal Mapping 11
    2.1.2 Performance and Energy Estimation 14
    2.1.3 Static and Dynamic Power Consumption 15
    2.1.4 Notation 16
  2.2 System Model 16
    2.2.1 Architecture 17
    2.2.2 Application 18
    2.2.3 Mapping 19
  2.3 Mapping of Dataflow Applications 19
    2.3.1 Dataflow Models of Computation 20
    2.3.2 Design-time and Runtime Mapping Approaches 22
    2.3.3 Trace-Based Simulation 23
  2.4 Hybrid Application Mapping 25
    2.4.1 Pareto-Optimal Operating Points 27
    2.4.2 Spatial Mapping Optimization 30
    2.4.3 Spatio-Temporal Mapping Optimization 32
    2.4.4 Addressing Adaptivity Challenges 34
  2.5 Mocasin Framework 35
    2.5.1 Overview of Mocasin 35
    2.5.2 Contributions to Mocasin 37
  2.6 Synopsis 37
3 Related Work 39
  3.1 Design-Time Application Mapping 39
  3.2 Runtime Application Mapping 40
    3.2.1 Runtime Mapping of Embedded Software . . . 40
    3.2.2 Runtime Mapping within OS Schedulers 42
  3.3 Hybrid Application Mapping 43
  3.4 Application Adaptivity 45
4 Efficient Spatio-Temporal Mapping Generation 47
  4.1 Motivational Example 48
  4.2 Spatio-Temporal Mapping Strategies 50
  4.3 Fixed-Point Spatio-Temporal Mapping 52

4.3.1 MMKP-based Algorithm 52
4.3.2 Evaluation 56
4.4 Flexible Spatio-Temporal Mapping 60
4.4.1 STEM: Spatio-Temporal Evolutionary Mapping 60
4.4.2 FFEMS: Fast Flexible Energy-Minimizing Scheduler 64
4.4.3 Evaluation 67
4.5 Synopsis 74
5 Domain-Specific Hybrid Mapping for Baseband Processing 75
5.1 Approaches to Baseband Processing 76
5.2 Baseband Processing Architecture and Parameterization 78
5.3 Task Graph with Phase-Sequential Structure 79
5.4 Efficient Mapping Algorithm for Phased Task Graphs . 80
5.5 Spatio-Temporal Mapping Reusing Previous Solutions 84
5.6 Evaluation 86
5.6.1 Platform Setup 86
5.6.2 Workload Model 87
5.6.3 Generation and Estimation of Operating Points 88
5.6.4 Energy-Efficient Runtime Mapping 91
5.7 Synopsis 95
6 Extending Kahn Process Networks with Adaptivity 97
6.1 Limitations of KPNs: A Motivational Example 98
6.2 Adaptive Process Network 100
6.2.1 Parallel Regions 100
6.2.2 Parallel Channels and Workload Distribution . 102
6.2.3 Malleability 104
6.3 Dynamic Process Manager (DPM) Library 105
6.3.1 Programmer Interface 106
6.3.2 Runtime Topology 108
6.3.3 Configuration Management 108
6.4 Evaluation 110
6.4.1 Experimental Setup 110
6.4.2 Performance Scalability with Parallelization . . 111
6.4.3 Runtime Adaptivity 112
6.5 Synopsis 113
7 Coordinating Adaptivity in General-Purpose Environments 115
7.1 Need for Two-Way Communication 116
7.2 Adapting HAM Methodologies 118
7.3 HARP Design 119
7.3.1 Application Support via `libharp` 120
7.3.2 Resource Allocation 123
7.4 Runtime Exploration of Operating Points 125
7.4.1 Runtime Performance and Power Monitoring . 126
7.4.2 Selection of the Regression Model 126
7.4.3 Runtime Exploration Algorithm 128
7.5 Evaluation 130

7.5.1 Experimental Setup 131
7.5.2 Intel Raptor Lake Evaluation 133
7.5.3 Odroid-XU3-E Evaluation 135
7.5.4 Evaluation of the Learning Process 137
7.5.5 Performance Overhead of HARP 140
7.6 Synopsis 141
8 Conclusions and Outlook 143

Glossary 149
List of Figures 155
List of Tables 157
List of Algorithms 159
List of Listings 161
Bibliography 163

1

INTRODUCTION

The evolution of processor architectures has seen a significant shift over the past few decades, driven by the increasing demands of applications. In general-purpose computing, early performance improvements were primarily driven by *Moore's Law* [134], which predicted that the number of transistors on a chip would doubles every 18–24 months. As transistor sizes shrank, clock frequencies also doubled over the same period, since the processing speed is inversely related to the distance between transistors. This trend was further aided by *Dennard Scaling* [55], which maintained constant power density [59].

However, starting at 65 nm process technology and below, leakage power began to rise exponentially, breaking Dennard Scaling and preventing further increases in clock frequency. To continue performance scaling, computer systems transitioned toward *multi-core architectures*, where multiple processing elements are integrated on a single chip [67]. In these systems, individual core frequencies and complexity were kept constant or even reduced to accommodate for the breakdown of Dennard scaling, and performance improvement were instead achieved through *Thread-Level Parallelism* (TLP) [131].

After certain point, core scaling hit the power budget: additional cores could be added, but they often had to be switched off or significantly underclocked to satisfy the *Thermal Design Power* (TDP) constraints. This phenomenon has been termed as *Dark Silicon* [59]. In response, processor designs began transitioning to *heterogeneous architectures*, where only the cores best suited to the application are activated, enabling faster and energy-efficient computing [131].

The shift toward heterogeneous architectures was seen much earlier in the embedded systems domain. In embedded systems, the focus was traditionally on energy efficiency, real-time constraints, and cost-effectiveness rather than solely increasing performance [125]. Historically, embedded software consisted of simple routines within a single control loop, that were often programmed in assembly. However, modern embedded devices run much more complex applications such as baseband and multimedia processing, including video conferencing and voice recognition [19, 95, 213]. This shift necessitated the adoption of multi-core architectures to handle increased computational demands while maintaining low power consumption and meeting real-time performance requirements [18].

To achieve energy efficiency and meet application needs, embedded systems have traditionally employed heterogeneous architectures that integrate different types of processing units, such as Central Processing

Units (CPUs), Digital Signal Processors (DSPs), Application-Specific Instruction-Set Processors (ASIPs), and hardware accelerators (e.g., FPGAs) [84, 125]. While this approach allows for specialized and energy-efficient designs, it also presents challenges due to differing Instruction Set Architectures (ISAs), making software development more complex and limiting task migration across processing units.

Over time, the boundary between embedded and general-purpose computing systems began to blur. Embedded systems have adopted functionalities commonly associated with personal computers, such as multitasking operating systems in mobile devices [82], and have started handling more dynamic workloads [22]. Conversely, general-purpose computing systems, facing the limitations of the TDP constraints, began adopting energy-efficient design principles traditionally associated with embedded systems [14]. This convergence has led to the emergence of Heterogeneous Multi-core Architectures (HMAs), which combine different core types within a single processor [8, 106].

These core types share the same ISA but differ in microarchitecture, offering different performance-energy trade-offs while maintaining software compatibility. This design enhances flexibility and allows for the dynamic resource allocation across core types [72]. Initially introduced in the embedded domain [8, 72], HMAs have since expanded to general-purpose computing [7, 86, 118], offering opportunities to optimize performance and energy-efficiency across domains.

However, this new form of heterogeneity also brings new challenges in resource allocation and application design [103]. Dynamic and unpredictable workloads, now prevalent in both embedded and general-purpose domains, demand flexibility and *adaptivity* at both *resource management* and *application* levels. Resource managers must account for the differing characteristics of heterogeneous cores when scheduling and distributing workloads, while applications must adapt not only to dynamically changing resource allocations but also to the heterogeneity of the assigned cores. Traditional methods may not be well-suited for fully leveraging the capabilities of these heterogeneous systems [163], requiring new strategies for scheduling, workload distribution, and application design.

This thesis explores the adaptivity challenges posed by HMAs and proposes comprehensive solutions at both the resource management and application levels. By addressing these challenges, the work aims to enhance performance and energy efficiency of modern processors, spanning embedded to general-purpose computing systems.

1.1 HETEROGENEOUS MULTI-CORE ARCHITECTURES

Heterogeneous Multi-core Architectures (HMAs) were initially introduced in mobile and embedded devices, with Arm's big.LITTLE architecture [8], which features two core clusters — one with high-performance

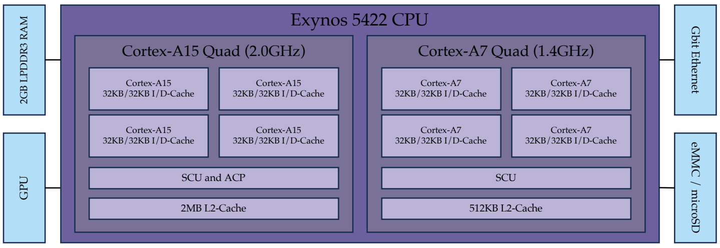

Exynos 5422 CPU
Cortex-A15 Quad (2.0GHz)
Cortex-A15 32KB/32KB I/D-Cache
Cortex-A15 32KB/32KB I/D-Cache
Cortex-A15 32KB/32KB I/D-Cache
Cortex-A15 32KB/32KB I/D-Cache
SCU and ACP
2MB L2-Cache
Cortex-A7 Quad (1.4GHz)
Cortex-A7 32KB/32KB I/D-Cache
Cortex-A7 32KB/32KB I/D-Cache
Cortex-A7 32KB/32KB I/D-Cache
Cortex-A7 32KB/32KB I/D-Cache
SCU
512KB L2-Cache
2GB LPDDR3 RAM
GPU
Gbit Ethernet
eMMC / microSD

Figure 1.1: Samsung Exynos 5422 CPU implementing the Arm's big.LITTLE architecture with two core clusters: a four-core Cortex-A15 (big) cluster and a four-core Cortex-A7 (LITTLE) cluster.

cores and one with energy-efficient cores (see Figure 1.1). This design later evolved into Arm's DynamIQ, which supports more flexible core configurations and more core types [9, 203]. Since then, HMAs have been adopted in desktop systems, beginning with Apple's M1 chip [7], and followed by incorporation into powerful x86 processors for desktops and servers, such as Intel's Alder Lake and Raptor Lake [86], as well as AMD's Phoenix 2 [29, 118] and Strix Point [79] processors. Most of these processors feature two types of cores, but different vendors use various names for these core types: Arm refers to them as "big" and "LITTLE" cores, while Intel labels them as "P-cores" and "E-cores".

The performance-energy differences between core types arise from their microarchitectures. Energy-efficient cores typically have simpler designs with shorter pipelines, in-order execution, and smaller caches, which contribute to lower power consumption but also lower single-thread performance. On the other hand, high-performance cores often feature deeper pipelines, out-of-order execution, larger caches, and other features like Simultaneous Multithreading (SMT), enabling higher performance but at the cost of increased power consumption.

This difference is also reflected in the die area. As illustrated in Figure 1.2, high-performance P-cores on modern Intel Raptor Lake processors occupy significantly more space on the die than energy-

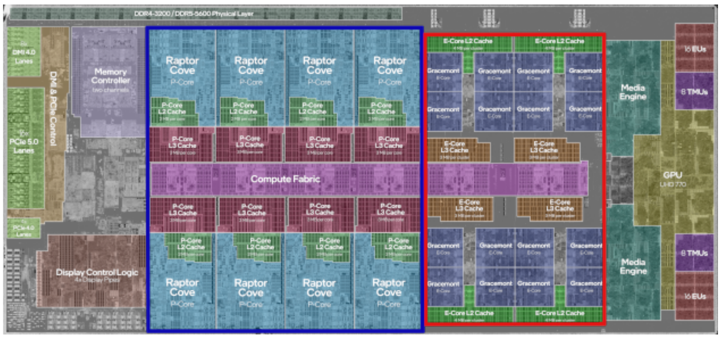


Figure 1.2: Labeled die shot of an Intel Raptor Lake Core i9-13900K. P-cores are located in the blue rectangle, and E-cores in the red one. Original die shot from [89].

efficient E-cores, even though the number of E-cores is larger than that of P-cores. The larger area required by P-cores also results in higher static power consumption.

Despite these microarchitectural differences, both core types share the same ISA, ensuring compatibility at the software level. This allows the Operating System (OS) to schedule and migrate tasks seamlessly across them, enabling dynamic trade-offs between performance and energy consumption. This flexibility allows systems to optimize performance and energy efficiency based on current workloads and requirements. For instance, compute-intensive tasks can be assigned to high-performance cores, while background and less demanding tasks can run on energy-efficient cores, reducing power consumption without compromising user experience.

However, HMAs also introduce new challenges for efficient application execution and resource allocation. Traditional execution models and resource allocation strategies may not be sufficient to fully leverage the capabilities of HMAs. This necessitates new approaches to application design, scheduling, and resource management that can exploit the advantages of heterogeneous cores.

In the following section, we explore the specific adaptivity challenges posed by modern HMAs and highlight the need for adaptive solutions that can fully leverage their potential.

1.2 NEED FOR ADAPTIVITY

As already mentioned, modern systems, both embedded and general-purpose, often execute dynamic and unpredictable workloads. Applications may start and stop unpredictably, triggered either by user interaction in general-purpose systems or by external sensors and events in embedded systems. Moreover, applications exhibit varied behaviors: some are compute-intensive and others are memory-intensive. Workloads also vary in priority, ranging from high-priority tasks to less critical background processes. The heterogeneity of input data also contributes to the unpredictability of resource demands.

At the same time, systems must ensure performance requirements are met. Embedded applications often have strict deadlines, while non-real-time applications in general-purpose systems, though not bound to explicit deadlines, must maintain acceptable performance to meet user expectations. Energy efficiency has also become a critical consideration across both domains.

To meet these demands, systems must balance energy efficiency and performance while being flexible and *adaptive* to dynamic workloads. These adaptivity requirements can be categorized at two levels: the *resource management level* and the *application level* (see Figure 1.3).

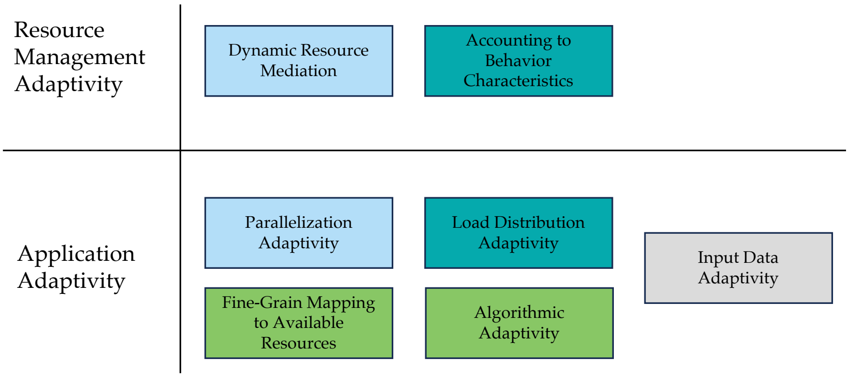

Resource Management Adaptivity
Dynamic Resource Mediation
Accounting to Behavior Characteristics
Application Adaptivity
Parallelization Adaptivity
Load Distribution Adaptivity
Input Data Adaptivity
Fine-Grain Mapping to Available Resources
Algorithmic Adaptivity

Figure 1.3: Some adaptivity challenges at the resource management and application levels brought by modern processors.

1.2.1 *Adaptivity at the Resource Management Level*

Since the introduction of multi-core systems, the system has to perform the resource management tasks: *space-sharing*, allocating tasks across processor cores. Traditionally, resource management was integrated within OS schedulers, which were previously focused on dividing processor time among tasks through *time-sharing*.

Resource managers must dynamically consider current workloads and system conditions when making decisions. This dynamism is necessary because systems often run multiple applications concurrently, with applications starting and stopping at any time, leading to fluctuating resource demands and necessitating dynamic reallocation of processor cores. The Resource Manager (RM) must adaptively assign cores to applications, ensuring efficient utilization of resources and maintaining system performance metrics such as overall throughput, fairness, or for real-time applications, meeting deadlines.

The introduction of HMAs adds complexity to resource allocation due to the varying performance-energy characteristics of different core types. The RM now also needs to determine which core type to allocate to each running task. This decision can be guided by different strategies, ranging from checking which of the applications are running in the background or actively used by a user [52] to analyzing applications' live behavior characteristics. For example, for memory- and I/O-bound applications, the performance gap between the core types may be minimal, making it more energy-efficient to run these applications on energy-efficient cores, while compute-intensive applications are located to high-performance cores.

Resource managers must make informed decisions about assigning tasks to appropriate core types. These decisions should optimize performance and energy efficiency while adapting to the dynamic workload, which requires consideration of the applications' characteristics and how they interact with different core types.

1.2.2 *Adaptivity at the Application Level*

Since the RM changes the resource allocation dynamically, each application may have access to different processor resources at different times. However, many applications are not designed to dynamically adjust their resource usage, whether by modifying their parallelization degree or accounting for heterogeneity in processor cores. Moreover, applications are often *unaware* of the specific processors on which they are executed, as the RM decisions are not explicitly communicated to them. To fully utilize processor resources, applications must adapt to the processor cores allocated by the RM.

Parallel applications typically spawn multiple threads to parallelize execution, such as data-parallel applications that distribute data processing among worker threads. Ideally, the number of threads should align with the available processor cores. This alignment helps to prevent inefficiencies such as the contention among threads for the same core. To handle the dynamic availability of processor cores, applications must be capable of adjusting their parallelization degree at runtime, scaling down their resource usage when cores are needed by other applications, and scaling up when additional resources become available.

Heterogeneous processors introduce additional challenges for applications, requiring adaptivity in workload distribution, application topology, or even algorithms. For instance, if parallel threads are distributed across both high-performance and energy-efficient cores, the slower threads on energy-efficient cores can create a bottleneck, causing load imbalance and underutilization of high-performance cores, thereby reducing overall performance.

In some cases, these adaptivity challenges can be addressed agnostically, without requiring the application to know which threads are assigned to which core types. For example, workload distribution can be dynamically managed using techniques like work-stealing in shared task pools (e.g., in OpenMP [160]). However, other solutions may require explicit awareness of the processor cores available to the application. This includes the ability to adjust the parallelization degree, reconfiguring application topology, or selecting algorithms optimized for specific core types [155, 156, 166].

Apart from adapting to available resources, applications may also adapt to their input data, which contributes to the unpredictability of resource demands. Input data can significantly influence an application's behavior [153, 165, 184]. For example, a WLAN application may switch between *send* and *receive* modes depending on input, altering the active component within the application. Internally, applications are often designed to adapt to input data, but these adaptations can result in varying execution characteristics and the resource demands.

Thus, to fully utilize processor resources, there is a need for a communication mechanism to ensure applications are aware of the available processor cores. Conversely, RM should also be aware of the dynamic characteristics of applications, to make more informed resource allocation decisions.

1.3 CONTRIBUTIONS OF THIS THESIS

The adaptivity challenges posed by modern systems executing dynamic workloads require comprehensive and flexible solutions. While no single solution can fully address the adaptivity challenges of HMAs, this thesis proposes a series of approaches that enhance adaptivity in different scenarios, targeting both resource management and application behavior.

In the embedded domain, one promising state-of-the-art approach is *Hybrid Application Mapping* (HAM), which combines design-time application optimization with runtime resource management that dynamically adapts resource allocations based on current workloads. By integrating extensive offline analysis with online decision-making, HAM effectively addresses adaptivity challenges at the resource management level, such as dynamic resource mediation and accounting for unique application characteristics. Building upon the advantages of HAM, this thesis adopts it as a central methodology and enhances it to further improve adaptivity in resource management.

Beyond resource management, this thesis also examines adaptivity challenges at the application level, focusing on dataflow Models of Computation (MoCs). Furthermore, it proposes a coordinated mechanism that integrates adaptivity at both resource management and application levels.

The following subsections outline the key contributions of this thesis.

Spatio-Temporal Mapping in Real-Time Systems

This thesis tackles the challenge of efficiently mapping parallel applications in dynamic real-time environments where workloads may start sporadically. Each application is associated with a specific deadline, and the RM must ensure that all applications complete within their time constraints. State-of-the-art approaches typically generate only *spatial* mappings, which limit the overall performance and energy efficiency when responding to workload changes.

To overcome the limitations, this work proposes a novel resource management approach that generates *spatio-temporal* mappings. This decision model allows the resource manager to not only allocate resources spatially but also strategically postpone execution or adjust mappings at a later time, accounting for foreseen workload changes. Two efficient algorithms are presented to generate energy-efficient spatio-temporal mappings within milliseconds. Additionally, a memetic algorithm generating near-optimal solutions is proposed to validate and demonstrate the efficiency of the rapid heuristics.

Domain-Specific Hybrid Mapping for Baseband Processing

Baseband processing is a critical component of wireless communication systems. Modern standards demand high flexibility to handle heterogeneous and dynamic workloads. This thesis introduces a domain-specific HAM methodology tailored to the phased-sequential structure of baseband receiver applications, which comprise hundreds of tasks, with execution behavior depending on specific input data parameters.

The proposed approach leverages this phased structure to efficiently exploit processor heterogeneity. It presents a fast algorithm that generates optimal mappings accounting for the performance-energy characteristics of different cores. Additionally, a refined runtime resource allocation algorithm is proposed to utilize prior decisions, reducing runtime overhead, which is highly critical for low-latency baseband systems, and ensuring efficient resource utilization and improved performance.

Adaptive Process Networks

While the previous contributions focus on resource management through mapping selection, applications themselves must also adapt to dynamically allocated processor resources. This thesis examines application-level adaptivity within the Kahn Process Networks (KPNs), a MoC commonly used in embedded systems for streaming and dataflow applications due to their deterministic properties, expressiveness, and pipeline parallelism. Traditional KPNs, however, suffer from rigid application topology that limits their ability to adapt dynamically.

This thesis proposes *Adaptive Process Network* (APN), an implicit data-parallel extension to KPNs. This extension enables runtime replication of process subnetworks, allowing applications to adapt their topology dynamically. It also introduces parallel channels that dynamically distribute workloads among processes and merge results without compromising application determinism. The Dynamic Process Manager (DPM) library is developed to provide runtime support for APN applications and facilitate communication with the RM, enabling application awareness of the assigned resources.

Coordinated Adaptivity in General-Purpose Environments

To fully address adaptivity challenges, resource management and application-level adaptivity must be considered together and, therefore, coordinated. This thesis proposes HARP, a resource management framework integrated into Linux, which provides a unified resource allocation solution across various desktop system with heterogeneous processors.

HARP coordinates resource management across diverse application models through the `libharp` library, which facilitates communication with the RM via a well-defined interface. This enables applications to internally adapt to allocated resources, such as adjusting their parallelization degree upon receiving the RM decisions. The `libharp` library supports common application models like OpenMP and Intel TBB, while also allowing customization, for instance, for APN applications.

This contribution also expands the scope of the thesis to powerful desktop and server systems. This shift necessitates overcoming a key limitation of HAM: its reliance on offline analysis to generate Pareto-optimal application configurations. Since general-purpose systems may run unknown applications without pre-generated configurations, this thesis introduces a runtime algorithm that monitors application behavior to quickly identify optimal configurations.

In summary, this thesis advances the state-of-the-art by addressing adaptivity challenges at both the resource management and applications levels. The initial contributions of this thesis focused on embedded systems, where HMAs were first introduced. With HMAs now being adopted by major processor manufacturers for general-purpose systems, this work has evolved to address the adaptivity challenges in desktop and server environments as well.

A Note on Originality

The thesis primarily focuses on contributions that result from my own research. However, some of the presented work result from collaborative efforts. I believe that outstanding research often results from exchanging of ideas and collaboration. However, it can sometimes be challenging to attribute specific ideas or results to a single individual. To ensure transparency, I have made an effort throughout the thesis to clearly distinguish my contributions from those of collaborators. Additionally, I have indicated when the presented material appeared in joint publications. If in doubt, any idea or result that I have included here which has already been published elsewhere is also due to my coauthors.

1.4 SYNOPSIS AND OUTLINE

This chapter has provided an overview of the evolution of modern processors, highlighting the transition from single-core designs to multi-core processors and, more recently, to heterogeneous multi-core architectures. It explored the opportunities and challenges brought by heterogeneous processors, emphasizing the need for adaptivity to fully exploit their capabilities. The chapter also provided an overview of this thesis's contributions, demonstrating how the identified adaptivity challenges are addressed at both the resource management and application levels, targeting both embedded and general-purpose computing systems.

The remainder of this thesis is organized as follows. The next chapter, Chapter 2, provides essential background information on application mapping and hybrid application mapping methodologies. The chapter also formalizes the system model and the optimization problems addressed in this thesis. Following this, Chapter 3 reviews the state-of-the-art, discussing relevant work in relation to the contributions of this thesis. The core contributions of the thesis are presented in Chapters 4 to 7, which cover spatio-temporal mapping for real-time systems, domain-specific hybrid mapping for baseband processing, the Adaptive Process Network (APN) application model, and a resource management approach coordinating adaptivity at both levels in general-purpose environments. Finally, Chapter 8 draws conclusions and outlines potential future research directions.

2

FOUNDATIONS OF APPLICATION MAPPING ONTO HMAS

The previous chapter highlighted the need for adaptivity in modern heterogeneous multi-core processors and provided an intuitive overview of the Hybrid Application Mapping (HAM) methodology, emphasizing the existing gaps in fully utilizing Heterogeneous Multicore Architectures (HMAs). To provide a foundation for the subsequent chapters, this chapter introduces the key concepts, formal models, and tools relevant to our work, offering detailed information about application mapping to heterogeneous processors.

The chapter begins with Section 2.1, introducing basic terminology, discussing performance and energy estimation, and presenting the notation used throughout the thesis. Section 2.2 formalizes the system model. Section 2.3 explores the dataflow Model of Computation (MoC), including its mapping to heterogeneous multi-core architectures. Section 2.4 introduces the Hybrid Application Mapping (HAM) methodology, presenting its core principles and formalizing the optimization problems addressed in this thesis. Section 2.5 describes the Mocasin framework, a rapid prototyping framework that is used in this research for modeling, simulating, and analyzing application mappings. The chapter ends with a summary in Section 2.6.

2.1 PRELIMINARIES

Before introducing the formal problem definitions, this section introduces the fundamentals of mapping, scheduling, and spatio-temporal mapping, basic concepts of performance and energy estimation, static and dynamic power consumption, and the notation used throughout this thesis.

2.1.1 *Mapping, Scheduling, and Spatio-Temporal Mapping*

Efficient utilization of processing resources in multi-core and heterogeneous computing systems requires careful allocation of tasks to processor cores and the management of their execution over time. This involves two fundamental concepts: *mapping* and *scheduling* [42]. These terms recur throughout this thesis and are essential for understanding the proposed resource management strategies. Additionally, the concept of *spatio-temporal mapping* is a key aspect central to this work. Therefore, it is crucial to clearly define and distinguish them.

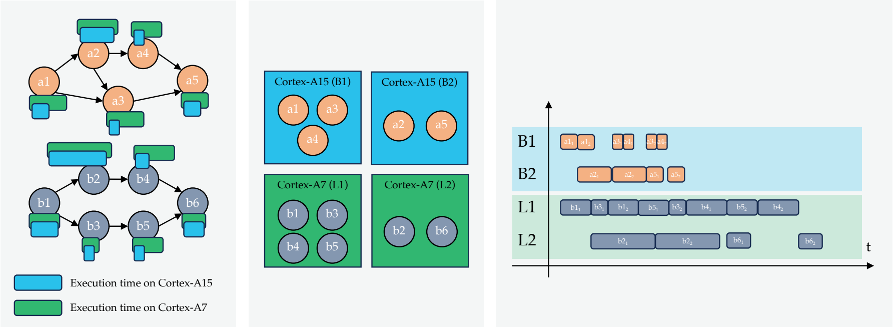
Mapping is the process of assigning tasks to processor cores, determining which tasks run on which cores. This process can be performed either statically or dynamically. Static mapping determines task-to-core allocations at design time, remaining fixed during execution. It is common in systems with predictable workloads where timing guarantees are critical, such as real-time and embedded systems. In contrast, dynamic mapping assigns tasks to cores at runtime according to a *mapping policy*, allowing the system to adapt to changing workloads and resource availability.

Scheduling process decides on the order and timing of task execution on allocated cores, determining at which time instants tasks are to be run, managing the execution of multiple tasks that may compete for the same processing core. Similar to mapping, scheduling can also be static or dynamic. Static scheduling determines task order and timing at design time, providing precise control over task execution but lacking flexibility to adapt to dynamic workloads or varying system condition. Dynamic scheduling makes decisions about task execution at runtime based on *scheduling policies*, allowing the system to respond to unpredictable workloads.

Mapping and scheduling can either be coupled or decoupled. A decoupled mapper dispatches tasks to local schedulers on each processor core, which then decide on task execution. This approach might, for example, combine a fixed mapping strategy with a dynamic scheduling policy such as *First-Come, First-Served* (FCFS), where tasks are executed in the order they arrive at the scheduler [172]. In a coupled approach, both the *where* (mapping) and the *when* (scheduling) decisions are made jointly. An example is the *Earliest Finishing Time* (EFT) scheduling strategy, where tasks are assigned to processor cores in a way that minimizes their earliest finish time, making task-to-core assignment and scheduling order decisions simultaneously.

Alongside mapping and scheduling, the concept of spatio-temporal mapping play a central role in this thesis. *Spatio-temporal mapping* extends the mapping by introducing a temporal component and is defined as a sequence of *mapping segments*, where each segment specifies task-to-core allocation (the mapping) and the duration for which this mapping is active. In each mapping segment, multiple tasks may be assigned to the same core (as in conventional *spatial* mapping), but the specific order and timing of task execution within the segment are governed by scheduling, which may be governed by dynamic scheduling policy at runtime. While scheduling details the precise task timing and sequence (including context switches and preemptions), spatio-temporal mapping operates at a higher abstraction level, providing a time-dependent task allocation plan without specifying detailed scheduling decisions.

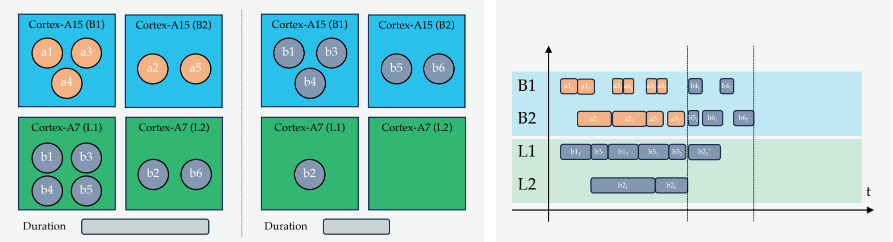
Figure 2.1 illustrates the concepts of mapping, scheduling, and spatio-temporal mapping. In this example, two applications, each rep-



(a) Sample Applications

(b) (Spatial) Mapping

(c) Mapping with FCFS Scheduling



(d) Spatio-Temporal Mapping

(e) Spatio-Temporal Mapping with FCFS Scheduling

Figure 2.1: Mapping, scheduling, and spatio-temporal mapping of two dataflow applications

resented as a Directed Acyclic Graph (DAG) and shown in Figure 2.1a, perform computations for two sample input data on HMA consisting of two Cortex-A15 (big) and two Cortex-A7 (LITTLE) cores. The colored bars attached to the nodes represent the execution time on each processor core, with communication costs ignored for simplicity.

Figure 2.1b shows spatial mapping, where tasks from the first application are mapped to the big cores, and tasks from the second application are mapped to the LITTLE cores. These tasks are then scheduled with the FCFS policy, shown in a *Gantt Chart* in Figure 2.1c, where time is plotted on the horizontal axis and the processor cores on the vertical axis.

Figure 2.1d and Figure 2.1e illustrate an alternative approach utilizing spatio-temporal mapping, with two mapping segments shown in Figure 2.1d. The first mapping segment is the same as the spatial mapping, while the second reassigns some tasks from the second application to the big cores as they become available. Scheduling within each segment still follows the policy (as shown in Figure 2.1e), but the overall execution benefits from the dynamic adjustment of task mappings over time.

The resource management algorithms in this thesis focus primarily on mapping techniques — both spatial and spatio-temporal — without delving into the specifics of scheduling policies. During Design Space Exploration (DSE) and performance evaluation, a scheduling policy is assumed implicitly to assess the effectiveness of the mapping strategies. Therefore, *mapping* in this context refers broadly to both task assignment and the underlying scheduling policy governing task execution.

2.1.2 *Performance and Energy Estimation*

Evaluating the effectiveness of application mappings (i.e., how applications perform under different mappings regarding performance and energy efficiency) on HMAs is essential to optimizing resource utilization, system performance, and energy efficiency. Common performance metrics include *execution time*, *throughput*, and application-specific *utility*, which may be defined based on the application's goals [101]. Energy metrics include *energy consumption*, *average power consumption*, and *peak power consumption*. Energy and performance metrics can be combined, resulting in metrics like the *Energy-Delay Product* (EDP), which considers both energy consumption and execution time [124, 144].

Various methods exist for assessing performance and energy metrics, depending on the application model, target architecture, available tools, and desired accuracy. Common evaluation methods include [42]:

- *Annotation-Based*: Estimates performance and energy by annotating code with expected values based on prior knowledge.
- *Source-Level Instrumentation*: Instruments source code to collect information about the execution of elementary operations, retrieving the costs of these operations from a table. While useful, this method may not accurately model complex operations, non-scalar architectures, or the effects of compiler optimizations [28].
- *Simulation-Based*: Uses simulators to model execution of the application on target hardware. Performance simulators estimate execution time by modeling processor behavior, while energy simulators use energy models to estimate energy consumption [31, 194].
- *Measurement-Based*: Measures actual performance and energy consumption by running applications on target hardware. This approach is potentially the most accurate but requires access to the real hardware. Energy measurements can be obtained using built-in sensors or external measurement tools, which vary in temporal and spatial granularity, accuracy, and cost [85].

This thesis relies primarily on measurement-based methods due to their accuracy. When possible, applications are executed on real hardware platforms equipped with energy measurement capabilities.

Specifically, we employ Odroid-XU3 [1] and Odroid-XU4 [78] boards; both feature Samsung Exynos 5422 CPUs (Figure 1.1), but only the

---

[1] Currently discontinued, `https://www.hardkernel.com/shop/odroid-xu3`

Odroid-XU3 has onboard INA231 energy sensors[2] to monitor the big and LITTLE cores, Graphics Processing Unit (GPU) and DRAM individually. For Odroid-XU4, we use the ZES Zimmer LMG450 power meter, which measures the power consumption of the entire system at a sampling rate of 20 Sa/s [218].

For the Intel Raptor Lake system, energy data is collected using the *Running Average Power Limit* (RAPL) interface, which provides data with high temporal (as fine as 1 ms) and spatial (RAM, CPU per socket) granularity [54, 85]. The accuracy of RAPL for fine-grained energy measurements has been validated in several studies [51, 74, 75, 164, 179].

When direct measurement is not feasible — for instance, when evaluating a virtual target architecture — we utilize simulation-based methods. For KPN applications, we employ trace-based simulation using the *Mocasin* tool (Section 2.5) to estimate performance and energy consumption based on collected execution traces [42].

2.1.3 *Static and Dynamic Power Consumption*

Processor power consumption consists of static and dynamic components:

$$P := P_{\text{static}} + P_{\text{dynamic}} \tag{2.1}$$

- *Static Power Consumption*: Power consumed due to leakage currents even when the processor is idle.

  $$P_{\text{static}} := I_q \cdot V_{\text{dd}} \tag{2.2}$$

  where I_q is leakage current and V_{dd} is supply voltage.

- *Dynamic Power Consumption*: Power consumed due to active computation and switching activities within the processor.

  $$P_{\text{dynamic}} := \gamma \cdot \hat{C}_{\text{eff}} \cdot V_{\text{dd}}^2 \cdot f_{\text{op}} \tag{2.3}$$

  where $\gamma \in [0, 1]$ is average switching activity, $\hat{C}_{\text{eff}}$ is effective capacitance, and f_{op} is operating frequency.

Distinguishing between static and dynamic power is important for evaluation the application's energy efficiency, especially in multi-application scenarios, to avoid double-counting static energy when evaluating the total energy consumption.

Accurate modeling of static power is challenging, as it depends on both voltage and temperature. Notably, static power is not equivalent to the system's idle power consumption: When the platform is idle (executing no applications), power consumption is reduced due to

---

[2] `https://www.ti.com/product/INA231`

power optimization techniques like power and clock gating, where parts of the processor are powered down or clock signals are disabled to save energy [154].

To estimate the static power, we apply regression models to correlate measured power and the number of active cores. By extrapolating to zero active cores, we can approximate the system's static power consumption [80]. By distinguishing between static and dynamic power, we can avoid overestimating energy consumption when aggregating results from multiple applications.

2.1.4 *Notation*

This thesis employs a consistent and intuitive notation system to describe various elements in the system model, problem formulations, and algorithm descriptions. The notation is designed to resemble familiar programming concepts to readers.

- *Objects and Attributes*: An object X with specific attributes is denoted as $X\langle a_1, a_2, \ldots, a_n \rangle$, where $a_1, a_2, \ldots, a_n$ are the attributes of the object X. In some cases, objects may be annotated with contextual information, to indicate, for example, their relationship to another object, e.g., $X^C\langle \ldots \rangle$, where C is an abstract context. For simplicity, the object can be referred to as X, with its definition indicated by $:=$, i.e., $X := X^C\langle a_1, a_2, \ldots, a_n \rangle$.

- *Accessing Attributes*: Attributes of an object X are accessed using the square brackets, denoted as $X[a_i]$ for the attribute a_i. This notation is analogous to accessing elements in a data structure.

- *Derived Information*: Some objects may have associated derived values or other related information, which are accessed similarly to attributes. These derived values are also accessed using square brackets and are defined using $:=$. For example, given an object X with two parameters t_{st} and t_{fin} representing the start and finish times, i.e., $X := X\langle t_{\text{st}}, t_{\text{fin}} \rangle$. A derived value, such as the duration d, can then be defined as:

  $$X[d] := X[t_{\text{fin}}] - X[t_{\text{st}}]$$

2.2 SYSTEM MODEL

This section provides formal definitions for the architecture, application, and mapping models that serve as the foundation for the approaches presented in this thesis.

2.2.1 *Architecture*

A Heterogeneous Multi-core Architecture (HMA) consists of processing elements of different types, each exhibiting distinct characteristics, as well as the interconnection mechanisms that enable communication between them.

Definition 2.1 (Processing Element Type)**.** A *processing element type* Ω represents a class of computational units that are instantiated, possibly multiple times, in the HMA. Each processing element type is characterized by its *Instruction Set Architecture* (ISA), performance-energy cost model, and other hardware attributes such as the number of hardware threads if it supports Simultaneous Multithreading (SMT). Formally, a processing element type is defined as $\Omega := \Omega\langle \text{isa}, \text{cm}, X\rangle$, where:

- isa specifies the underlying ISA used by the processing element type.
- cm is an abstract *cost model* specifying power consumption and performance characteristics.
- X is a set of additional attributes providing information about the processing element type. For example, $x_{\text{smt}} \in X$ specifies the number of simultaneous hardware threads.

Different processing element types can share the same ISA while differing in other characteristics. With appropriate OS support, a task can be migrated between processing elements of types Ω_i and Ω_j if they share the same ISA, denoted as $\Omega_i \sim \Omega_j$.

The processing element types available in the HMA are represented as a finite sequence $(\Omega_1, \Omega_2, \ldots, \Omega_{|(\Omega_i)|})$, or concisely as (Ω_i), where $|(\Omega_i)|$ is the number of processing element types.

Each processing element type Ω could represent either a general-purpose processor core or an application-specific accelerator. Where needed, we explicitly distinguish between them.

Definition 2.2 (Processing Element)**.** A *processing element* ψ is an instance of a given processing element type Ω, denoted as $\psi := \psi\langle\Omega\rangle$. It inherits the ISA, cost model, and attributes from its processing element type Ω. The set of all processing elements in the HMA is denoted Ψ.

Although the performance-energy cost model $\Omega[\text{cm}]$ is left abstract, we may denote specific power consumption metrics for processing elements ψ where needed. Specifically, for a processing element ψ, we denote $\psi[P_{\text{static}}]$, $\psi[P_{\text{dynamic}}]$ and $\psi[P]$ as the static, dynamic and total power consumption, respectively.

Definition 2.3 (Platform)**.** A *platform* $\mathcal{P}$ for an HMA is defined as $\mathcal{P} := \mathcal{P}\langle\Psi, \mathcal{IC}\rangle$. Ψ is the set of processing elements available in the platform and $\mathcal{IC}$ is an abstract interconnection model characterizing inter-processor communication.

For convenience, let $\mathcal{P}[\omega_i]$ denote the number of processing elements of type Ω_i, i.e.,

$$\mathcal{P}[\omega_i] := |\{\psi \in \mathcal{P}[\Psi] \mid \psi[\Omega] = \Omega_i\}| \tag{2.4}$$

Let $\mathcal{P}[\vec{\omega}]$ denote the *resource vector* of the platform $\mathcal{P}$:

$$\mathcal{P}[\vec{\omega}] := \Big[\mathcal{P}[\omega_1], \mathcal{P}[\omega_2], \ldots, \mathcal{P}[\omega_{|(\Omega_i)|}]\Big]^\top \tag{2.5}$$

This thesis focuses on HMAs such as ARM big.LITTLE (Figure 1.1) or the Intel Alder Lake and Raptor Lake families (Figure 1.2). These systems feature two types of processor cores: high-performance cores (referred to as "big" in ARM, and "P-core" in Intel) and energy-efficient cores (referred to as "LITTLE" in ARM and "E-core" in Intel).

In the proposed algorithms, we assume that all cores of the same type are symmetrical from an interconnection point of view. This assumption is valid for the ARM big.LITTLE platform, which features a single cluster for big cores and a single cluster for LITTLE cores. For Intel's architecture, this assumption is partially true: while each P-core is located in its own cluster, E-cores are grouped into several clusters with four E-cores per cluster. The difference lies in the fact that two E-cores within the same cluster can communicate starting via the L2 cache (which is the closest shared cache level), while communication between E-cores in different clusters begins through the L3 cache.

2.2.2 *Application*

The HMA executes multiple applications that may vary in their MoC. In addition to information about application components (tasks or threads), applications may also provide a way to control their internal configuration through so-called *adaptivity knobs*. By controlling these adaptivity knobs, the RM can change the application topology, the degree of parallelization, the algorithms used, and more. Along with resource allocation, the RM may leverage these knobs to better adapt the application to the current workload.

This section defines an abstract application model that allows the formulation of the application mapping problem in a general way.

Definition 2.4 (Application)**.** An *application* is a tuple $A := A\langle\mathcal{M}, \Gamma, \mathcal{V}\rangle$, where

- $\mathcal{M}$ is an abstract model of the application.
- Γ is a set of configuration parameters.

- $\mathcal{V}$ is a set of *application components* (e.g., tasks, processes, threads) that need to be mapped.

A *configuration* γ is an assignment of values to the configuration parameters in Γ. For each configuration γ, the corresponding set of active application components is denoted by $\mathcal{V}_\gamma \subseteq \mathcal{V}$. The set of all applications is denoted by $\mathcal{A}$.

This definition deliberately separates the application parameters into configuration parameters and application components. The *configuration parameters* specify the internal settings of the application, such as the degree of parallelism or algorithmic choices. The *application components* represent the computational elements (e.g., tasks or threads) that need to be mapped onto the platform's processing elements.

As an example, consider a simple parallel application where the configuration parameter is the number of threads $\Gamma = \{n_{\text{threads}}\}$. A specific configuration $\Gamma = \{n_{\text{threads}} = k\}$ determines the number of threads used by the application. The set of application components V_γ would then consists of k threads that need to be mapped, i.e., $|V_\gamma| = k$.

2.2.3 *Mapping*

As discussed in Section 2.1.1, mapping can be performed either statically or dynamically. This work focuses on static assignments of tasks to processing elements, which can be defined formally as follows.

Definition 2.5 (Mapping). A *mapping* μ for application $A\langle\mathcal{M},\Gamma,\mathcal{V}\rangle$ with a selected application configuration γ on a platform $\mathcal{P}\langle\Psi,\mathcal{IC}\rangle$ is a specific assignment of each application component $v \in A[\mathcal{V}_\gamma]$ to exactly one core $\psi \in \mathcal{P}[\Psi]$ in the HMA, i.e.,

$$\mu : A[\mathcal{V}_\gamma] \to \mathcal{P}[\Psi] \tag{2.6}$$

The set of processing elements utilized by the mapping is referred to as the *allocation* $\mu[\Psi]$, defined as:

$$\mu[\Psi] := \{\psi \mid \exists v \in A[V_\gamma] : \psi = \mu(v)\} \tag{2.7}$$

The *resource vector* $\mu\,[\vec{\omega}]$ specifies how many instances are allocated per processing element type:

$$\mu[\vec{\omega}] := \Big[\mu[\omega_1], \dots, \mu[\omega_{|(\Omega_i)|}]\Big]^\top \tag{2.8}$$

$$\text{where } \mu[\omega_i] := |\{\psi \in \mu[\Psi] \mid \psi[\Omega] = \Omega_i\}| \tag{2.9}$$

2.3 MAPPING OF DATAFLOW APPLICATIONS

Building upon the system model introduced in the previous section, this section discusses dataflow Models of Computation (MoCs), which are common application models for streaming applications used in embedded systems. These models naturally represent the flow of data between computational components [107]. Particularly, the section introduces common MoC such as Synchronous Dataflow (SDF), and Kahn Process Network (KPN). We define these dataflow MoCs, discuss their mapping onto platforms, and outline the trace-based simulation used for performance and energy analysis.

2.3.1 *Dataflow Models of Computation*

Dataflow Models of Computation (MoCs) have gained momentum in the embedded domain, especially for describing streaming and signal processing applications. These models represent computations as directed graphs where the nodes denote computational tasks, and the arcs represent communication channels [107]. Depending on the specific model, these nodes may be referred to as *actors* or *processes*. Each node performs computations based solely on its input values.

Dataflow models have become attractive for several reasons: they explicitly expose parallelism, are well-suited for graphical programming — a common specification paradigm for signal processing algorithms — and the properties of the underlying MoC enable tools to perform analysis and compile the specifications into both software and hardware [37].

Various dataflow models have been proposed for embedded programming, each offering different trade-offs between expressiveness and analyzability. Examples include *Synchronous Dataflow* (SDF) [109], *Cyclo-Static Dataflow* (CSDF) [21], *Boolean Dataflow* (BDF) [108], *Dynamic Dataflow* (DDF) [33], *Process Network* (PN) and *Kahn Process Network* (KPN) [91].

2.3.1.1 *Synchronous Dataflow*

Synchronous Dataflow (SDF), introduced by Lee and Messerschmitt [109], is a dataflow model where the nodes, called *actors*, have the amount of data produced and consumed by each actor firing fixed and known at compile time. The execution of an SDF (*firing*) is controlled by data in its edges; each actor executes or *fires* once it has enough tokens in its input channels. This deterministic behavior makes SDF models statically analyzable: properties such as deadlock-freedom, bounded memory usage, and throughput can be analyzed at design time, making SDF suitable for real-time and resource-constrained applications.

If each actor in SDF consumes and produces exactly one token per firing, the graph is referred to as a *Homogeneous SDF* (HSDF). Any SDF graph can be transformed into an equivalent HSDF graph [186]. Acyclic HSDF (i.e., those without cycles) are structurally equivalent to Directed Acyclic Graphs (DAGs) and can effectively model *task graphs*.

2.3.1.2 *Kahn Process Networks*

Kahn Process Network (KPN), first introduced by Gilles Kahn [91], is a more general and expressive Model of Computation (MoC). In this model, each node executes concurrently as a separate *process*, typically a non-terminating program that reads data tokens from input unbounded First In, First Out (FIFO) channels and writes data tokens to output unbounded FIFO channels. Unlike actors in dataflow models such as SDF, processes in KPN do not have firing semantics; instead, they are either *ready* to execute or *blocked* waiting for input. All processes execute simultaneously [110].

Communication between processes is characterized by *blocking reads* and *non-blocking writes* (also referred as Kahn-MacQueen execution semantics [92]). A process attempting to read from an empty channel will block until data becomes available, whereas writing to a channel is non-blocking as channels are assumed to be unbounded in size.

Importantly, processes cannot check whether data is available on a channel without attempting to read from it. Due to this restriction and the deterministic behavior of each process, the entire KPN is also deterministic. Regardless of the relative speeds of processes or communication delays, the overall system behavior remains predictable and consistent for any given set of input sequences.

KPNs are Turing-complete, meaning they can represent any computation that a Turing machine can perform [33]. This makes them expressive model and capable of representing complex applications. Task graphs and SDF can be considered as restrictions of Kahn Process Networks [107].

Relating KPNs to the application model, defined earlier (Definition 2.4), we observe that for a KPN, the application model $\mathcal{M}$ is a directed multigraph (V, E), where V is the set of processes and E is the set of unbounded FIFO channels connecting them. This graph encapsulates the computational structure and data dependencies of the application. Since the structure and behavior of the KPN are fixed, there are no tunable parameters, thus, $\Gamma = \emptyset$. The set of application components $\mathcal{V}$ consists of all the processes in the KPN that need to be mapped onto processing elements in the platform.

However, in practical implementations channels cannot be truly unbounded due to physical memory limitations. As a result, channels are implemented as bounded FIFO [143]. This introduces the possibility of *blocking writes*: If a channel is full when a process attempts to write, the process must block until space becomes available. This deviation from the theoretical model can introduce deadlocks and requires careful application design and buffer size determination to avoid such issues.

As an example, Figure 2.2 depicts the implementation of a JPEG Image Encoding application [201] as a KPN graph. The `Source` node reads the input pixel data and distributes the RGB components to the `R2BRed`, `R2BGreen`, and `R2BBlue` processes. These `R2B` (Row-To-Block)

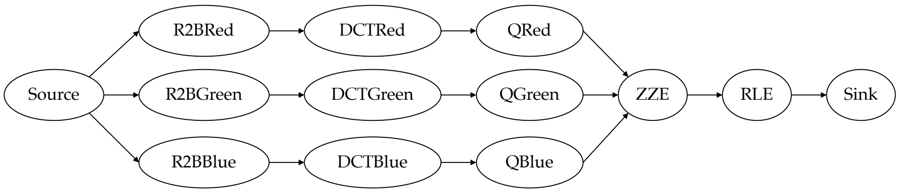


Figure 2.2: JPEG Image Encoding application as a KPN graph

processes convert the pixel data into 8×8 blocks for each color channel. The `DCT` nodes then transform the blocks from the spatial domain to the frequency domain using the *Discrete Cosine Transform*. Quantization is applied by `QRed`, `QGreen`, and `QBlue`, reducing the precision of the DCT coefficients. The *Zig-Zag Encoding* (`ZZE`) process reorders the quantized coefficients into a one-dimensional array, which is further compressed by `RLE` through the *Run-Length Encoding* algorithm. Finally, `Sink` prepares the compressed data for output.

This implementation demonstrates a key strength of the KPN: *pipeline parallelism*. The `Source` node continuously streams pixel data to the `R2B` processes, which convert this data into fixed-size blocks. While `R2B` prepares the next blocks, subsequent processes immediately begin working on the first block. As the blocks move through the pipeline, the `Source` keeps sending new pixel data, and `R2B` continues converting them into blocks. This overlap in processing across different stages creates an efficient pipeline structure that maximizes parallelism.

Due to their deterministic properties, expressiveness, and ability to exploit pipeline parallelism, KPNs are well-suited for embedded software, particularly in streaming and signal processing applications.

2.3.2 *Design-time and Runtime Mapping Approaches*

The process of mapping application tasks or processes onto multicore systems involves allocating and coordinating tasks and their communications across platform resources. The objective is to optimize criteria such as computational performance and energy consumption.

The mapping problem is similar to the *Quadratic Assignment Problem*, a well-known NP-hard problem [66]. As a result, finding an optimal solution that satisfies all constraints is computationally expensive and time-consuming. To achieve near-optimal solutions within a reasonable timeframe, heuristics, leveraging domain-specific knowledge, are frequently employed.

Extensive research has been conducted on application allocation methodologies, which can be broadly classified based on the workload scenarios they address. For static workloads, design-time mapping approaches optimize resource allocation prior to runtime, while runtime mapping methodologies dynamically allocate resources during application execution [175].

2.3.2.1 *Design-Time Mapping*

Design-time mapping determines task allocation to processor cores before application execution begins. These methods utilize extensive computational time to perform sophisticated DSE, aiming for near-optimal mappings that optimize metrics such as execution time, throughput, and energy consumption.

Common strategies in design-time mapping include domain-specific heuristics, which leverage application-specific insights to generate efficient mappings, and metaheuristics, such as Evolutionary Algorithms (EAs), which iteratively explore the search space to identify high-quality solutions. Metaheuristics are often integrated with trace-based simulations to estimate application performance and energy consumption. More details on trace-based simulation are provided in Section 2.3.3.

While design-time mapping can produce high-quality mappings, it has notable limitations. As the number of applications increases, the number of possible use cases increases exponentially, making exhaustive analysis impractical. For n applications, exhaustive analysis would require evaluation of 2^n use cases. Additionally, design-time mapping lacks flexibility in adapting to dynamic scenarios, such as changes in application standards or the addition of new applications.

2.3.2.2 *Runtime Mapping*

In contrast, the *runtime mapping* defers resource allocation decisions to runtime, enabling adaptation to the current set of executing applications. These approaches typically rely on *mapping policies* (see Section 2.1.1) implemented as lightweight heuristics to rapidly assign tasks to resources.

Although runtime mapping is well-suited for unpredictable workloads, the constraints of rapid decision-making limit the complexity of the heuristics used. This often results in suboptimal mappings, leading to inefficient resource utilization and higher energy consumption.

A detailed survey of design-time approaches is presented in Section 3.1, while runtime mapping methodologies are explored in Section 3.2.

2.3.3 *Trace-Based Simulation*

For dataflow applications such as KPNs, performance and energy consumption can be evaluated through *trace-based simulation*. This approach simulates the sequential execution of processes and their communication patterns, allowing for detailed analysis of application behavior under different mappings [41, 42].

In trace-based simulation, a KPN application is analyzed with a given input stimulus. As discussed in Section 2.3.1, KPN applications exhibit deterministic behavior, allowing us to rely on the fact that each process will follow the same sequence of operations regardless of how the application is mapped onto the target hardware.

To perform trace-based simulation, we first generate a *process trace* for each process in the KPN application. A process trace is defined as a sequence of *segments*, each representing a sequence of statements executed and ending with a communication event (a channel read or write) or process termination.

The set of all process traces forms the *application trace*, which serves as the basis for the simulation. Given a mapping μ, the trace-based simulation emulates the runtime system of the target platform, producing a Gantt chart (such as one shown in Figure 2.1c) that visually represents application execution by *replaying* the process traces. Each segment in the traces can be divided into two parts: the *computation part*, representing the sequential computations performed by the process, and the *communication event*, representing the interaction with other processes via channel reads or writes.

The computational parts are analyzed to determine execution time and dynamic energy consumption on each processor core type Ω. As discussed in Section 2.1.2, execution time and energy estimations can be obtained through techniques like source-level instrumentation, measurement-based methods, or simulation.

To formalize this, consider a process π executing a segment s, characterized by performance-energy data $\langle \vec{\tau}, \vec{\varepsilon} \rangle$. Here, $\vec{\tau} = \left[\tau_1, \dots, \tau_{|(\Omega_i)|}\right]^\top$ and $\vec{\varepsilon} = \left[\varepsilon_1, \dots, \varepsilon_{|(\Omega_i)|}\right]^\top$, where τ_i and ε_i represent the latency and energy consumption on the processing element type Ω_i. We denote $\pi[s][\tau_i]$ and $\pi[s][\varepsilon_i]$ as the latency and energy consumption of a process π executing a segment s on the processing element type Ω_i.

Communication costs are estimated by calculating the transfer time and energy based on factors such as message size and the characteristics of the communication channels between processor cores ψ_i and ψ_j. Communication cost functions may exhibit non-linear behavior, such as a step function, due to factors like buffering effects and protocol overhead.

The trace-based simulation also takes into account the dependencies between segments and communication events, including *sequential order*, *read-after-compute*, *block-reads* and *unblock-reads*, and *block-writes* and *unblock-writes* [42]. The blocking and unblocking write dependencies are introduced due to bounded channels in KPNs. The simulation also emulates scheduling policies and events such as context switches, to reflect the behavior of the target system.

At the end of the simulation, the total execution time of the application is measured from the application start to its completion, while the dynamic energy consumption is estimated by summing the energy consumed during computational and communication events. This method provides insights into the performance and energy characteristics of KPN applications under different mappings, offering lightweight simulation for faster Design Space Exploration (DSE).

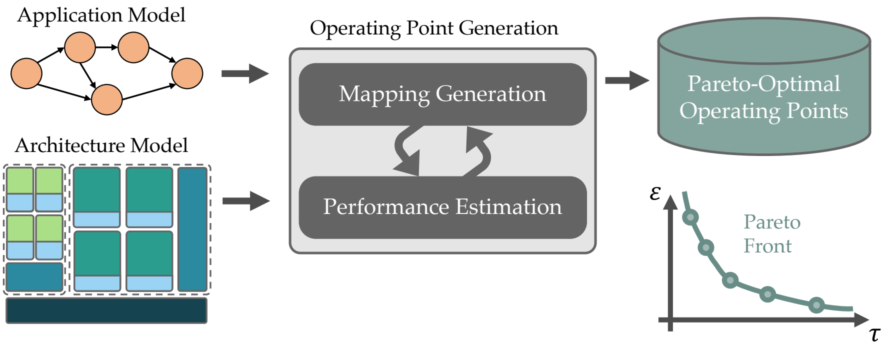
To perform mapping exploration and trace-based simulation, this thesis employs the MOCASIN framework, which implements the simulation techniques described above. Further details about the MOCASIN framework can be found in Section 2.5.

2.4 HYBRID APPLICATION MAPPING

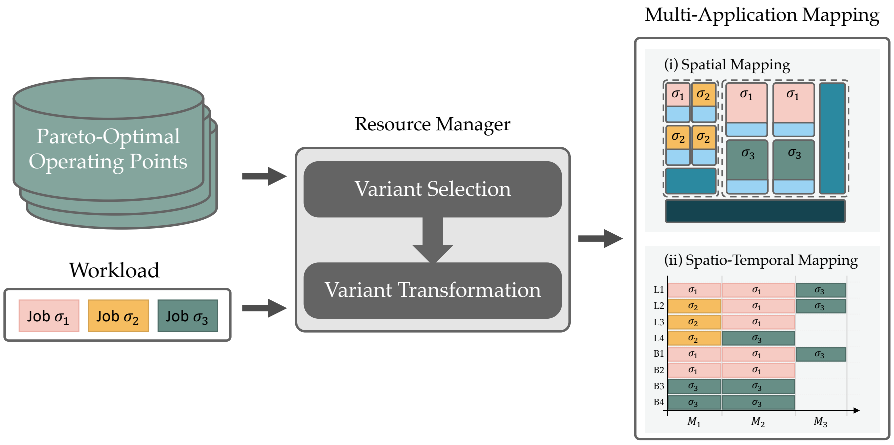
As discussed in Section 2.3.2, design-time mapping approaches produce high-quality mappings but cannot adapt to dynamic workloads. In contrast, runtime mapping approaches can easily adapt to dynamic workloads but cannot afford extensive computational time to find good mappings, potentially resulting in suboptimal solutions.

To overcome the limitations of both design-time and runtime approaches, researchers have proposed *Hybrid Application Mapping* (HAM), which combines the strengths of these two mapping classes while mitigating their weaknesses. In HAM, the mapping generation process is split into two stages: a design-time stage and a runtime stage. At design time, HAM performs sophisticated analysis to generate mapping candidates, each varying in resource usage and performance-energy characteristics. Then, at runtime, the Resource Manager (RM) analyzes the current workload and selects one of the pre-generated mapping candidates for each application, so that all applications satisfy the constraints and the overall system utilization is maximized.

Figure 2.3 illustrates a typical Hybrid Application Mapping (HAM) flow. At design time, each application is analyzed *in isolation* using DSE, leveraging the models of application and target architecture (Figure 2.3a). Similar to pure design-time mapping, DSE in HAM may use iterative metaheuristics, such as Evolutionary Algorithms (EAs), or domain-specific heuristics. As a result, DSE generates a set of *Pareto-optimal operating points*, a (possibly partial) mapping annotated with *non-functional characteristics* (i.e., performance and energy metrics). Pareto optimality is defined in multi-dimensional objective space: each operating point is not dominated by any other, meaning that it is better than others in at least one objective, such as performance, energy consumption, and the number of processing elements used per type. The operating points may be partial mappings, given as constraints [204], or complete mappings that are later adapted at runtime.



(a) Design-time generation of operating points



(b) Runtime resource management stage. The decision model is depicted in two variants: (i) Spatial multi-application mapping, and (ii) Spatio-temporal multi-application mapping

Figure 2.3: Overview of Hybrid Application Mapping, showing the two key stages: (a) Design-time exploration and (b) Runtime resource management.

The identified Pareto-optimal operating points are provided to the Resource Manager (RM), which manages applications at runtime, as depicted in Figure 2.3b. For each executing application, the RM selects the most appropriate operating point based on the current system state and workload. The selected variant is then adapted to the current system workload, either by finalizing the mapping from a partial one or by transforming a complete mapping to utilize the free resources in the platform.

To improve predictability — so that applications behave in the same way as they were analyzed, during the design-time stage — the applications at runtime are mapped with *spatial isolation* [68]. This means that threads from different applications do not compete for the same processor cores (although threads of the same application may share cores, and this behavior is already accounted for in the operating point's non-functional characteristics). To further improve predictability, some approaches also take into account composability at the interconnection level [204].

The runtime decision-making leverages scalable, lightweight heuristics, benefiting from the precomputed options provided by the design-time analysis. Traditional runtime algorithms employed in HAM generates spatial mappings and therefore select a single operating point for each application (as depicted under (i) in Figure 2.3b). In this thesis, Chapter 4 proposes novel algorithms that generate spatio-temporal mappings (described in Section 2.1.1, and depicted under (ii) in Figure 2.3b). In such approaches, the RM may select different operating points for different mapping segments, allowing for better adaptivity and thereby enhancing system performance and energy efficiency.

In the following, Section 2.4.1 introduces a formal model of operating points and Pareto optimality. Sections 2.4.2 and 2.4.3 define the resource management problem for spatial and spatio-temporal multi-application mappings, respectively. Finally, Section 2.4.4 explores how HAM approaches address adaptivity challenges in modern Heterogeneous Multi-core Architectures (HMAs).

2.4.1 *Pareto-Optimal Operating Points*

Operating points are the primary data structures that link the design-time and runtime stages of Hybrid Application Mapping (HAM). They represent specific application configurations and their mappings onto the HMA, annotated with performance and energy characteristics.

The total number of possible operating points is extremely large, growing exponentially with the number of application components that need to be mapped. Fortunately, methods exist to efficiently reduce this number to a manageable set of high-quality operating points, enabling fast decision-making at runtime.

This section formalizes the concept of operating points and explores their optimization through Pareto optimality.

2.4.1.1 *Operating Point*

The concept of operating points is formalized as follows.

Definition 2.6. An *operating point* o for application $A\langle \mathcal{M}, \Gamma, \mathcal{V} \rangle$ on a platform $\mathcal{P}\langle \Psi, \mathcal{IC} \rangle$ is a tuple $o := o^A\langle \gamma, \mu, \eta \rangle$, where:

- γ is a specific configuration from the set of application configuration parameters in $A[\Gamma]$.
- $\mu : A[\mathcal{V}_\gamma] \to \mathcal{P}[\Psi]$ is a mapping of the application's active components under configuration γ onto the platform's processing elements (see Definition 2.5).
- $\eta := \eta\langle \bullet, \bullet, \ldots \rangle$ is a set of *non-functional characteristics* representing the performance and energy efficiency of the application under this operating point.

The set of all operating points for the application A is denoted $\mathcal{O}^A$.

Depending on the specific multi-application optimization problem, the non-functional characteristics η can be defined differently. For real-time applications, $\eta := \eta\langle \tau, \varepsilon \rangle$, where τ is the execution time and ε is the energy consumption. The information about execution time is crucial for checking real-time constraints.

For non-real-time applications, the characteristics may be represented with *instant* metrics, such as utility v or power consumption p, i.e., $\eta := \eta\langle v, p \rangle$.

For convenience, indirect access to nested attributes is permitted. For example, $o[\vec{\omega}] := o[\mu][\vec{\omega}]$ refers to the resource vector, or $o[\varepsilon] := o[\eta][\varepsilon]$ refers to the energy consumption.

2.4.1.2 *Pareto Optimality*

To evaluate how "good" a particular operating point is, it is necessary to have criteria for assessment, expressed as *objective functions*. In most cases, multiple objectives are optimized simultaneously, and these objectives often conflict with each other, resulting in a partial rather than a total ordering on the search space [49].

For operating points defined above, the following objectives are generally considered:

1. Minimization of execution time $o[\tau]$ or maximization of utility $o[v]$.
2. Minimization of energy consumption $o[\varepsilon]$ or minimization of power consumption $o[p]$.

3. Minimization of the number of resources used per type $o[\vec{\omega}]$.

With a single objective function, the optimal solution is usually unique. However, with multiple objective functions, the notion of "optimum" changes, leading to a large number of different solutions. These solutions vary in their values across the objective functions, but each solution is superior to any other by at least one objective, thus, representing a compromise or "trade-off". Each of these optimal trade-offs is termed *Pareto optimum*.

Objective functions can be represented as a vector function:

$$\vec{F}(o) = [f_1(o), f_2(o), \ldots, f_k(o)]^\top \tag{2.10}$$

Assuming minimization of the objective functions the concepts of Pareto dominance and Pareto optimality can be formalized as follows.

Definition 2.7 (Pareto Dominance [49]). A vector $\vec{u} = [u_1, \ldots, u_k]$ *dominates* another vector $\vec{v} = [v_1, \ldots, v_k]$ (denoted by $\vec{u} \preceq \vec{v}$) if and only if $\vec{u}$ is partially less then $\vec{v}$, i.e., $\forall i \in \{1, \ldots, k\}, u_i \leq v_i \wedge \exists i \in \{1, \ldots, k\} : u_i < v_i$.

Definition 2.8 (Pareto Optimality [49]). An operating point $o \in \mathcal{O}$ is *Pareto Optimal* if and only if there is no $o' \in \mathcal{O}$ such that $\vec{F}(o') \preceq \vec{F}(o)$.

Definition 2.9 (Pareto-Optimal Set [49]). The *Pareto-optimal set* of operating points $\mathcal{O}^A_{\text{opt}}$ is defined as:

$$\mathcal{O}^A_{\text{opt}} := \left\{ o \in \mathcal{O}^A \mid \nexists o' \in \mathcal{O}^A \text{ s.t. } \vec{F}(o') \preceq \vec{F}(o) \right\} \tag{2.11}$$

The objective vectors of the Pareto optimal solutions are termed *nondominated* and form the Pareto front.

Definition 2.10 (Pareto Front [49]). For a given $\vec{F} : \mathcal{O}^A \to \mathbb{R}^k$, and Pareto optimal set $\mathcal{O}^A_{\text{opt}} \subset \mathcal{O}^A$, the *Pareto Front* (also called *Pareto frontier*) $\mathcal{F}^A_{\text{opt}}$ is defined as:

$$\mathcal{F}^A_{\text{opt}} := \left\{ \vec{u} = F(o) \mid o \in \mathcal{O}^A_{\text{opt}} \right\} \tag{2.12}$$

Reducing the set of operating points to only the Pareto optimal ones significantly decreases the number of points, retaining only a select set of high-quality operating points. This reduction aids the runtime heuristics employed by the RM, enabling rapid selection among these points to effectively balance performance, energy efficiency, and resource usage. If the number of operating points remains large, further reduction can be achieved through *distillation* techniques [149].

2.4.1.3 *Problem Statement*

The problem of finding the Pareto-optimal set of operating points can be formalized as follows:

Definition 2.11 (Multiobjective Optimization Problem)**.** Given an application $A\langle\mathcal{M},\Gamma,\mathcal{V}\rangle$, a target platform model $\mathcal{P}\langle\Psi,\mathcal{IC}\rangle$, and a set of objective functions $\vec{F}(o) = [f_1(o), f_2(o), \dots, f_k(o)]^\top$, find the Pareto-optimal set of operating points $\mathcal{O}^A_{\text{opt}} \subset \mathcal{O}^A$.

While this thesis does not focus on DSE in general, it describes the generation of operating points for task graphs with phased structures in Chapter 6. Additionally, Chapter 7 proposes a runtime algorithm for rapid estimation of operating points for general-purpose systems.

2.4.2 *Spatial Mapping Optimization*

At runtime, the Resource Manager (RM) manages the execution of multiple applications by handling incoming job requests and optimizing the allocation of resources. Each job request is associated with an application, with predefined operating points generated during the design stage. The RM generates a multi-application mapping that integrates these requests while ensuring energy efficiency and optimal resource utilization.

This section focuses on the optimization problem for *spatial* multi-application mapping, a conventional decision model in HAM methodologies (see Section 3.3). In this model, each job request is assigned a specific operating point, ensuring that the jobs do not overlap on resources.

While this thesis highlights the benefits of the spatio-temporal mapping, the spatial variant is more appropriate when job completion times are unpredictable. This scenario arises, for example, when the operating points are annotated with instantaneous metrics rather than expected (worst-case) execution times. This is particularly relevant for general-purpose systems where applications are typically non-real-time and their execution time and energy consumption can vary based on input. Given this unpredictability, the RM generates the spatial multi-application mappings.

The following subsections formalize the models of job requests and multi-application mappings and formulate the optimization problem.

2.4.2.1 *Job Request*

When a new job request arrives, the RM is activated to generate a new multi-application mapping that incorporates this request alongside other ongoing jobs. If a suitable multi-application mapping is successfully generated, the job is admitted into the system. The following definition applies to both spatial and spatio-temporal mapping optimizations.

Definition 2.12. A *job request* σ to execute an application on the HMA is defined as $\sigma := \sigma\langle A, t_{\text{arr}}, \theta, \rho\rangle$, where

- A is the application to execute,
- t_{arr} is the arrival time of the request,
- θ is the (relative) deadline (optional, used in real-time scenarios),
- $\rho \in [0, 1]$ is the current progress ratio (optional, used in real-time scenarios).

The set of all job requests is denoted Σ. Σ^{new} denotes the set of job requests that are not yet admitted by the RM, and Σ^{adm} is the set of previously admitted job requests.

2.4.2.2 *Spatial Multi-Application Mapping*

The RM generates a multi-application mapping to allocate resources across applications. Spatial multi-application mapping is defined as follows:

Definition 2.13 (Spatial Multi-Application Mapping)**.** A *spatial multi-application mapping* M assigns each job request $\sigma \in \Sigma$ an operating point $o \in \mathcal{O}^{\sigma[A]}_{\text{opt}} \cup \{\bot\}$ where $\bot$ represents no operating point. This assignment is denoted as $M[\sigma] := M[\sigma]\langle o \rangle$.

For convenience, some derived information is denoted as follows:

- *Resource vector* $M[\sigma][\vec{\omega}]$:

$$M[\sigma][\vec{\omega}] := M[\sigma][o][\vec{\omega}] \tag{2.13}$$

- *Non-functional characteristics* of the selected operating point $M[\sigma][\tau]$, $M[\sigma][\varepsilon]$ or $M[\sigma][v]$, $M[\sigma][p]$, each defined as:

$$M[\sigma][\bullet] := M[\sigma][o][\eta][\bullet] \tag{2.14}$$

2.4.2.3 *Problem Statement*

As discussed in Section 2.4.1.1, the non-functional characteristics of operating points can vary, encompassing metrics such as execution time and energy consumption or instantaneous metrics like utility and power consumption. To generalize the optimization problem, an *energy-utility cost* $\eta[\zeta]$ is introduced, defined as a function of other values within the non-functional characteristics η. The optimization problem is formalized as follows.

Definition 2.14 (Spatial Multi-Application Mapping Optimization Problem)**.** Given a set of job requests Σ and the Pareto Optimal set of operating points $\mathcal{O}^{\sigma[A]}$ for each $\sigma \in \Sigma$, find a spatial multi-application mapping M that minimizes the overall system's energy-utility cost:

$$\text{minimize } M[\zeta] = \sum_{\sigma \in \Sigma} M[\sigma][\zeta] \tag{2.15}$$

subject to the resource constraint, which ensures that the total resource demand does not exceed the available resources for each processing element type:

$$M[\vec{\omega}] = \sum_{\sigma \in \Sigma} M[\sigma][\vec{\omega}] \leq \mathcal{P}[\vec{\omega}] \tag{2.16}$$

2.4.2.4 *Equivalence to the Multiple-choice Multidimensional Knapsack Problem*

The optimization problem in Definition 2.14 is equivalent to the *Multiple-choice Multidimensional Knapsack Problem* (MMKP) [123]. In the MMKP, items have a *scalar* value and a *multi-dimensional* weight, and they are divided into several groups. The goal is to select a single item per group (*multiple-choice*) such that the overall value is maximized and the total weight does not exceed the maximum allowed weight at each dimension.

In our optimization problem, each application's operating points represent items in a group. The goal is to select one operating point per application (one item per group) such that the overall energy-utility cost is minimized. Here, the *weight* corresponds to the number of used processing elements of each type, and the *value* of an item is represented as a negative energy-utility cost.

Given that MMKP is NP-hard [151], finding an optimal solution is computationally challenging, particularly for a large number of applications and operating points. Section 3.3 discusses heuristics to rapidly approximating solutions to MMKP. In Chapter 7, a runtime system, HARP, employs a state-of-the-art Lagrangian relaxation to efficiently solve the optimization problem by relaxing constraints and then selecting feasible operating points [206, 207].

2.4.3 *Spatio-Temporal Mapping Optimization*

This section focuses on the optimization problem for *spatio-temporal* multi-application mapping, which extends the spatial mapping model by incorporating the time dimension. Algorithms using this model consider postponing and reconfiguring applications as part of the decision-making process. To achieve this, they account for the expected completion times of jobs to determine when resources will be released and available for other jobs, thereby extending the plan-ahead window. This approach improves the acceptance rate and the system's overall energy efficiency.

In this scenario, the non-functional characteristics η include the execution time τ and energy consumption ε. The workload is assumed to be real-time, meaning jobs have a deadline $\sigma[\theta]$ that must be met. Additionally, the RM tracks the current progress ratio $\sigma[\rho]$ for each job. The primary goal is to minimize energy consumption while ensuring that all admitted jobs meet their deadlines.

This section builds upon the workload model introduced in Section 2.4.2.1, presenting the model for spatio-temporal multi-application mapping and the corresponding optimization problem.

2.4.3.1 *Spatio-Temporal Multi-Application Mapping*

Spatio-temporal multi-application mapping is represented as a sequence of spatial mappings, each paired with a duration to form a *mapping segment*. The model is defined as follows.

Definition 2.15 (Spatio-Temporal Multi-Application Mapping)**.** A *Spatio-Temporal Multi-Application Mapping* K is a sequence of pairs:

$$K := \big(\langle M_1, \delta_1\rangle, \langle M_2, \delta_2\rangle, \ldots, \langle M_{|K|}, \delta_{|K|}\rangle\big) \tag{2.17}$$

where each pair $\langle M_i, \delta_i\rangle$ represents a *mapping segment*, with M_i being a spatial multi-application mapping (Definition 2.13) and δ_i its duration. The notation $K[M_i]$ and $K[\delta_i]$ denotes access to these fields.

The cumulative duration of the first n segments, denoted $K[\Delta_n]$, is defined as:

$$K[\Delta_n] := \sum_{i=1}^{n} K[\delta_i] \tag{2.18}$$

The overall energy consumption of the spatio-temporal multi-application mapping K, denoted $K[\varepsilon]$, is defined as:

$$K[\varepsilon] := \sum_{\langle M,\delta\rangle \in K} \sum_{\sigma\in\Sigma} M[\sigma][\varepsilon] \cdot \rho(M[\sigma], \delta) \tag{2.19}$$

where $\rho(M[\sigma], \delta)$ is the progress ratio of job σ during the mapping segment $\langle M, \delta\rangle$, defined as:

$$\rho(o, \delta) = \begin{cases} \frac{\delta}{o[\tau]} & \text{for} \quad o \neq \perp \\ 0 & \text{for} \quad o = \perp \end{cases} \tag{2.20}$$

2.4.3.2 *Problem Statement*

The goal of the RM is to minimize system energy consumption while ensuring that all admitted jobs meet their deadlines. The optimization problem is formulated as follows.

Definition 2.16 (Spatio-Tempotal Multi-Application Mapping Optimization Problem)**.** The *spatio-temporal multi-application mapping optimization problem* is defined as:

$$\text{minimize } K[\varepsilon] \tag{2.21}$$

where $K[\varepsilon]$ is defined in Equation (2.19).

The solution must satisfy the following constraints:

1. *Resource constraint*: The total resource demand of each mapping segment must not exceed the available resources:

$$M[\vec{\omega}] := \sum_{\sigma \in \Sigma} M[\sigma][\vec{\omega}] \leq \mathcal{P}[\vec{\omega}], \quad \forall \langle M, \delta \rangle \in K \tag{2.22}$$

2. *Deadline constraint*: Each job must finish before its deadline:

$$t_{\text{act}} + K[\Delta_{n^{\sigma}_{\text{last}}}] \leq \sigma[t_{\text{arr}}] + \sigma[\theta], \quad \forall \sigma \in \Sigma \tag{2.23}$$

   where t_{act} is the activation time of the RM and n^{σ}_{last} is the index of the last segment in which job σ is active:

$$n^{\sigma}_{\text{last}} = \max \{ i \mid K[M_i][\sigma] \neq \bot,\ 1 \leq i \leq |K| \} \tag{2.24}$$

3. *Completion constraint*: Each job must complete its execution:

$$\sigma[\rho] + \sum_{\langle M, \delta \rangle \in K} \rho(M[\sigma], \delta) = 1, \quad \forall \sigma \in \Sigma \tag{2.25}$$

If a feasible solution exists, the RM admits the job and activates the new multi-application mapping; otherwise, the job request is rejected.

Note that the formulation excludes runtime overhead associated with preemption and switching operating points. While some platforms exhibit negligible runtime overhead [68], this is not true for all platforms. Here, a simpler formulation is adopted, excluding overhead. Nevertheless, overhead can be easily incorporated into the algorithms proposed in Chapter 4.

2.4.4 *Addressing Adaptivity Challenges*

As discussed in Section 1.2, Heterogeneous Multi-core Architectures (HMAs) present adaptivity challenges at both the resource management and application levels. HAM approaches aim at addressing these challenges by integrating extensive offline optimization with rapid online adaptation.

At the resource management level, the system must dynamically allocate tasks to processor cores, considering the current workload and system conditions. The RM needs to make informed decisions about assigning tasks to appropriate core types to optimize performance and energy efficiency. HAM effectively addresses these challenges by:

- *Dynamic Resource Allocation*: By deferring the final mapping decisions to runtime, HAM allows the RM to adaptively *partition* cores among applications based on the current workload.

- *Incorporating Application Characteristics*: Sophisticated DSE generates optimal mappings with performance and energy annotations. These annotations allow the RM to consider specific characteristics of the entire application rather than individual threads.

- *Optimizing for Heterogeneity*: The precomputed operating points capture the performance and energy trade-offs across core types, enabling the RM to optimize energy efficiency of HMAs.

While addressing adaptivity at the resource management level, HAM typically focuses on thread-to-core pinning and does not address adapting the application internally. Applications are often not aware of the processor resources allocated to them, which limits their ability to fully utilize processor resources.

Enabling the application adaptivity to the allocated resources requires coordination between the RM and applications, with the RM informing applications about resource allocations and receiving back application behavior data. This two-way coordination between RM and applications is essential to enable both system and application-level optimizations.

This thesis bridges this gap by extending HAM with such two-way coordination mechanism. Chapter 6 presents an Adaptive Process Network (APN) model, in which the application topology can be dynamically changed based on the RM decision communicated. Chapter 7 presents HARP and `libharp`, a RM and application library that facilitate this two-way communication between the RM and various types of applications, not limited to KPNs. Additionally, HARP addresses a core limitation of HAM — the need for prior analysis at design time to obtain operating points — by introducing online monitoring and approximating Pareto-optimal points at runtime, thereby enabling efficient resource management even without prior design-time analysis.

2.5 MOCASIN FRAMEWORK

This thesis uses Mocasin, an open-source[3] rapid prototyping framework designed to support research on mapping applications to Multi-Processor Systems-on-Chip (MPSoC) [130]. Developed as a collaborative effort among multiple researchers at the Chair for Compiler Construction at TU Dresden, Mocasin is tailored for researchers and developers working on MoC-based analysis and simulation. While the framework itself is a joint effort, several algorithms and approaches presented in this thesis were both developed and tested in Mocasin, reflecting the author's contributions to the tool.

2.5.1 *Overview of Mocasin*

Mocasin supports multiple dataflow models of computation, particularly KPN, along with other models like SDF or task graphs, which can be viewed as specializations of KPN [107]. Figure 2.4 illustrates

---

[3] `https://github.com/tud-ccc/mocasin`

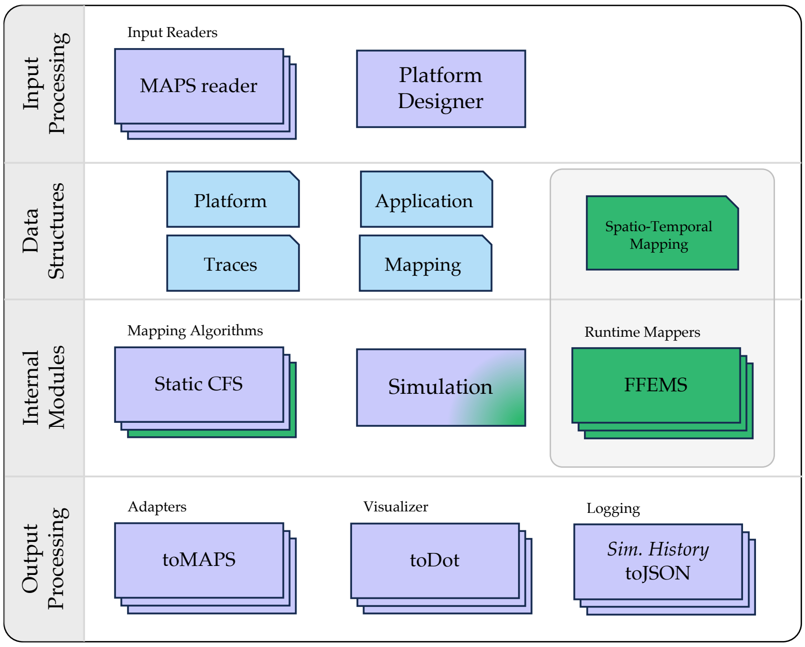


Figure 2.4: Overview of the Mocasin framework architecture (adapted from [130]). Components highlighted in green indicate contributions made as part of this thesis.

the modular architecture of the Mocasin framework, where each component stands independently and interacts with other modules as needed, enabling users to create specific tools, such as mapping algorithms or trace-based simulation.

The framework provides internal data structures to represent dataflow applications, platforms, mappings, and additional information about the runtime behavior, such as pre-recorded application traces. These structures are utilized by other components, including mapping algorithms and trace-based simulation module, as well as several convenience tools (e.g., for visualization).

A core component of Mocasin is its discrete-event simulation, which estimates application performance and energy consumption on the given platform based on application traces and mapping [129] (also discussed in Section 2.3.3). The simulation module considers communication events and dependencies between processes, providing a detailed estimation of application performance and energy efficiency. The execution of all processes in the system is controlled by a set of schedulers, such as the FCFS and Round Robin schedulers. The results are produced in the form of Gantt charts and simulation statistics.

Mocasin also features a modular mapper structure with a common interface for rapid prototyping of mapping algorithms and includes several pre-implemented algorithms based on heuristics and meta-heuristics. The heuristics utilize domain knowledge and internal data structures to derive mappings efficiently. For instance, Mocasin provides a static fair mapper that follows the basic design principle of the

Linux Completely Fair Scheduler (CFS) scheduler [142]. Metaheuristic methods explore the design space of possible mappings by evaluating multiple candidates (invoking the simulation component) and refining them through iterative searches. The implemented metaheuristics include genetic algorithms (using the DEAP framework [64]), tabu-search [121], and simulated annealing [141].

2.5.2 *Contributions to Mocasin*

During the course of this thesis, MOCASIN framework served as a platform for implementing and evaluating novel mapping algorithms and resource management algorithms. Key contributions include:

1. *Energy Estimation in Simulation*: The simulation module was extended to estimate energy consumption by incorporating core power characteristics (static and dynamic power consumption), enabling it to calculate the dynamic energy usage, as described in Section 2.3.3.
2. *Domain-Specific Mapping Algorithm*: A fast, energy-efficient mapping algorithm was added for task graphs with a phased structure, taking into account the performance-energy characteristics of the cores. This algorithm generates Pareto-optimal mappings by considering different processor core subsets, as discussed in Chapter 5.
3. *Runtime Resource Management*: A new module was developed in MOCASIN for prototyping runtime resource management algorithms within Hybrid Application Mapping (HAM) approach. This contribution includes the introduction of a *spatio-temporal mapping* data structure, as outlined in Section 2.1.1 and defined in Definition 2.15. This data structure is utilized in newly developed runtime resource management algorithms described in Chapter 4. Additionally, a state-of-the-art algorithm based on Lagrangian Relaxation [206, 207], which generates spatial mappings, was also implemented in MOCASIN. These management algorithms were subsequently integrated into the simulation module and used for evaluations presented in Chapter 5.

Through these contributions, MOCASIN facilitates rapid prototyping, testing, and validation of these novel runtime resource management approaches, advancing research on adaptive resource management on HMAs.

2.6 SYNOPSIS

This chapter established the foundational concepts and methodologies underpinning the research presented in this thesis. Key topics introduced include: (1) an overview of the terminology for mapping, scheduling, and spatio-temporal mapping in Section 2.1.1, (2) a survey of the performance and energy estimation methods in Section 2.1.2, (3) an explanation of static and dynamic components of power consumption in Section 2.1.3, (4) the background knowledge on dataflow models of computation in Section 2.3.1, along with their mapping approaches in Section 2.3.2, (5) the hybrid application mapping methodology in Section 2.4, and (6) an overview of the Mocasin framework in Section 2.5. Throughout the chapter, formal models were introduced to systematically represent the optimization problems for mapping applications onto heterogeneous platforms. Additionally, the Mocasin framework was presented, a critical tool used in this research for modeling, simulating, and analyzing application mappings.

Before delving into details of the solutions proposed, the next chapter presents an overview of solutions to similar problems.

3

RELATED WORK

Building upon the foundational concepts introduced in the previous chapter, this chapter surveys existing methodologies and research efforts related to application mapping and application adaptivity. The chapter begins by examining design-time application mapping approaches (Section 3.1), which focus on optimizing task placement prior to runtime. It then explores runtime mapping methodologies (Section 3.2), which make mapping decisions during application execution, with a focus on efforts targeting both embedded systems and general-purpose OS schedulers. Hybrid application mapping, which leverages the strengths of both design-time and runtime methods, is discussed in Section 3.3. Finally, existing methods for application adaptivity are reviewed in Section 3.4.

3.1 DESIGN-TIME APPLICATION MAPPING

For embedded software applications, often represented as dataflow applications, researchers have employed efficient mapping methodologies for decades. Broadly, these methodologies can be classified based on workload scenarios. For static and dynamic workload scenarios, the mapping methodologies optimize at design-time and runtime, respectively, classifying them as design-time and runtime methodologies [175].

Design-time mapping methodologies are particularly well-suited for static workload scenarios where both the set of applications and the hardware platform remain constant. In such environments, applications exhibit predictable computation and communication behaviors. This predictability allows design-time approaches to leverage a comprehensive, global view of the system, facilitating more informed and optimized resource allocation decisions. These methodologies can utilize detailed information, such as static code analysis, execution traces, profiling data, and hardware models, to determine optimal resource allocation.

Various strategies have been employed to find efficient mappings. Well-established search algorithms are often used to find *optimal* or *near-optimal* placement of tasks on platform cores. For example, Simulated Annealing (SA) have been utilized in [116, 141], Evolutionary Algorithm (EA) in [4, 47, 94, 152, 209], and Tabu Search in [121]. Additionally, some approaches have formulated the mapping problem as an Integer Linear Programming (ILP) [88, 157] or employed

Satisfiability Modulo Theories [120], solving them with dedicated solvers.

While these search-based methods can provide efficient mapping solutions, they often come with high computational costs, particularly for large-scale problems such as applications with a large number of tasks. To address this challenge, various strategies have been developed to reduce the computational overhead by pruning the search space. Techniques such as constraint programming have been used to effectively limit the search space [24, 158]. Additionally, analytical performance models can be leveraged in the early stages of iterative algorithms to reduce the overhead associated with simulating multiple mapping candidates [193]. Simpler heuristics based on domain-specific knowledge are also employed to generate mappings more efficiently [32, 38, 40, 43, 174, 187]. While these approaches provide mapping solutions faster than exhaustive search methods, they may miss higher-quality solutions due to search space pruning.

Despite their advantages, design-time mapping approaches have limitations in dynamic environments, as they cannot adapt to unpredictable workload changes. Since the mappings are determined statically, they lack the flexibility to adjust to runtime variations, potentially leading to suboptimal performance in dynamic scenarios.

3.2 RUNTIME APPLICATION MAPPING

In contrast to design-time mapping, runtime application mapping methodologies must account for the time taken to map each task, as this time contributes to the overall application execution. Additionally, tasks are typically mapped one by one as they arrive, unlike in design-time mapping, where all tasks are mapped simultaneously with a global view of the system. Consequently, runtime mapping approaches often rely on fast, greedy heuristic algorithms to efficiently map tasks, aiming to optimize metrics such as execution time, energy consumption, communication latency, and resource utilization.

Schedulers used in general-purpose operating systems can also be viewed as runtime mappers. This section first reviews runtime mapping strategies in embedded systems, which often focus on dataflow applications and specialized hardware architectures, and then discusses developments in general-purpose OS schedulers.

3.2.1 *Runtime Mapping of Embedded Software*

In the embedded domain, runtime mapping approaches often target dataflow applications, such as task graphs, Synchronous Dataflow (SDF), or Kahn Process Networks (KPNs). These applications exhibit characteristic communication and computation patterns that can be leveraged for efficient mapping. While the mapping occurs at runtime, some approaches also utilize pre-obtained execution traces to improve predictability and ensure timing guarantees.

For homogeneous MPSoC platforms with Network-on-Chip (NoC) interconnections, where all Processing Elements (PEs) are identical, runtime mapping methods focus on both computation and communication aspects. For example, Singh et al. [176] capture trace information of individual applications at design-time and merge execution intervals of multiple applications at runtime. Other approaches focus on communication optimization, such as ensuring timing requirements are met when time-multiplexing messages at NoC routers [135], or minimizing energy consumption due to communication overhead [48, 128].

In heterogeneous MPSoC NoC-based architectures, where PEs have different performance characteristics, mapping approaches account for this heterogeneity to optimize resource utilization. Strategies include self-adaptability that adjusts scheduling parameters based on resource utilization [83], multi-step heuristics that generate initial solutions and then refine them through re-mapping [167], and methods to reduce communication overhead by minimizing network congestion [36] or mapping adjacent tasks to the same or nearby PEs [177]. Additionally, there are runtime mapping strategies that target architectures featuring reconfigurable hardware components, such as multi-core reconfigurable processors [3], multi-core systems with shared reconfigurable fabric [45], NoC systems with Field-Programmable Gate Array (FPGA) fabric tiles [139] or Domain-Specific Reconfigurable Hardware (DSRH) tiles [181].

For further energy efficiency, resource management strategies incorporate Dynamic Voltage Frequency Scaling (DVFS). Brião et al. [30] propose a strategy that deactivates idle processors and applies DVFS to processors with slack time, conserving energy. Das et al. [53] use energy-awareness of single applications to improve both energy consumption and thermal dissipation.

With Arm's big.LITTLE platform — the first commercially available HMA — researchers began focusing on strategies specific to such platforms. Venkataramani et al. [200] present a runtime greedy approach that schedules multiple applications to increase overall throughput by exploiting the concavity of throughput in multi-threaded applications, though they do not explicitly optimize for energy efficiency.

Other runtime managers targeting the big.LITLLE platforms aim for energy efficient execution. Tzilis et al. [196] use profiling data to predict performance and energy consumption of a single-threaded application when co-scheduled with other applications, and decide on application placement and frequency settings. Similarly, Libutti et al. [115] utilize offline-collected data, such as CPU demands and memory sensitivity, for a job co-scheduling algorithm that optimizes resource usage while mitigating contention, thus improving performance and energy efficiency.

3.2.2 *Runtime Mapping within OS Schedulers*

Unlike embedded systems, general-purpose operating systems lack detailed application knowledge such as execution traces or profiling data, treating applications as black boxes. Since the dawn of multiprogramming, OS schedulers have evolved from handling the relatively simple task of time-sharing a single CPU to managing more complex resource allocations. Initially, schedulers focused on dividing CPU time among tasks to execute them concurrently on single-core systems. However, with the emergence of multi-core processors, schedulers had to handle *space-sharing* — allocating tasks across multiple cores. This transition introduced new challenges, such as contention for shared resources like caches and memory controllers. To address these issues, advanced thread placement strategies were developed to group threads that benefit from shared resources while separating those that would compete for them [27, 220].

Following the introduction of multi-core processors, researchers began exploring single-ISA heterogeneous processors, which integrate cores of different performance and power characteristics but share the same instruction set. These designs promised potential benefits in performance and energy efficiency but also introduced new challenges in OS scheduling [6, 106, 133].

Before heterogeneous processors became commercially available, studies emulated them using cycle-accurate simulation [16, 106], frequency scaling [161, 163], or modified core configurations [103]. Many of these works focused on optimizing thread performance by assigning CPU-intensive threads to faster cores [16, 103, 106, 163], while others proposed alternative strategies, such as assigning sequential applications and sequential phases of parallel applications to fast cores [161].

As heterogeneous processors entered the market, major operating systems like Linux and Windows began integrating new scheduling strategies, following the approach of assigning CPU-intensive tasks to fast cores. For Arm's big.LITTLE processors, Linux introduced the *Energy-Aware Scheduler* (EAS), which uses CPU energy consumption models to optimize task placement [145]. EAS tracks task CPU demand via Per-Entity Load Tracking (PELT), allocating tasks to minimize energy consumption while maintaining performance, preferring LITTLE cores for low-demand tasks. The PELT data is also used by the DVFS subsystem, making EAS tightly integrated with dynamic frequency scaling.

Intel's Alder Lake processors are equipped with the *Intel's Thread Director* (ITD), a hardware-based feature that monitors the runtime instruction mix of each thread and the state of each core at the nanosecond level [87]. ITD uses machine learning to classify threads and calculate performance and energy-efficiency scores for each core-class combination. This classification and the corresponding scores are provided as feedback to the OS at the microsecond level. Windows 11 already supports ITD [52]. The Linux community is working on incorporating this functionality through patches that are not yet merged into the kernel [46, 137]. Meanwhile, PMCSched is a recently published research scheduler extension for Linux that can integrate ITD data into task placement decisions [20, 162].

While these systems account for thread behavior on different core types to improve energy efficiency, they overlook the collective impact on overall application performance. Additionally, OS schedulers are inherently limited in their optimization capabilities: the applied heuristics must remain lightweight to ensure efficient execution, which restricts them to operating with limited information. Furthermore, they do not propagate resource allocation decisions to the application level, which prevents applications from dynamically adapting to the allocated resources. This lack of coordination between the OS scheduler and applications limits the potential for further optimization of resource utilization.

3.3 HYBRID APPLICATION MAPPING

To address the limitations of purely design-time and runtime mapping methods, a trend towards *Hybrid Application Mapping* (HAM) approaches has emerged [148, 175]. HAM strategies effectively combine the strengths of both methods. As discussed in Section 2.4, HAM leverages extensive analyses from Design Space Exploration (DSE) at design time while deferring final mapping decisions to runtime. This dual approach allows systems to adapt rapidly to changing workloads while benefiting from pre-computed analysis results, ensuring efficiency and flexibility in managing dynamic system loads. The use of HAM began in the 2000s with pioneers like Yang et al. [210] for real-time systems.

The goal of the design-time stage is to generate a set of (possibly incomplete) mapping options, referred to as *operating points*, each characterized by the required amount and types of resources and their expected execution properties. Typically, HAM employs Evolutionary Algorithms (EAs) during DSE to generate partial or complete Pareto-optimal mappings [10, 122, 204], though efficient heuristics can also approximate optimal configurations [126, 140, 174]. Some approaches also distinguish different application scenarios [153, 165, 168, 183]. Non-functional characteristics are identified through models, traces, static analysis [40, 77, 122], and direct measurements. The search space can also be reduced by exploiting system symmetries [68, 69].

At runtime, a Resource Manager (RM) uses these precomputed operating points to allocate resources among concurrently executing applications, aiming to maximize system energy efficiency. The RM's main task is to select the appropriate operating point for each executing application. Since this is an NP-hard problem [151], faster algorithms are preferred at runtime. Some methods use iterative algorithms to map applications incrementally; Singh et al. [174] use this approach, while Weischlgartner et al. [204, 205] enhance it with a repair heuristic.

Other methods select operating points jointly, often framing the optimization as a *Multiple-choice Multidimensional Knapsack Problem* (MMKP) [123], where the goal is to select a single item per group with multidimensional weights while maximizing overall value. Since MMKP is NP-hard [151], approximate solutions are commonly applied. For instance, Ykman-Couvreur et al. [212] propose a heuristic that represents resource demands of operating points as single values, then applies a greedy algorithm to solve the MMKP. This heuristic underlies solutions like those in [122, 126]. Shojaei et al. [171] use a compositional Pareto-algebraic heuristic, while Wildermann et al. [206, 207] apply Lagrangian relaxation methods.

Some HAM are also scenario-aware. For example, Spieck et al. [183, 184] propose a scenario-aware HAM methodology where DSE classifies input stimuli into scenarios with similar performance-energy characteristics. At runtime, the manager identifies scenarios and allocates resources accordingly, using the Lagrangian relaxation on MMKP.

However, these runtime approaches generate only *spatial* multi-application mappings, without considering application reconfiguration. To enhance system utilization, some methods consider postponement and reconfiguration of real-time applications, generating *spatio-temporal* mappings. Mukherjee et al. [136] explored spatio-temporal job scheduling in heterogeneous data centers, proposing an EA-based algorithm that finds near-optimal solutions, but is time-intensive. Another approach, cluster scheduling in heterogeneous cloud environments, applies Mixed-Integer Linear Programming (MILP) solvers [195], though this incurs significant overhead as the plan-ahead window size increases. In the embedded domain, some works generate spatio-temporal mappings, although these are often limited to single-threaded applications [138]. In contrast, our approaches proposed in Chapter 4 generate the spatio-temporal mappings for multi-threaded applications within milliseconds.

Spatio-temporal mapping is inapplicable to general-purpose OS schedulers, as it may delay application execution, leading to potential starvation. In desktop environments, scheduling must avoid starvation and ensure steady application progress. For such system, the proposed

HARP approach, described in Chapter 7, generates spatial mappings using a state-of-the-art Lagrangian relaxation heuristic [206, 207].

While HAM approaches adapt to dynamic workloads, they require sophisticated offline analyses to understand the application behavior across different mappings. These analyses are time-intensive, making HAM less practical for general-purpose desktop systems. AdaMD [15] addresses this limitation by using hardware performance counters to analyze application behavior on each core type individually at runtime, after the application has arrived. However, it relies on performance models to estimate whole-application performance, which may introduce inaccuracies. In contrast, HARP executes the application on various subsets of cores to obtain more reliable measurements and construct Pareto-optimal set of operating points.

3.4 APPLICATION ADAPTIVITY

To fully leverage modern processing capabilities, solely managing thread-to-core assignments is insufficient. Many applications remain unaware of current system load or available cores and do not internally adapt to the allocated resources, such as adjusting their parallelization degree.

According to Feitelson and Rudolph's classification [63], parallel jobs can be categorized based on *who* specifies the number of processors (user or system) and *when* this decision is made (at job submission or during execution). Jobs are *rigid* if the parallelization degree is specified by the user at submission time (e.g., fixed in the program) and *moldable* if specified by the system. If the parallelization degree can change during execution, jobs are considered *evolving* when the program internally adjusts its parallelization (e.g., through different phases) or *malleable* when the system can dynamically alter the parallelization degree. To better utilize modern processors, applications need to be malleable, adapting their parallelization degree in response to resource manager triggers.

In embedded systems, optimizing data parallelism in dataflow applications is often referred to as fission, partitioning, or replication [81]. While many approaches perform this optimization at design time [62, 70, 71, 105, 182, 188, 219], some address it at runtime. For example, Lee et al. [111] introduced a dynamic scheduling approach that replicates actors in SDF applications. Lund et al. [119] offload the computation of stateless actors to GPU units, though this approach targets RVC-CAL, a non-deterministic dataflow MoC.

Beyond replication, some approaches propose more generalized transformations for dataflow MoCs. For instance, Reconfigurable Dataflow (RDF) [65] extends SDF with transformation rules specifying the conditions under which the topology and actors of the graph may be reconfigured. Schor et al. [166] introduced Expandable Process

Network (EPN) as an extension of a more expressive KPN. Similar to RDF, EPN introduces refinement and contraction rules for stateful processes.

While both RDF and EPN address more general transformations, they do not explicitly consider different performance-energy characteristics of emerging Heterogeneous Multi-core Architectures (HMAs), such as by employing dynamic load distribution across heterogeneous cores. Evenly distributing workloads can lead to imbalances, as tasks executing on energy-efficient cores take longer to execute. In general-purpose environments, such optimizations have been proposed for OpenMP [160], but it does not address changes to the parallelization degree in response to resource availability.

In this thesis, an APN is presented in Chapter 6, introducing implicit parallelism into KPNs. This extension allows applications to change their parallelization degree in a malleable way and balance load distribution across heterogeneous cores. Furthermore, instead of addressing each application model individually, a runtime system, HARP, is proposed in Chapter 7 that provides a flexible framework for customizing application model support and offers a unified interface for controlling adaptivity across various applications. This framework allows applications to be aware of the allocated resources and perform internal optimizations such as adapting parallelization degree, load balancing, or employing techniques like algorithm switching, thereby fully utilizing the allocated processor cores.

4

EFFICIENT SPATIO-TEMPORAL MAPPING GENERATION

This chapter addresses the spatio-temporal mapping optimization problem introduced in Section 2.4.3.2, focusing on firm real-time applications executing on Heterogeneous Multi-core Architectures (HMAs). The optimization problem focuses on generating spatio-temporal mappings that enhance resource utilization and energy efficiency while meeting application deadlines.

Two strategies for spatio-temporal mappings are explored: fixed-point and flexible. Both approaches produce spatio-temporal mappings, but differ in the flexibility of their solutions.

The chapter introduces three algorithms. The first, *MMKP-MDF*, generates fixed-point spatio-temporal mappings by formulating the problem as a Multiple-choice Multidimensional Knapsack Problem (MMKP) with a Maximum Difference First (MDF) policy. The second, *Spatio-Temporal Evolutionary Mapping* (STEM), creates flexible spatio-temporal mappings using Memetic Algorithms (MAs). The third, *Fast Flexible Energy-Minimizing Scheduler* (FFEMS), also produces flexible spatio-temporal mappings but employs fast heuristics, making it suitable for runtime implementation.

The chapter begins with a motivational example in Section 4.1, illustrating the benefits of spatio-temporal mapping for improving system efficiency. Section 4.2 discusses strategies for generating spatio-temporal mappings. Section 4.3 introduces the MMKP-MDF algorithm and its evaluation. Section 4.4 presents the flexible mapping algorithms, STEM and FFEMS, along with their evaluations. The chapter ends with a summary in Section 4.5.

A Note on Publications and Contributions

Most of the content in this chapter, including the motivational example, algorithms, figures, and results, was previously published in Khasanov and Castrillon, "Energy-efficient Runtime Resource Management for Adaptable Multi-application Mapping," 2020 [97], and Khasanov, Dietrich, and Castrillon, "Flexible Spatio-Temporal Energy-Efficient Runtime Management," 2024 [98]. The main ideas and algorithms were conceptualized and developed by the author of this thesis, with one exception: while the idea and design of the STEM algorithm, including its local search heuristics, were contributions of the author, the implementation of the core genetic component of STEM was carried out by Marc Dietrich.

4.1 MOTIVATIONAL EXAMPLE

The following motivational example illustrates how spatio-temporal mapping strategies can improve the acceptance of job requests and reduce the energy consumption in a HMA.

Consider a Resource Manager (RM) managing a HMA with two energy-efficient cores (denoted as `L`) and two high-performance cores (denoted as `B`). The RM handles job requests arriving according to two scenarios, S_1 and S_2, as shown in Table 4.1. Each job request σ includes the application to execute A, its arrival time t_{arr}, and its absolute deadline θ_{abs}.

Table 4.1: Job request parameters for the motivational example, showing two execution scenarios S_1 and S_2. Each job request σ specifies the application A, arrival time t_{arr}, and absolute deadline θ_{abs}.

		Scenario S_1		Scenario S_2	
Job σ	App. A	t_{arr}	θ_{abs}	t_{arr}	θ_{abs}
σ_1	A_1	0	9	0	9
σ_2	A_2	1	5	1	4

Table 4.2 lists the operating points for the two applications, A_1 and A_2. The operating points specify the number of energy-efficient cores (`#L`) and high-performance cores (`#B`), execution time τ (in seconds), and energy consumption ε (in joules). The values of execution time and energy consumption are synthetic but reflect ratios similar to those observed in real applications (see Section 4.3.2 and Section 4.4.3).

For application A_1, the execution times and energy consumptions are provided as triples, representing the values at different progress ratios: initial (0 %), after approximately 19 %, and after approximately 62 % progress. These progress points correspond to specific moments in the execution where mapping is reevaluated in this motivational example.

At time $t = 0$, the RM receives the request σ_1 to execute application A_1. An energy-optimizing mapper decides to map it to two energy-efficient cores and one high-performance core (configuration `2L1B`), since this configuration meets the deadline at $t = 9$ while consuming the least energy (8.9 J, as underlined Table 4.2).

After 1 s, the job request σ_2 arrives while σ_1 has progressed to approximately 19 % completion. To meet its deadline ($\theta_{\text{abs}} = 5$ in scenario S_1), σ_2 must be executed using one of the following configurations: `2B`, `1L1B`, `1L2B`, `2L1B` or `2L2B`. If any of these mappings is chosen for σ_2, σ_1 must continue with a configuration using the remaining resources, such as `1L`, `2L`, `1L1B`, or `1B`.

Table 4.2: Operating points for applications A_1 and A_2 in the motivational example. Each operating point is characterized by the number of energy-efficient (#L) and high-performance cores (#B) used, and the corresponding execution time τ and energy consumption ε.

		A_1, pr. 0 % - 19 % - 62 %		A_2, pr. 0 %	
#L	#B	τ, s	ε, J	τ, s	ε, J
1	0	16.8 - 13.63 - 6.37	7.90 - 6.41 - 3.00	10.0	2.00
2	0	10.3 - 8.36 - 3.91	7.01 - 5.69 - 2.66	7.0	2.87
0	1	11.2 - 9.09 - 4.25	18.54 - 15.04 - 7.03	5.0	7.55
0	2	6.3 - 5.11 - 2.39	17.70 - 14.36 - 6.71	3.5	10.5
1	1	8.1 - 6.57 - 3.07	10.90 - 8.84 - 4.13	3.5	6.44
1	2	7.9 - 6.41 - 3.00	10.60 - 8.60 - 4.02	3.0	6.81
2	1	5.3 - 4.30 - 2.01	$\underline{8.90}$ - 7.22 - 3.38	3.0	5.73
2	2	4.7 - 3.81 - 1.78	11.00 - 8.92 - 4.17	2.0	6.58

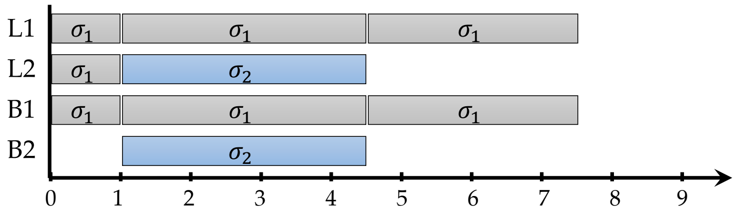
A mapper that only explores spatial mappings would choose configurations where both jobs meet their deadlines without changing operating points during execution; for example, mapping both σ_1 and σ_2 to 1L1B. By time $t = 4.5$, σ_2 finishes execution, and σ_1 progresses to approximately 62 % completion. If σ_1 continues to use configuration 1L1B until completion, as depicted in Figure 4.1a, the overall energy consumption would be 16.96 J.

Alternatively, if the RM decides to remap σ_1 at $t = 4.5$ to the more energy-efficient configuration 2L, as shown in Figure 4.1b, the overall energy consumption reduces to 15.49 J.

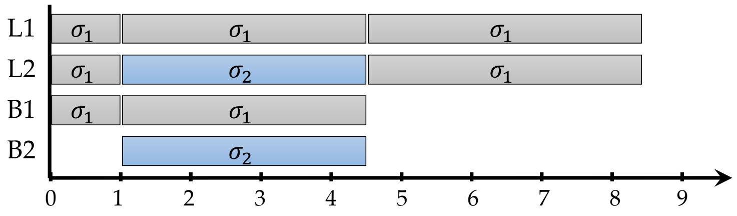
However, if at $t = 1$ the RM runs σ_2 on configuration 2L1B and temporary suspends σ_1, then after σ_2 completes, σ_1 can resume with configuration 2L1B. This approach leads to an even lower overall energy consumption of 14.63 J, as depicted in Figure 4.1c.

Now, consider the tighter scenario S_2 in Table 4.1. At $t = 1$, σ_2 can only choose configurations 1L2B, 2L1B, or 2L2B to meet its deadline ($\theta_{abs} = 4$), which leaves at most either one high-performance or one energy-efficient core available for σ_1. Since these configurations are insufficient for σ_1 to meet its deadline under a fixed mapping, a mapper that does not consider spatio-temporal strategies would fail to find a feasible solution. Consequently, σ_2 would be rejected. By employing spatio-temporal mapping strategies, the RM can generate the same solution as in Figure 4.1c and meet the constraints.

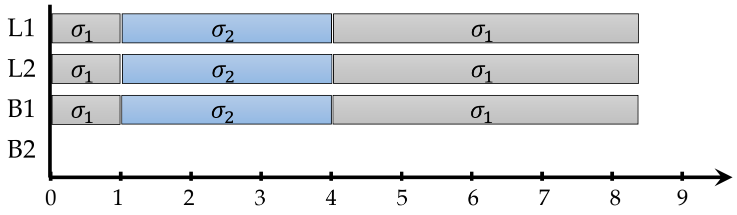
This example demonstrates the advantages of spatio-temporal mapping strategies over spatial mappings. By extending the plan-ahead window analysis and allowing applications to change operating points and be postponed, the RM can improve the acceptance of job requests



(a) Spatial mapper activated only at application start (resulting energy: 16.96 J).



(b) Spatial mapper activated at both application start and completion (resulting energy: 15.49 J).



(c) Spatio-temporal mapper leveraging reconfiguration and postponement strategies (resulting energy: 14.63 J).

Figure 4.1: Gantt charts comparing three resource management strategies in the motivational example.

and optimize for energy efficiency, especially under tight timing constraints.

4.2 SPATIO-TEMPORAL MAPPING STRATEGIES

As observed in the motivational example in Section 4.1, the choice of mapping decision model — spatial or spatio-temporal — significantly impacts system utilization and energy efficiency. This section delves deeper into strategies for generating multi-application mappings, with a particular focus on spatio-temporal approaches.

A spatial multi-application mapping strategy requires the RM to select a single operating point for each application, ensuring that all applications meet their deadlines without sharing processing elements. Since spatial mapping does not account for future events, such as the release of resources when applications complete, its optimization is limited to a *local* scope. When some applications finish, the RM can be reactivated to generate a new spatial mapping, potentially

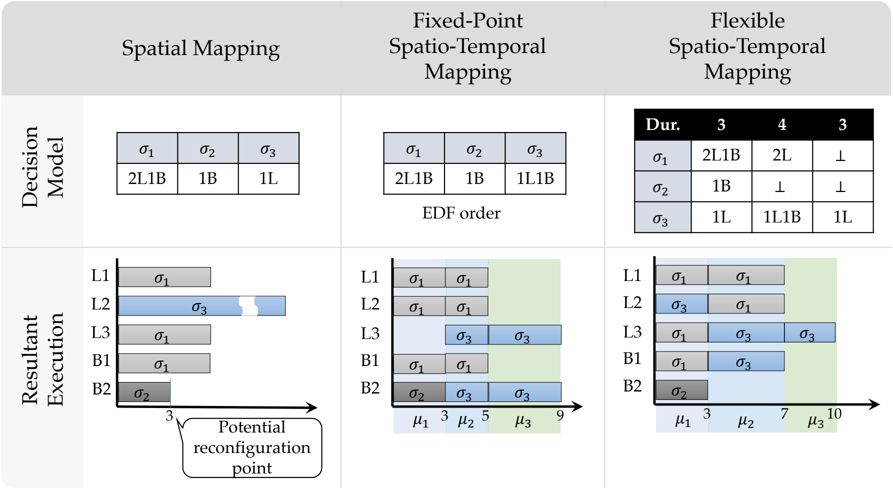

Spatial Mapping
Fixed-Point Spatio-Temporal Mapping
Flexible Spatio-Temporal Mapping
Decision Model
Resultant Execution
σ_1 σ_2 σ_3
2L1B 1B 1L
σ_1 σ_2 σ_3
2L1B 1B 1L1B
EDF order
Dur. 3 4 3
σ_1 2L1B 2L ⊥
σ_2 1B ⊥ ⊥
σ_3 1L 1L1B 1L
L1 L2 L3 B1 B2
3
Potential reconfiguration point
μ_1 3 μ_2 5 μ_3 9
μ_1 3 μ_2 7 μ_3 10

Figure 4.2: Overview of multi-application mapping strategies. Spatial mapping selects a single operating point per application, ensuring that all applications run simultaneously without sharing processing elements. Fixed-point spatio-temporal mapping assigns a single operating point per application but schedules applications over time in a specified order (e.g., EDF policy), and, allowing for postponement of execution. Flexible spatio-temporal mapping allows applications to change operating points across different mapping segments, with the RM specifying segment durations.

improving energy efficiency, as demonstrated in Figure 4.1b. However, this sequence of locally optimal decisions might result in suboptimal performance over the entire runtime of the system.

Spatio-temporal mapping strategies address this limitation by incorporating the temporal component, enabling improved system utilization and energy efficiency. These strategies consider expected changes in the workload, such as applications completing, to generate more efficient execution plans. However, incorporating the temporal dimension leads to a combinatorial increase in the complexity of the search space.

To address this complexity, two variants of spatio-temporal mapping strategies are illustrated in Figure 4.2:

- *Fixed-Point Spatio-Temporal Mapping Strategy*: This strategy assigns a single operating point per application, similar to the spatial mapping approach. However, it allows sharing processing elements in this decision, which is resolved by applying a specific execution order (e.g., using an Earliest Deadline First (EDF) policy), thereby finalizing a spatio-temporal multi-application mapping. This approach benefits from a reduced search space, allowing for faster heuristics. Compared to pure spatial mapping, it can postpone application execution to achieve better energy efficiency and resource utilization.

- *Flexible Spatio-Temporal Mapping Strategy*: This strategy fully leverage the spatio-temporal mapping model by allowing applications to execute under different operating points in different mapping segments. The resource manager designates mapping segments and their durations, providing greater adaptability. While this approach can enhance system performance and energy efficiency, it introduces significantly greater number of variables into the decision-making process, making it more challenging to find efficient configurations.

Algorithms employing these two spatio-temporal strategies are presented next. Section 4.3 introduces the MMKP-MDF algorithm, which employs the fixed-point spatio-temporal strategy. Section 4.4 describes algorithms that generate more flexible decision models: one based on a Memetic Algorithm (MA) that produces high-quality solutions but is impractical due to long search times, and another utilizing more lightweight heuristics for faster decision-making.

4.3 FIXED-POINT SPATIO-TEMPORAL MAPPING

The first approach producing spatio-temporal mappings builds upon the work of Niknafs et al. [138], who proposed a rapid heuristic based on a Multiple-choice Multidimensional Knapsack Problem (MMKP) formulation. Their heuristic selects a single operating point for each application, and then the jobs are scheduled according to the Earliest Deadline First (EDF) policy. As such, their approach can be interpreted as a fixed-point strategy.

However, their algorithm is limited to single-threaded tasks. Extending it to multi-threaded execution poses challenges, as multiple threads of single applications have to be executed on several processing elements at the same time. This work generalizes the approach to multi-threaded applications, addressing these complexities.

Section 4.3.1 describes a fast algorithm designed for firm real-time multi-threaded applications. The algorithm analyzes all applications within scope until the last application completes and generates spatio-temporal mappings optimized for overall energy consumption. In Section 4.3.2, this approach is evaluated, demonstrating how the enlarged scope of the analysis improves system performance.

4.3.1 *MMKP-based Algorithm*

The *MMKP-MDF* algorithm is a heuristic based on the Multiple-choice Multidimensional Knapsack Problem (MMKP) combined with a Maximum Difference First (MDF) selection policy, adapted for fixed-point spatio-temporal mapping of multi-threaded applications.

Processing element types (Ω_i) are modeled as knapsacks, where the *capacity* of each knapsack represents the available processing time per resource type over the analysis time horizon. Each operating point $o \in \mathcal{O}_{\text{opt}}^{\sigma[A]}$ is treated as an *item* with associated *weight* and *value*. The weight is defined as the product of the execution time $o[\tau]$ and the number of processing elements of each type $o[\vec{\omega}]$, representing the total resource-time consumption. The total available capacity $\vec{J}$ represents the maximum available resource-time product for each resource type.

Each job forms a group of items (its possible operating points), and exactly one item must be chosen from each group. By negating the energy consumption $o[\varepsilon]$ of each item to obtain a *value*, the optimization goal becomes to maximize the total value (i.e., minimize the total energy consumption).

This problem can be formulated as the Multiple-Choice Multidimensional Knapsack Problem (MMKP) [123]. Previously, in Section 2.4.2.4, we introduced an MMKP-based formulation for spatial mapping. In the spatial problem, the weights correspond to the resource used per type. In contrast, in this algorithm, the weights are defined as the required resources multiplied by the execution time.

Algorithm 4.1 describes our heuristic to solve this problem. At each activation of the RM, the capacities $\vec{J}$ are initialized with the total available resource time products per resource type, calculated as the available resource $\mathcal{P}[\vec{\omega}]$ multiplied by the time horizon, which is the largest job deadline (Line 1). The dictionary *opc*, which stores the selected operating points for each job, is initialized with the value $\perp$ (Line 3).

The algorithm iterates over unmapped jobs (Line 5), using NEXTJOB-MDF function to select the next job to map (Line 6). This function performs the following steps:

1. For each unmapped job, it filters its operating points by checking whether they can meet job's deadlines and fit within the current remaining capacities $\vec{J}$.
2. It computes the difference in energy consumption between the most energy-efficient feasible configuration and the second-best configuration for each job.
3. It selects the job σ^* with the maximum difference, following the MDF policy
4. It returns the selected job σ^* along with its list of feasible configurations CL.

The MDF policy prioritizes the job that would experience the highest degradation in energy consumption if the most energy-efficient configuration is not chosen in this iteration.

Algorithm 4.1 Main Procedure of the MMKP-MDF Algorithm

```
Input: Job requests $\Sigma$, platform $\mathcal{P}$, Pareto-optimal operating points
    $\mathcal{O}^{\sigma[A]}_{\text{opt}}$ for each $\sigma \in \Sigma$ (shortly, $\mathcal{O}^{\Sigma[A]}_{\text{opt}}$)
Output: Spatio-temporal mapping $K$
 1: $\vec{J} \leftarrow \mathcal{P}[\vec{\omega}] \cdot \max(\sigma[\theta] \mid \sigma \in \Sigma)$        ▷ Initialize the capacities
 2: for all $\sigma \in \Sigma_t$ do
 3:     $opc[\sigma] \leftarrow \bot$        ▷ Initialize selected operating points
 4: $\Sigma_{\text{rem}} \leftarrow \Sigma$
 5: while $\Sigma_{\text{rem}} \neq \emptyset$ do
 6:     $\sigma^*, CL \leftarrow$ NextJobMDF($\vec{J}, \left\{ \langle \sigma, \mathcal{O}^{\sigma[A]}_{\text{opt}} \rangle \mid \sigma \in \Sigma_{\text{rem}} \right\}$)
 7:     while $\sigma^* \in \Sigma_{\text{rem}}$ do
 8:         if $CL = \emptyset$ then return null
 9:         $o^* \leftarrow \text{argmin}_{o \in CL}\{o[\varepsilon]\}$
10:         $opc^* \leftarrow opc$; $opc^*[\sigma^*] \leftarrow o^*$
11:         $K^* \leftarrow$ ConstructSTM($\Sigma, opc^*, \mathcal{P}$)
12:         if $K^* \neq$ null then
13:             $opc \leftarrow opc^*$; $K \leftarrow K^*$; $\Sigma_{\text{rem}} \leftarrow \Sigma_{\text{rem}} \setminus \sigma^*$
14:             $\vec{J} \leftarrow \vec{J} - (1 - \sigma^*[\rho]) \cdot o^*[\tau] \cdot o^*[\vec{\omega}]$
15:         else
16:             $CL \leftarrow CL \setminus o^*$
17: return $K$
```

For the selected job σ^*, the algorithm iterates over its feasible configurations in order of increasing energy consumption (Lines 7–16). For each operating point o^*, it attempts to construct a spatio-temporal mapping including σ^* along with already mapped jobs, using the ConstructSTM function detailed in Algorithm 4.2 (Line 11). If a feasible mapping K^* is found, the algorithm updates opc and K, and adjusts the capacities $\vec{J}$ to account for the resource-time product used by σ^* (Lines 13–14). If no feasible spatio-temporal mapping with the current operating point, it removes it from the list CL and continues to the next operating point.

If all operating points are exhausted (i.e., CL becomes empty) and no feasible spatio-temporal mapping is found, the algorithm returns no solution (Line 8).

Algorithm 4.2 takes the selected operating points as input and generates a feasible spatio-temporal mapping on. In Line 2, the algorithm initializes the spatio-temporal mapping. It iterates over unmapped jobs (Lines 3–26) in non-decreasing order of their deadlines, i.e., Earliest Deadline First (EDF) policy (Line 4).

Within the loop, the algorithm attempts to map the job onto the existing mapping segments (Lines 7–20). For each segment, it checks whether there are sufficient resources to accommodate the job's operating point (Line 8). If resources are sufficient, it determines whether the job can execute for the entire duration of the segment (Lines 10–11) or

Algorithm 4.2 ConstructSTM: Construct spatio-temporal mapping with EDF policy

```
Input: Job requests $\Sigma$, selected operating points $opc$, platform $\mathcal{P}$
Output: Spatio-temparoal mapping $K$
 1: $\widetilde{\Sigma} \leftarrow \{\sigma \in \Sigma \mid opc[\sigma] \neq \perp\}$
 2: Initialize an empty spatio-temporal mapping $K$
 3: while $\widetilde{\Sigma} \neq \emptyset$ do
 4:     $\sigma^* \leftarrow \text{argmin}_{\sigma \in \widetilde{\Sigma}}\{\sigma[\theta]\}$      ▷ Select job with earliest deadline
 5:     $o^* \leftarrow opc[\sigma^*]$
 6:     $t_{\text{rem}} \leftarrow o^*[\tau] \cdot (1 - \sigma^*[\rho])$      ▷ Remaining execution time
 7:     for $i \in \{1, \dots, |K|\}$ do
 8:         if $o^*[\vec{\omega}] + K[M_i][\vec{\omega} \leq \mathcal{P}[\vec{\omega}]$ then
 9:             if $t_{\text{rem}} \geq K[\delta_i]$ then
10:                 $K[M_i][\sigma^*] \leftarrow o^*$
11:                 $t_{\text{rem}} \leftarrow t_{\text{rem}} - K[\delta_i]$
12:             else
13:                 SplitSegment($K$, $i$, $t_{\text{rem}}$)
14:                 $K[M_i][\sigma^*] \leftarrow o^*$
15:                 $t_{\text{rem}} \leftarrow 0$
16:                 $t_{\text{fin}} \leftarrow K[\Delta_i]$
17:                 break
18:             if $t_{\text{rem}} = 0$ then
19:                 $t_{\text{fin}} \leftarrow K[\Delta_i]$,
20:                 break
21:     if $t_{\text{rem}} > 0$ then
22:         AppendSegment($K$, $t_{\text{rem}}$)
23:         $K[M_{|K|}][\sigma^*] \leftarrow o^*$
24:         $t_{\text{fin}} \leftarrow K[\Delta_{|K|}]$
25:     if $t_{\text{fin}} > \sigma^*[\theta]$ then return null      ▷ Deadline missed
26:     $\widetilde{\Sigma} \leftarrow \widetilde{\Sigma} \setminus \sigma^*$
27: return $K$
```

only a part of it (Lines 13–17). In the latter case, the mapping segment is split at the point where the job completes (Line 13), the job is added only to the first part of split segment.

To track the remaining time of the job while iterating the mapping segments, the algorithm initialize t_{rem} in Line 6 and updates it in Lines 11 and 15. If the job is not completed after iterating through all existing segments, the new mapping segment is appended to accommodate the remaining execution time (Lines 21–24).

After finishing mapping the job, the algorithm verifies that the job completes before its deadline (Line 25). If the deadline is missed, the algorithm returns null.

By following the EDF policy, the algorithm prioritizes time-critical job requests, placing them into the earliest possible mapping segments.

Note that the proposed algorithm is backward-compatible with the single-threaded version presented in [138], generating the same spatio-temporal mappings by following the MDF and EDF policies. However, because multi-threaded applications require all threads to be scheduled on the same mapping segments, the original single-threaded algorithm cannot be directly applied.

4.3.2 *Evaluation*

To assess the effectiveness of the MMKP-MDF algorithm, a set of experiments was conducted using the representative dataflow applications mapped to a heterogeneous embedded platform. Section 4.3.2.1 describes the experimental setup, the generation of the experimental workload, and the alternative algorithms used for comparison. Section 4.3.2.2 presents the acceptance rate and the energy-efficiency of the generated spatio-temporal mappings, while Section 4.3.2.3 analyzes the runtime overhead of the approach.

4.3.2.1 *Experimental Setup and Test Generation*

The experiments utilize three dataflow applications from the automotive and multimedia domains:

- A *speaker recognition* algorithm with 8 processes [25],
- An *audio filter*, a stereo frequency filter with 8 processes [68], and
- A *pedestrian recognition* algorithm with 6 processes, provided by Silexica[1].

To obtain the operating points, these applications were exhaustively benchmarked with input data of different sizes on the Hardkernel Odroid-XU4, which features an Exynos 5422 big.LITTLE chip with four Cortex-A15 and four Cortex-A7 cores, fixed at frequencies of 1.8 GHz and 1.5 GHz, respectively. The power consumption of the Odroid-XU4 board was measured using a ZES Zimmer LMG450 Power Analyzer connected to the DC input, with an external readout rate of 20 Sa/s.

To identify Pareto-optimal operating points, each candidate configuration was executed 50 times to obtain average execution times and energy consumptions. This process yielded 36 Pareto-optimal operating points for the audio filter, 35 for pedestrian recognition, and 28 for speaker recognition.

The multi-application setup consists of 1676 test cases. Each test case includes one to four jobs, characterized by their current progress ratio and the remaining deadline. Approximately 31.9 % of the test cases consist of job requests for a single application (uniformly distributed

---

[1] Silexica was acquired by Xilinx, which was subsequently acquired by AMD. The tool support for dataflow applications has since been discontinued.

among each application and input data), while the remaining 68.1 % are application mixes. In around 22.6 % of the tests, the progress state of the jobs is set to zero (initial state). For all others, a progress rate is randomly chosen in the range 0–0.9, except for the first job, which emulates a newly arrived job.

To set deadlines, an operating point is randomly selected, the remaining time to finish the job using this operating point is calculated, and then scaled by a factor. For weak deadlines, large factors in the range 2–6 are randomly chosen. For tight deadlines, factors are randomly selected in the range 0.6–2. Table 4.3 reports the number of tests for each combination of number of jobs and deadline level.

Table 4.3: Number of test cases by jobs and deadline level

# Jobs / Deadline level	1	2	3	4
Weak	15	255	255	230
Tight	35	340	340	206

The proposed algorithm, *MMKP-MDF* (Section 4.3.1) was implemented in Python 3 within Mocasin framework (Section 2.5), and executed on a 3.20 GHz Intel Core i5-6500 CPU. The RM prototype receives the Pareto-optimal operating points, reads a test case, and maps the applications onto the Odroid-XU4 platform.

To evaluate the solution, alternative algorithms were implemented:

- *EX-MEM*: This algorithm exhaustively checks all possible mappings for each of the mapping segments. After constructing each mapping segment, it identifies the shortest jobs, cuts the segment at that point, and generates the next mapping segment. To accelerate the algorithm, memoization is used by storing and reusing the best energy consumption for a given current state (a pair of jobs, their progress rates, and time).

- *MMKP-LR*: Based on the Lagrangian Relaxation algorithm described in [207], this algorithm solves Lagrangian relaxations of the MMKP problem using a subgradient method (limited to 100 iterations) and iteratively maps applications in order of increasing minimum cost. Similarly, during job mapping, the algorithm iteratively checks the configurations in order of increasing cost. An operating point is selected if there are enough resources and the job can meet its deadline either using this operating point until completion or by reconfiguring to another point at the end of the mapping segment (an optimistic check). This process is repeated for the next mapping segment, so the analysis scope is limited to a single mapping segment.

4.3.2.2 *Acceptance Rate and Energy Efficiency*

All three implemented algorithms were evaluated with respect to the percentage of test cases for which they could find a feasible spatio-temporal mapping. All algorithms successfully found solutions for 100 % of the test cases with weak deadlines. The results differ significantly for the tests with tight deadline, as shown in Figure 4.3. For test cases with one or two jobs, all three algorithms achieved a similar acceptance rate, with differences within 2.3 %. For tests with more jobs, EX-MEM shows a significantly higher success rate than the other two algorithms, up to 14.1 % higher. In all test cases, MMKP-LR and MMKP-MDF achieve similar acceptance rates, with differences within 3.6 % in favour of MMKP-LR.

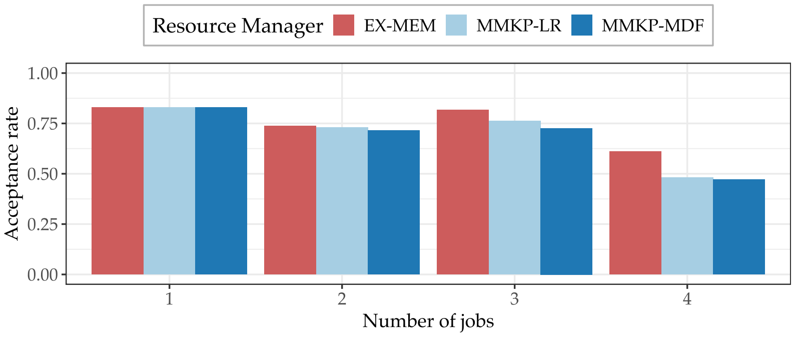


Figure 4.3: Acceptance rate of different algorithms for test cases with tight deadlines

In terms of energy efficiency, the algorithms are compared to the optimal solutions obtained by EX-MEM, which serves as the baseline. For each successfully found solution, the relative energy consumption compared to EX-MEM is computed, and the geometric mean of these values is reported for each test group, as shown in Table 4.4.

All algorithms generate optimal solutions in case of a single job. For tests with weak deadlines, the relative energy consumption of MMKP-MDF solutions increases slowly from 0.03 % for two jobs to 0.99 % for four jobs (in geometric mean). Overall, for tests with weak deadlines, the MMKP-MDF solutions are off by 0.42 % from the optimal ones. For tight deadlines, the relative energy consumption of MMKP-MDF varies non-monotonically with the number of jobs, and they are off by 7.56 % in geometric mean.

For MMKP-LR, the relative energy consumption increases with the number of jobs, with geometric means of 14.52 % and 19.23 % for weak and tight deadlines, respectively. Overall, MMKP-MDF generates more energy-efficient spatio-temporal mappings by 13.1 % compared to MMKP-LR.

Figure 4.4 presents the relative energy consumption across test scenarios using a monotonic curve layout. In this figure, the relative energy values for all test cases are arranged in ascending order. The

Table 4.4: Geometric mean of the relative energy consumption across different numbers of jobs and deadline levels compared to EX-MEM.

	MMKP-LR		MMKP-MDF	
Number of Jobs	Weak	Tight	Weak	Tight
1	1.0000	1.0000	1.0000	1.0000
2	1.0480	1.1291	1.0003	1.0682
3	1.1534	1.2250	1.0031	1.0978
4	1.2648	1.3404	1.0099	1.0618
Overall	1.1452	1.1923	1.0042	1.0756
(all levels)	1.1665		1.0356	

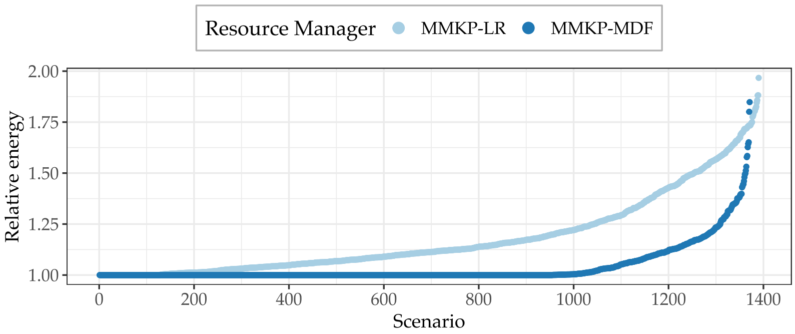


Figure 4.4: Monotonic distribution curves of the relative energy consumption compared to EX-MEM (lower is better).

curve's rightmost endpoint indicates the success rate, and its contour provides insights into the distribution of relative energy values. As shown, MMKP-MDF generates optimal solutions for 954 tests (69.6 % of successfully scheduled cases), while MMKP-LR does so for only 125 tests (9.0 %).

4.3.2.3 *Runtime Overhead*

Figure 4.5 shows the box plots and average values of the execution times of different algorithms, differentiated by the number of jobs. The execution time increases with the number of jobs for all three implementations. As expected, EX-MEM displays exponential growth, with an average of 152 s to schedule four jobs, while the median and worst-case values are 22.65 s and 2550 s (≈37.5 min), respectively. The execution time of MMKP-LR and MMKP-MDF grow less rapidly. MMKP-LR requires around 1.3 ms to schedule one job and around 163 ms for four jobs. MMKP-MDF is significantly faster, requiring only 5.7 ms on average for four jobs, with a worst-case of 21.6 ms.

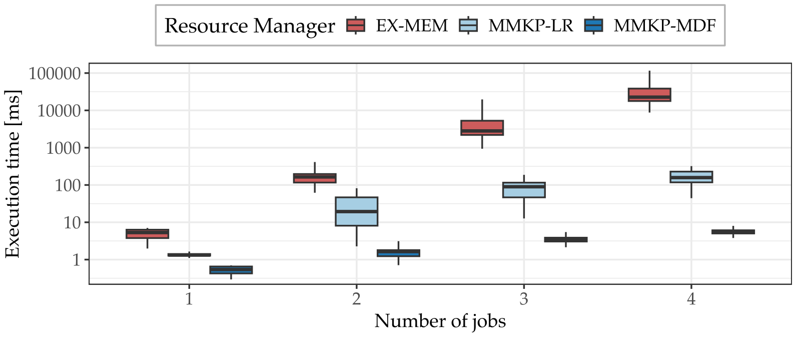


Figure 4.5: Box plots and average values summarizing the mapping overhead for different algorithms

In summary, the proposed MMKP-MDF algorithm achieves comparable scheduling success rate to the MMKP-LR approach while outperforming it in terms of overall energy efficiency and runtime overhead. MMKP-MDF generates solutions within 21.6 ms, which makes it a suitable candidate for implementation in a fully functional runtime resource manager. As mentioned in the experimental setup, the overhead analysis for all algorithms was performed on a prototyped RM written in Python 3. Better performance can be expected from an implementation in C/C++.

4.4 FLEXIBLE SPATIO-TEMPORAL MAPPING

The second class of approaches generates *flexible* spatio-temporal mappings, fully exploring the search space of this decision model. These algorithms explicitly incorporate application reconfigurations into their mapping plans, allowing them to better adapt to dynamic workloads than fixed-point mapping strategies.

Two mapping approaches are presented. The first, called Spatio-Temporal Evolutionary Mapping (STEM), leverages Memetic Algorithms (MAs) [57] and is detailed in Section 4.4.1. The second approach, Fast Flexible Energy-Minimizing Scheduler (FFEMS), employs fast greedy heuristics, as described in Section 4.4.2. Section 4.4.3 presents the evaluation of these algorithms.

4.4.1 *STEM: Spatio-Temporal Evolutionary Mapping*

The *Spatio-Temporal Evolutionary Mapping* (STEM) algorithm employs Memetic Algorithms (MAs) to generate flexible spatio-temporal mappings, achieving the combined benefits of a Genetic Algorithm (GA) and knowledge-guided heuristics. The motivation for adopting a MA is twofold: first, to find near-optimal solutions and evaluate the efficacy of the fast algorithm; second, to analyze the influence of knowledge-guided heuristics in the search for optimal spatio-temporal mappings.

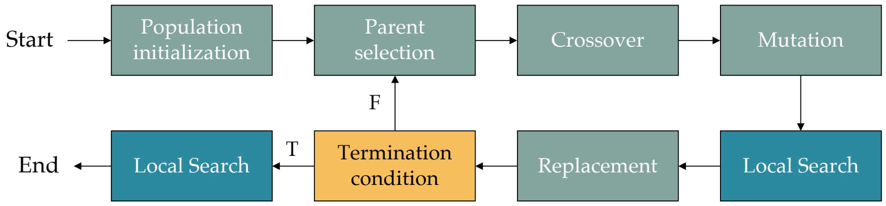


Figure 4.6: STEM algorithm flow

Figure 4.6 illustrates the flow of the STEM algorithm. The components of STEM include chromosome representation and evaluation, population initialization, genetic operators, local search methods, and survival selection, which are detailed in the following subsections.

4.4.1.1 *Chromosome Representation and Evaluation*

In evolutionary algorithms, a chromosome (also referred to as an individual) encodes a potential solution to the problem at hand. In the context of STEM, a chromosome represents a spatio-temporal mapping model, as defined in Definition 2.15 and depicted in Figure 4.2. During evolution, redundancies can arise, such as when jobs complete execution mid-segment or when operating points are assigned to completed jobs. To address these issues, operating points after job completion are discarded, and any discrepancy are considered during the fitness evaluation, thereby relaxing Equation (2.25).

FITNESS EVALUATION Chromosomes are categorized based on compliance with resource and deadline constraints, as defined in Equations (2.22) and (2.23), respectively:

1. Valid chromosomes that fulfill all constraints.
2. Chromosomes that satisfy resource constraints but violate deadlines.
3. Chromosomes that violate resource constraints.

Fitness is represented as a tuple, where the first value indicates the category, and the second value measures group-specific metrics, with lower values indicating better solutions. For valid chromosomes, the second value reflects energy consumption and a genotype-to-phenotype discrepancy. For the second category, it measures the average deadline violation as a fraction of the deadline. For the third category, it considers both the average deadline violation and the average overuse of processor element types across mapping segments.

4.4.1.2 *Population Initialization*

STEM generates an initial population of $P = 90$ individuals using a structured approach rather than purely random generation to avoid creating an excessive number of unfit solutions. First, the number of segments is randomly chosen from the range $\{1, \ldots, 2 \cdot |\Sigma|\}$. Segment durations $K[\delta_i]$ are then sampled from a normal distribution $\mathcal{N}(m_d, (m_d/2)^2)$, where m_d is the maximum deadline divided by the number of segments (rounded up to the nearest valid duration). For each segment, a random number $p \in \{1, \ldots, \min(|\Sigma|, |\mathcal{P}[\vec{\omega}]|)\}$ is generated, and p jobs are randomly sampled. The algorithm then randomly selects an operating point for these jobs while assigning $\perp$ (no mapping) to all others.

4.4.1.3 *Parent Selection and Genetic Operators*

In each iteration, STEM selects two parent individuals and applies crossover and mutation operators to produce two offsprings.

PARENT SELECTION STEM uses *exponential ranking selection* for parent selection [23]. In this strategy, the population is first sorted in ascending order of fitness, placing the fittest individuals at the beginning: $fit_1 \leq fit_2 \leq \cdots \leq fit_s$. The selection probability of ind_i is given by $p_{sel}(ind_i) = f^{i-1}/c$, where $f < 1$ is typically set close to 1 (e.g., 0.97 in STEM), and c is a normalizing coefficient. *Stochastic universal sampling* [57] selects two individuals as parents from the sorted population.

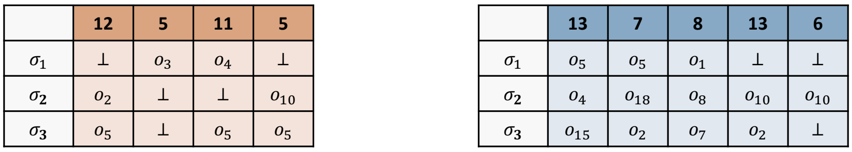
CROSSOVER Selected parents undergo crossover with a probability of $p_c = 0.7$. STEM randomly chooses one of two crossover operators shown in Figure 4.7. Given l is the smallest number of segments in the parent chromosomes, the segment-level crossover (Figure 4.7b) applies a one-point crossover technique, cutting the parent chromosome at a random point $p \in \{1, \ldots, l\}$ and swapping subsequent segments. The job-level crossover (Figure 4.7c) randomly decides for each job whether its mappings should be swapped individually (i.e., *uniform* crossover). If so, the swap is performed for the first l segments.

MUTATION After crossover, each offspring undergoes mutation with a probability of $p_m = 0.6$. Three mutation operators are defined for this step:

1. *Structure Modification*: This operator modifies the mapping structure by (a) swapping two random segments, (b) inserting a new segment at a random position, or (c) removing a random segment and redistributing its duration.

2. *Segment Duration Adjustment*: This operator changes the duration of a randomly selected segment.

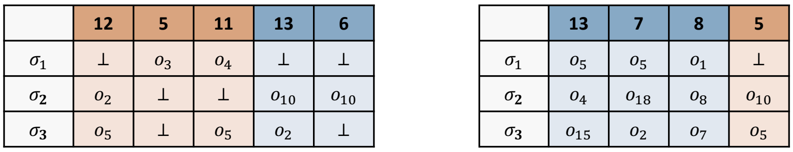
3. *Operating Point Mutation*: This operator changes the operating point index of a random job within a random segment, selecting the $\perp$ value with a probability $p_\perp = 0.5$.



	12	5	11	5
σ_1	$\perp$	o_3	o_4	$\perp$
σ_2	o_2	$\perp$	$\perp$	o_{10}
σ_3	o_5	$\perp$	o_5	o_5

	13	7	8	13	6
σ_1	o_5	o_5	o_1	$\perp$	$\perp$
σ_2	o_4	o_{18}	o_8	o_{10}	o_{10}
σ_3	o_{15}	o_2	o_7	o_2	$\perp$

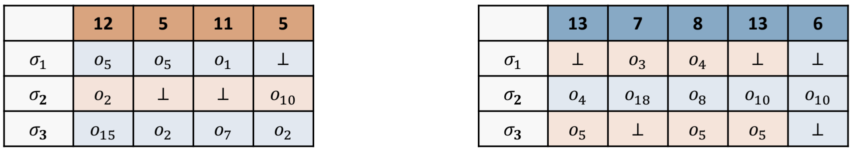
(a) Selected parent chromosomes before crossover



	12	5	11	13	6
σ_1	$\perp$	o_3	o_4	$\perp$	$\perp$
σ_2	o_2	$\perp$	$\perp$	o_{10}	o_{10}
σ_3	o_5	$\perp$	o_5	o_2	$\perp$

	13	7	8	5
σ_1	o_5	o_5	o_1	$\perp$
σ_2	o_4	o_{18}	o_8	o_{10}
σ_3	o_{15}	o_2	o_7	o_5

(b) Segment-level one-point crossover



	12	5	11	5
σ_1	o_5	o_5	o_1	$\perp$
σ_2	o_2	$\perp$	$\perp$	o_{10}
σ_3	o_{15}	o_2	o_7	o_2

	13	7	8	13	6
σ_1	$\perp$	o_3	o_4	$\perp$	$\perp$
σ_2	o_4	o_{18}	o_8	o_{10}	o_{10}
σ_3	o_5	$\perp$	o_5	o_5	$\perp$

(c) Job-wise uniform crossover

Figure 4.7: Two crossover operators in STEM: segment-level one-point crossover, and job-level uniform crossover.

4.4.1.4 *Local Search Methods*

After generating offspring through genetic operators, the algorithm applies local search heuristics to refine individuals using problem-specific knowledge. This *memetic* part of the algorithm comprises several local search heuristics. The selection of methods is based on their efficiency and runtime overhead, with individuals subjected to different sets of methods based on their constraint violations.

For individuals violating resource constraints, the *Resource Overuse Reduction* method targets the segment with the highest resource overuse. It assigns the mapping of each job to $\perp$ (no mapping) and selects the variant with the best resultant fitness. This heuristic is applied with a probability of $p_{r3} = 0.5$.

Individuals that do not violate resource constraints undergo refinement with a probability $p_{r1} = p_{r2} = 0.8$, randomly selecting one of the following methods:

- *Chromosome Simplification* This method reduces the genotype-to-phenotype gap by removing false-active segments. It identifies the last active segment for each job, marks subsequent segment mappings as $\perp$, and removes idle segments.

- *Segment Manipulations* These methods explicitly modify the segment count. The first method merges two segments into one in one of two ways: (a) merging two segments with identical operating points, or (b) removing the shortest duration segment and adding its duration to a longer segment. The second method splits segments where a job finishes mid-segment. Following a split, it applies chromosome simplification and alters job mappings in the second part of the split segment.

- *Segment Duration Adjustment* This method adjusts segment durations within permissible bounds, incrementing or decrementing by a power of two to reduce the exploration space.
- *Front Propagation of Operating Points* This method collects all operating points used in the current individual and attempts to use them in earlier segments where the job has a $\perp$ mapping. Preference is given to jobs with the most significant deadline violation or highest energy consumption.

4.4.1.5 *Survivor Selection and Termination*

At the end of each iteration, the survivor selection strategy determines which individuals from the combined population advance to the next generation. The process employs the *round-robin tournament selection* [57]: For each individual, the algorithm randomly selects $q = 8$ competitors and assigns a score based on the number of competitors the individual outperforms based on their fitness. The two individuals with the lowest scores are eliminated from the population, with ties resolved randomly.

STEM terminates when the maximum number of iterations is reached. The termination condition could be further refined, for example, by terminating if no significant improvement is observed for a certain number of generations.

4.4.2 *FFEMS: Fast Flexible Energy-Minimizing Scheduler*

The *Fast Flexible Energy-Minimizing Scheduler* (FFEMS) algorithm offers an alternative approach to flexible spatio-temporal mapping, focusing on energy-efficient scheduling with significantly lower computational overhead than evolutionary methods. It achieves so by selecting operating points from an incrementally expanding Candidate Mappings Set (CMS).

Algorithms 4.3 and 4.4 detail the operation of FFEMS. Initially, jobs are ordered by Earliest Deadline First (EDF) (Algorithm 4.3, Line 2), and the operating points of each job are sorted by increasing energy consumption (Line 4).

The Candidate Mappings Set CMS is initialized to include the most energy-efficient operating points that cannot meet the job's deadline, along with the first operating point that can meet the deadline (Line 5). By including these operating points, the CMS starts with a set of highly energy-efficient mappings that might be too slow to satisfy the deadline constraint individually.

Algorithm 4.3 Main Procedure of the FFEMS Algorithm

```
Input: Job requests $\Sigma$, platform $\mathcal{P}$, Pareto-optimal operating points
    $\mathcal{O}_{\text{opt}}^{\sigma[A]}$ for each $\sigma \in \Sigma$
Output: Spatio-temporal mapping $K$
Initialize an empty spatio-temporal mapping $K$
Sort jobs in $\sigma \in \Sigma$ by EDF order
for each job $\sigma \in \Sigma$ do
    Sort the operating points $o \in \mathcal{O}_{\text{opt}}^{\sigma[A]}$ by increasing $o[\varepsilon]$
    $CMS \leftarrow$ InitCMS($\sigma$, $\mathcal{O}_{\text{opt}}^{\sigma[A]}$)
    $s \leftarrow$ False
    while $\neg s \wedge (CMS \neq \emptyset)$ do
        $\rho^* \leftarrow \sigma[\rho]$; $\theta_{\text{miss}} \leftarrow$ False
        for $i \in \{1, \dots, |K|\}$ do
            $K' \leftarrow$ GenerateJobTail($K$, $\sigma$, $CMS$, $i$, $\rho^*$)
            if $K' \neq$ null then
                $K \leftarrow K'$; $\rho^* \leftarrow 1$
                break
            if $\sigma[\theta] \leq K[\Delta_i]$ then
                $\theta_{\text{miss}} \leftarrow$ True
                break
            $o^* \leftarrow \text{argmax}_{o \in CMS}\{o[\tau] \mid o[\vec{\omega}] + K[M_i][\vec{\omega}] \leq \mathcal{P}[\vec{\omega}]\}$
            $K[M_i][\sigma] \leftarrow o^*$
            $\rho^* \leftarrow \rho^* + \rho(K[M_i][\sigma], K[\delta_i])$        ▷ $\rho(o, \delta)$ as in Eq. (2.20)
        if $\theta_{\text{miss}} =$ False then
            if $\rho^* < 1$ then
                $K' \leftarrow$ GenerateJobTail($K$, $\sigma$, $CMS$, $|K| + 1$, $\rho^*$)
                if $K' \neq$ null then
                    $K \leftarrow K'$; $\rho^* \leftarrow 1$
            if $\rho^* = 1$ then
                $s \leftarrow$ True
                break
        if $CMS = \mathcal{O}_{\text{opt}}^{\sigma[A]}$ then
            $CMS \leftarrow \emptyset$
        else
            $CMS \leftarrow$ IncrementCMS(CMS, $\mathcal{O}_{\text{opt}}^{\sigma[A]}$)
    if $\neg s$ then
        for $i \in \{1, \dots, |K|\}$ do
            $K[M_i][\sigma] \leftarrow \perp$
return $K$
```

This approach allows the algorithm to set an upper bound for the energy consumption of the job in this iteration of CMS. If the algorithm fails to generate a feasible spatio-temporal mapping with the job using the current CMS, it expands the set by including the next operating point from the sorted list. This process is repeated, gradually increasing the energy budget allocated to the job, until a feasible mapping is found or all operating points have been considered.

Algorithm 4.4 GenerateJobTail procedure

```
Input: Current spatio-temporal mapping $K$, platform $\mathcal{P}$, job request
    $\sigma$, Candidate Mappings Set $CMS$, index of the starting mapping
    segment $i_{\text{start}}$, current progress ratio $\rho^*$
Output: Updated spatio-temporal mapping $K$, or null if unsuccessful
1: $o \leftarrow$ FindMappingForTail($K$, $\mathcal{P}$, $\sigma$, $CMS$, $i_{\text{start}}$, $\rho^*$)
2: if $o = \perp$ then
3:     return null
4: $t_r \leftarrow o[\tau] \cdot (1 - \rho^*)$                ▷ Remaining execution time
5: for $i \in \{i_{\text{start}}, \dots, |K|\}$ do
6:     if $K[\delta_i] \leq t_r$ then
7:         $K[M_i][\sigma] \leftarrow o$
8:         $t_r \leftarrow (t_r - K[\delta_i])$
9:     else
10:        SplitSegment($K$, $i$, $t_r$)
11:        $K[M_i][\sigma] \leftarrow o$
12:        $t_r \leftarrow 0$
13:        break
14: if $t_r > 0$ then
15:     AppendSegment($K$, $t_r$)
16:     $K[M_{|K|}][\sigma] \leftarrow o$
17: return $K$
```

During the algorithm, for each job and its corresponding CMS, the FFEMS algorithm iterates over the current mapping segments (Lines 9–19). It first attempts to find a mapping that can be used continuously until the job's completion (call at Line 10 to Algorithm 4.4). For each potential operating point, the algorithm checks for sufficient free resources from the current mapping segment to the job's potential end time. This corresponds to a call to FindMappingForTail (Algorithm 4.4, Line 1). If a suitable mapping is found, it is applied until job completion, with mapping segments appended (Line 15) or split (Line 10) as necessary.

If no suitable continuous mapping until job completion is found, FFEMS selects the fastest available mapping for the current segment, anticipating a switch to a more energy-efficient operating point at later point (Algorithm 4.3, Line 17). If the job remains incomplete at the end of the spatio-temporal mapping K, a new mapping segment is appended, and the algorithm attempts to schedule the remaining portion of the job (Line 22).

If FFEMS fails to generate a spatio-temporal mapping including the current job with the current CMS, it extends the CMS by including the next operating point from the sorted list (Line 31). If FFEMS fails to create a valid spatio-temporal mapping with all possible operating points in $\mathcal{O}_{\text{opt}}^{\sigma[A]}$, the algorithm rejects the job (Line 33).

In the worst-case scenario, the time complexity of FFEMS is $O(|\Sigma| \cdot \left|\mathcal{O}_{\text{opt}}^{A}\right|^2 \cdot |K|)$. This arises from iterating over all jobs in the set Σ, each operating point in the set $\mathcal{O}_{\text{opt}}^{A}$ during CMS expansion, each segment of the spatio-temporal mapping K, and each operating point within the CMS during the mapping of the job until completion.

4.4.2.1 *Tail-Switch Optimization*

The energy efficiency of FFEMS can be improved using *tail-switch* optimization. If a power-intensive configuration is initially selected in GenerateJobTail, the system can switch to a slower but more energy-efficient mapping at a later point. This optimization involves iterating over all pairs of mappings (o_1, o_2) to determine the optimal switch point from the faster configuration o_1 to the more energy-efficient point o_2, while satisfying the deadline constraint. The pair that minimizes energy consumption is selected. However, this adds additional computational complexity, resulting in a time complexity of $O(|\Sigma| \cdot \left|\mathcal{O}_{\text{opt}}^{A}\right|^3 \cdot |K|)$.

4.4.3 *Evaluation*

The proposed approaches are evaluated on two heterogeneous platform models in terms of acceptance rate, energy efficiency, and runtime overhead.

Section 4.4.3.1 details the experimental setup. Section 4.4.3.2 analyzes the impact of knowledge-guided heuristics in STEM, while Section 4.4.3.3 examines the impact of different mapping decision models. Finally, Section 4.4.3.4 evaluates the runtime overhead of the algorithms.

4.4.3.1 *Experimental Setup*

The STEM and FFEMS approaches were implemented in Python 3 within the Mocasin prototyping tool, described in Section 2.5. Evaluations were conducted on two platform models:

- 4B4L: An Odroid-XU4 with an Exynos 5422 big.LITTLE chip featuring four Cortex-A15 cores and four Cortex-A7 cores, running at 1.8 GHz and 1.5 GHz, respectively.

- `8B8L`: A larger system similar to the Odroid-XU4 but with double the cores —– eight big and eight little cores.

APPLICATION MODELS As in Section 4.3.2.1, three dataflow applications from the automotive and multimedia domains were utilized in these experiments: *speaker recognition* [25], *audio filter* [68], and a *pedestrian recognition* algorithm. For the `4B4L` platform, operating points were obtained by benchmarking the real platform, yielding 28 to 36 operating points. For `8B8L`, operating points were generated using Genetic Algorithm (GA) in Mocasin, selecting 40 operating points via the k-means clustering [2].

WORKLOAD GENERATION Test cases are represented as tables of job requests, each containing an application, progress ratio, and deadline. A total of 2000 cases per platform were generated, altering the number of jobs in the request table from 1 to 10, providing 200 cases per job count. Half of these tests feature weak deadlines, and the other half have tight deadlines, thus facilitating observation under more stress-intensive situations.

Each test randomly assigned an application to each job and allocated progress ratios between 0 and 0.9 (with the first job set to 0, emulating a newly arrived job). Deadlines were determined by first selecting a random configuration, calculating the remaining time based on the configuration and remaining progress ratio, and then multiplying it by a factor. The factor was randomly chosen within a range defined by the deadline level and request number, as detailed in Table 4.5.

Table 4.5: Bounds of factor ranges used during workload generation.

Platform	Deadline Level	Factor Range Bounds	
		Lower	Upper
`4B4L`	Weak	$1.5 + 0.1 \cdot \lvert\Sigma\rvert$	$3 + 0.1 \cdot \lvert\Sigma\rvert$
	Tight	1	$1 + 0.3 \cdot \lvert\Sigma\rvert$
`8B8L`	Weak	$1 + 0.1 \cdot \lvert\Sigma\rvert$	$1.5 + 0.1 \cdot \lvert\Sigma\rvert$
	Tight	1	$1 + 0.1 \cdot \lvert\Sigma\rvert$

EVALUATED ALGORITHMS In the evaluation, several variations of the proposed algorithms were tested. For STEM, we considered four variations:

- $STEM^{100K}$: 100000 MA iterations.
- $STEM^{500K}$: 500000 MA iterations.
- $STEM^{5M}$: 5 million MA iterations.

- $STEM_{GA}^{500K}$: 500000 iterations but omitting the memetic part of the algorithm to assess the effect of knowledge-guided heuristics.

For FFEMS, a basic version, *FFEMS*, and its derivative, $FFEMS^{TS}$, which applies tail-switch optimization, were included.

Additionally, approaches generating other mapping models were assessed:

- *MMKP-LR* [207]: Generates spatial mappings using the Lagrangian Relaxation algorithm (with 200 iterations). The spatio-temporal mapping is constructed by applying this algorithm segment by segment.
- *MMKP-MDF*: The fixed-point spatial-temporal strategy described in Section 4.3.1.

overview of results The algorithms were evaluated based on:

- *Acceptance Rate*: Percentage of successfully generated spatio-temporal mappings.
- *Relative Energy Consumption*: Energy consumption normalized to the best solution found for each test case. Results are reported as a geometric mean across test cases.
- *Runtime Overhead*: Average execution time required to generate solutions.

Tables 4.6 and 4.7 summarize results for the `4B4L` and `8B8L` platforms, respectively, comparing acceptance rates, relative energy consumption, and runtime overhead under weak and tight deadline conditions. Figures 4.8, 4.9 and 4.11 illustrate the metrics by job count. Additionally, Figure 4.10 presents the relative energy consumption across test scenarios using a monotonic curve layout.

4.4.3.2 *Impact of Knowledge-Guided Heuristics in STEM*

The incorporation of knowledge-guided heuristics in STEM significantly enhances both acceptance rate and energy efficiency. As shown in Tables 4.6 and 4.7, this improvement is particularly notable in test scenarios with tight deadlines, where $\text{STEM}^{500\text{K}}$ finds solutions for up to 13.5 % more cases than its counterpart, $\text{STEM}_{\text{GA}}^{500\text{K}}$. Notably, STEM with just 100K MA iterations outperforms the version with 500K GA iterations, highlighting the value of knowledge-guided heuristics.

However, despite leveraging local search heuristics, STEM still requires a substantial number of iterations. Specifically, $\text{STEM}^{100\text{K}}$ trails FFEMS in acceptance rate by as much as 5 %. Given its enormous computational overhead, deploying STEM at runtime is impractical. In the following analysis, for STEM, the focus shifts to $\text{STEM}^{5\text{M}}$, which best demonstrates the algorithm's peak potential in discovering optimal solutions.

Table 4.6: Aggregate acceptance rates, relative energy consumption, and average runtime overhead of resource management algorithms on the `4B4L` platform under weak and tight deadline conditions.

Resource	Weak Deadlines		Tight Deadlines		Avg.
Manager	Acc. R.	Rel. Energy	Acc. R.	Rel. Energy	Overhead
MMKP-LR	94.6 %	1.2981	71.0 %	1.2442	447.4 ms
MMKP-MDF	96.8 %	1.0260	75.6 %	1.0755	9.3 ms
$\text{STEM}^{100\text{K}}$	99.8 %	1.0329	88.5 %	1.0341	65.6 s
$\text{STEM}_{\text{GA}}^{500\text{K}}$	99.2 %	1.0672	80.5 %	1.0696	216 s
$\text{STEM}^{500\text{K}}$	100 %	1.0180	91.7 %	1.0220	320 s
$\text{STEM}^{5\text{M}}$	100 %	1.0070	94.4 %	1.0071	3376 s
FFEMS	100 %	1.0376	91.6 %	1.0982	4.8 ms
FFEMS^{TS}	100 %	1.0270	91.8 %	1.0649	15.6 ms

Table 4.7: Aggregate acceptance rates, relative energy consumption, and average runtime overhead of resource management algorithms on the `8B8L` platform under weak and tight deadline conditions.

Resource	Weak Deadlines		Tight Deadlines		Avg.
Manager	Acc. R.	Rel. Energy	Acc. R.	Rel. Energy	Overhead
MMKP-LR	97.9 %	1.1552	81.0 %	1.2103	14.4 ms
MMKP-MDF	98.9 %	1.0746	80.7 %	1.0852	10.9 ms
$\text{STEM}^{100\text{K}}$	99.4 %	1.0261	88.4 %	1.0292	68.9 s
$\text{STEM}_{\text{GA}}^{500\text{K}}$	97.9 %	1.0396	78.6 %	1.0494	217 s
$\text{STEM}^{500\text{K}}$	99.8 %	1.0163	92.1 %	1.0213	342 s
$\text{STEM}^{5\text{M}}$	99.9 %	1.0087	94.4 %	1.0113	3671 s
FFEMS	100 %	1.0442	93.6 %	1.0784	5.5 ms
FFEMS^{TS}	100 %	1.0150	94.3 %	1.0445	17.2 ms

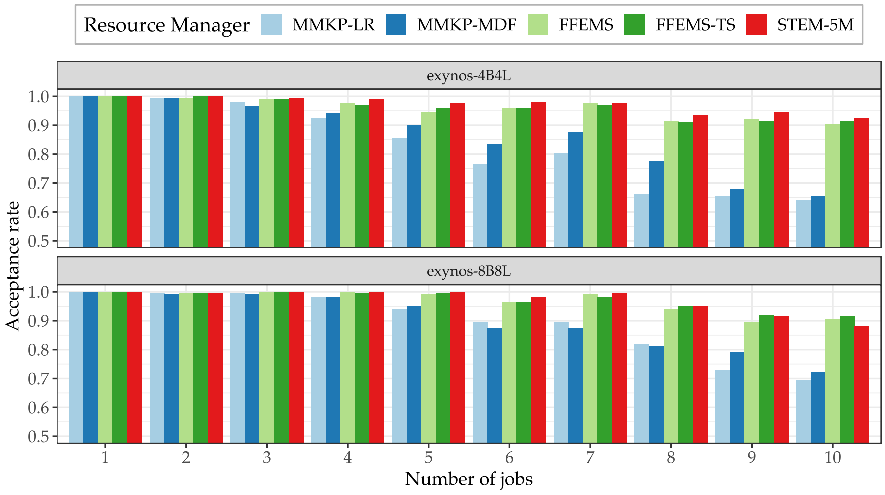


Figure 4.8: Acceptance rates of resource management algorithms across varying job counts

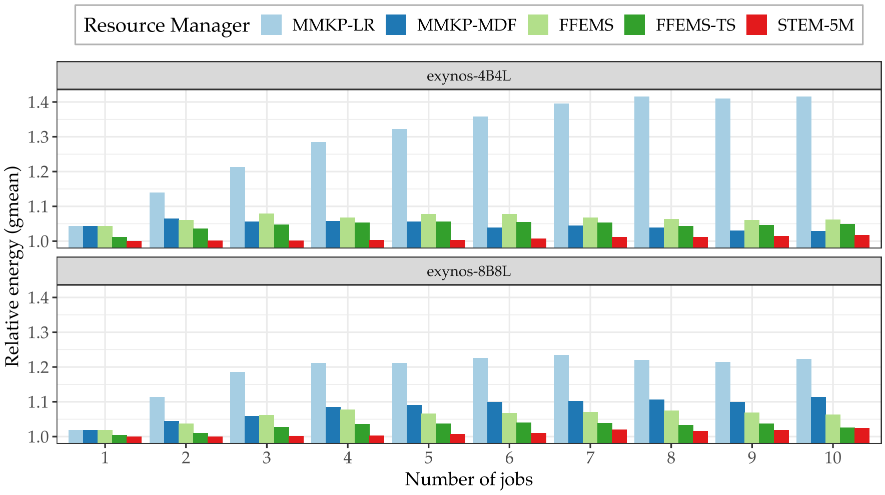


Figure 4.9: Relative energy consumption of solutions generated by different resource management algorithms across varying numbers of jobs.

4.4.3.3 *Impact of the Mapping Decision Model*

The choice of decision model is critical to the efficiency of mapping algorithms. Spatial-only approaches, such as MMKP-LR, often yields suboptimal solutions in both acceptance rate and energy efficiency. The performance of MMKP-LR varies with platform size and worsens under increased resource pressure. For instance, under tight deadlines, MMKP-LR achieves a similar acceptance rate as MMKP-MDF on the `8B8L` platform (Table 4.7), but underperforms by 4 % on the smaller `4B4L` platform (Table 4.6).

In contrast, MMKP-MDF, which employs the fixed-point spatio-temporal mapping model, demonstrates significant improvements in energy efficiency, outperforming MMKP-LR by a 26.5 % on the `4B4L` platform.

The best performance is achieved by the novel methods, FFEMS and STEM, which leverage the full flexibility of spatio-temporal mappings. As shown in Tables 4.6 and 4.7, FFEMS outperforms the spatial and fixed-point spatio-temporal approaches significantly. Under weak deadlines, it maps all test cases, while under tight ones, it schedules 16 % more test cases on the `4B4L` platform and 12.9 % on `8B8L`.

As shown in Figure 4.8, the difference in acceptance rate grows with the number of jobs, peaking at 25 % for 10 jobs on the `4B4L` platform. The FFEMS$^{\mathrm{TS}}$ variant, enhanced with the tail-switch optimization, achieves comparable acceptance rate but improves on energy efficiency, saving up to 3.4 % more energy than FFEMS.

Interestingly, FFEMS algorithms even outperform STEM$^{5\mathrm{M}}$ in acceptance rate on the `8B8L` platform, particularly with a bigger number of applications, as shown in Figure 4.8. This observation might indicate that STEM's performance declines with increasing platform and application sizes.

Figure 4.10 illustrates relative energy consumption using a monotonic curve layout. These curves arrange relative energy values for all test cases in ascending order. The rightmost endpoint indicates the acceptance rate, while the curve's contour reveals the distribution of relative energy values. These curves illustrate the differences between the decision models.

The curve corresponding to MMKP-LR (employing the spatial mapping model) reveals that for most test cases, it fails to select the most energy-efficient configurations. This behavior can be attributed to MMKP-LR's inability to consider postponing certain job executions, leading it to opt for less energy-efficient configurations.

In contrast, MMKP-MDF's curve is more aligned with the ideal energy values, showing a slight improvement in the acceptance rate. This improvement is explained by a more relaxed constraint, allowing the selection of configurations otherwise infeasible within a single mapping segment.

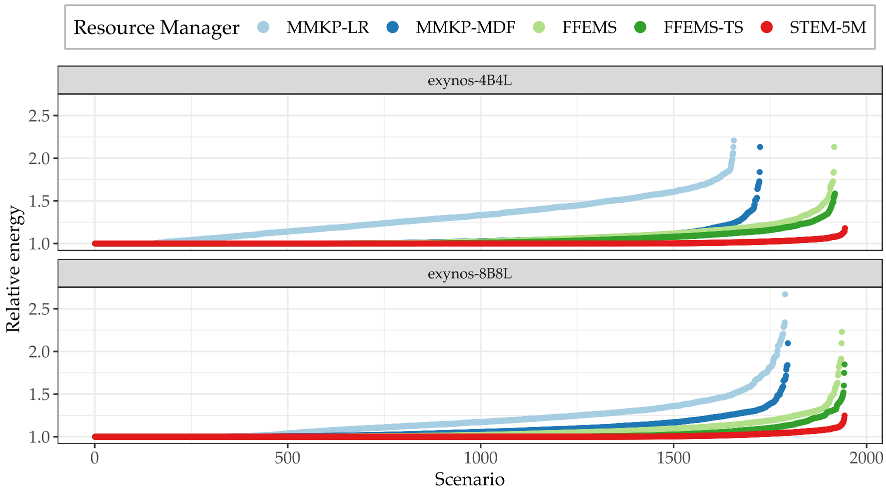


Figure 4.10: Monotonic distribution curves of the relative energy consumption for different resource management algorithms. The rightmost endpoint of each curve indicates the acceptance rate.

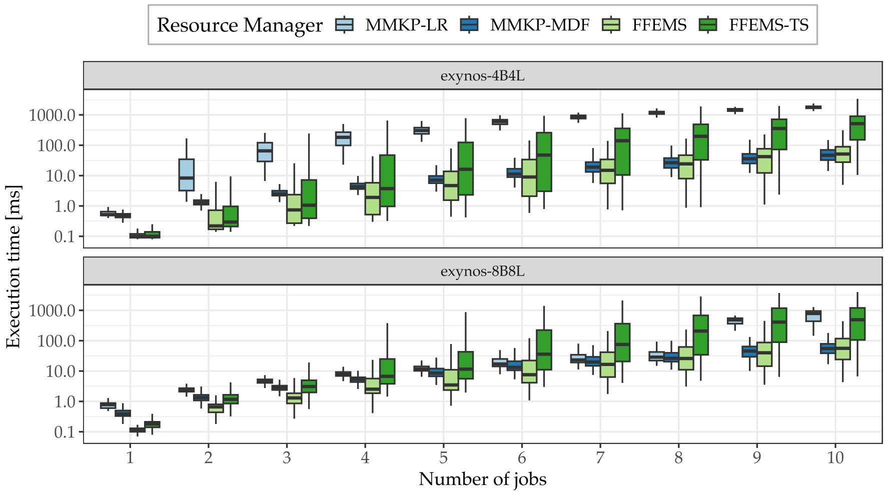


Figure 4.11: Runtime overhead of different resource management algorithms (results for STEM are excluded for readability).

Lastly, the adoption of flexible spatio-temporal mappings significantly improves the acceptance rate. This improvement is attributed to the flexible decision structure, which permits jobs to be dynamically reconfigured to meet their deadlines.

4.4.3.4 *Runtime Overhead*

Figure 4.11 shows the execution times of different algorithms using box plots. The results indicate an increase in mapping overhead with the number of applications for all algorithms. MMKP-LR exhibits higher overhead on the smaller `4B4L` platform where resource pressure is increased, possibly due to its inability to converge until the final iteration.

MMKP-MDF and FFEMS achieve similar execution times, both capable of scheduling ten applications within 100 ms. However, FFEMS schedules significantly more test cases, making it the superior choice. As expected, FFEMS$^{\mathrm{TS}}$ incurs higher overhead due to tail-switch optimization, scheduling 10 jobs within 100 ms – 1 s, a trade-off for improved energy efficiency.

4.5 SYNOPSIS

This chapter investigated the impact of the mapping decision model, particularly the temporal component of spatio-temporal mappings, on real-time acceptance and energy efficiency. Three algorithms were proposed: MMKP-MDF, which generates fixed-point spatio-temporal mappings, and two flexible spatio-temporal mapping algorithms, STEM and FFEMS.

The results indicate that employing fixed-point spatio-temporal mapping models enhances the energy efficiency of the solutions found, while the flexible variant of this model significantly increases acceptance rate. Among the algorithms, STEM produces the most energy-efficient solutions, but its enormous overhead makes it unsuitable for use in runtime systems.

In contrast, FFEMS demonstrates an excellent balance between performance and runtime overhead. With a runtime overhead similar to MMKP-MDF (up to 100 ms for 10 jobs), FFEMS successfully maps up to 16 % more test cases successfully. Moreover, its tail-switch optimization further improves energy efficiency, albeit with a slight increase in runtime overhead, which remains acceptable for runtime systems.

5

DOMAIN-SPECIFIC HYBRID MAPPING FOR BASEBAND PROCESSING

Baseband processing is a critical component in wireless communication systems, responsible for processing signals received by the base station from multiple User Equipments (UEs). The increasing demands of modern telecommunication standards — higher data rates, lower latencies, and support for diverse applications — present significant challenges for baseband processing implementations. Traditionally, specialized hardware has been employed to meet performance and energy efficiency requirements. However, the need for flexibility to handle heterogeneous and dynamic workloads has led to a shift towards software-based solutions.

This chapter addresses the challenge of balancing flexibility and efficiency in baseband processing by proposing a domain-specific Hybrid Application Mapping (HAM) approach. By leveraging the structure of the baseband processing application and refining mapping methodologies, the proposed approach improves scalability at design time and reduces runtime overhead, enabling efficient execution on heterogeneous platforms.

The chapter begins with a discussion of existing approaches to baseband processing in Section 5.1, highlights their limitations, and motivates the need for hybrid application mapping. Section 5.2 details the structure and parameters of baseband processing, followed by Section 5.3, which introduces the phased-sequential task graph model for these applications. Section 5.4 presents a domain-specific algorithm for generating efficient operating points for such task graphs. Section 5.5 outlines enhancements to the MMKP-MDF algorithm, including the reuse of previous solutions to minimize runtime overhead. Section 5.6 evaluates the proposed hybrid mapping methodology, analyzing both design-time operating point generation and runtime resource management. The chapter concludes with a synopsis in Section 5.7.

A Note on Publications and Contributions

The content in this chapter, including the discussion of existing approaches, baseband processing architecture, application model, algorithms, figures, and results, was previously published in Khasanov, Robledo, Menard, Goens, and Castrillon, "Domain-specific Hybrid Mapping for Energy-efficient Baseband Processing in Wireless Networks," 2021 [100]. This work is a collaborative effort with several contributors, each focusing on specific aspects of the overall approach.

Julian Robledo contributed to the benchmarking, modeling, and validation of the baseband receiver application and developed a multi-application simulation component within MOCASIN. Christian Menard implemented a work-stealing algorithm mimicking LTE PHY application behavior. Andrés Goens developed state-of-the-art mapping algorithms, including simulated annealing and tabu search, used for comparison in the evaluation.

The author of this thesis formalized the application model for task graphs with phased-sequential structure, designed and developed the algorithms for operating point generation and runtime resource allocation, and introduced energy evaluation within the MOCASIN framework.

All contributors worked collaboratively to refine the overall approach through iterative discussions and joint evaluations.

5.1 APPROACHES TO BASEBAND PROCESSING

The rapid growth of mobile data demands poses significant challenges for traditional distributed Radio Access Networks (RANs). Compared to 4G Long-Term Evolution (LTE), the 5G network aims to support $1000 \times$ higher data traffic [5], introducing new use cases and increasing workload heterogeneity for Baseband Units (BBUs). As the number of connected devices increases in the Internet of Things (IoT), diverse devices generate data traffic with varying requirements. Meeting these challenges requires more flexible BBUs, ranging from full-fledged base stations to femtocells and smart surfaces [190], with demands expected to grow under future telecommunication standards [159].

To enhance flexibility and adaptability, there has been a shift towards software-based baseband processing. Cloud RANs (cRANs) [44] improve resource allocation by forwarding preprocessed data from distributed Remote Radio Heads (RRHs) to a centralized pool of BBUs [112]. Virtualized RANs (vRANs) [202] further facilitate the move from costly, Commercial Off-The-Shelf (COTS)-based solutions to software-based ones, leveraging cRANs and the utilization of programmable hardware. Software solutions offer the flexibility to support multiple standards and dynamic workloads, while reducing development costs and time-to-market. Similar motivations led to research on Software-Defined Radio (SDR) for handheld devices [39, 170, 197]; however, the dynamic and multi-user nature of BBUs in cRANs presents additional challenges.

This flexibility, however, comes at the expense of energy efficiency. Specialized hardware like Application-Specific Integrated Circuits (ASICs) offers orders of magnitude more efficiency than General-Purpose Processors (GPPs) for baseband processing tasks [17] but lacks flexibility to changing standards. Reconfigurable hardware, such as FPGAs, offers a compromise but struggles with runtime workload variability. Modern hardware approaches, including dynamically reconfigurable FPGAs, Coarse-Grained Reconfigurable Arrays (CGRAs) [96], and ASIPs [169], address these issues by targeting different points in the flexibility-efficiency spectrum, but they also increase hardware heterogeneity. These systems demand a structured approach to software/hardware co-design [12, 73, 208].

Early baseband processing implementations on GPPs used techniques like Single Instruction Multiple Data (SIMD) optimizations, look-up tables for latency reduction [93], and multi-threading [189] for performance improvement. However, these approaches relied on static mappings, limiting their adaptability to dynamic workloads. To address this, dynamic scheduling methods like work-stealing were introduced, as seen in the PHY benchmark [178], where tasks are distributed across processors at runtime. While flexible, such methods may not fully exploit the capabilities of heterogeneous architectures, potentially resulting in suboptimal energy efficiency — similarly as in other application domains (Section 3.2).

Some efforts in small cell base stations include heterogeneous multiprocessor platforms with hardware accelerators and ASIPs [185]. For resource management, the authors use a task graph and describe a mapping approach that incurs high power consumption. More recent approaches introduce an array-based multi-core architecture for baseband processing [198, 199], leveraging dataflow-based mapping methodology. However, this mapping methodology are tied to the specific hardware architecture and cannot be easily extended to support heterogeneous platforms. Although the baseband application contains certain irregularities potentially leading to divergent control, authors in [114] proposed a novel algorithm that is well-suited to a GPU cluster.

In summary, existing approaches involve trade-offs between adaptivity, performance, and energy efficiency. Software-based solutions offer flexibility but lack efficiency, while hardware-accelerated methods are efficient but struggle with workload variability. As discussed in this thesis, HAM approaches excel in both efficiency and adaptivity. However, general HAM approaches are domain-oblivious, designed for comparatively smaller applications, and, as shown in Section 5.6, do not scale well to baseband processing, where each UE requires handling dozens of computational tasks. Scaling these methods accordingly requires domain-specific improvements which leverage the structure of the application.

This chapter introduces a domain-specific HAM approach tailored to baseband processing. By leveraging the inherent structure and regularity of baseband tasks, the proposed method improves scalability during design time and reduces runtime overhead. This specialization enables flexible and energy-efficient execution of baseband processing applications on heterogeneous platforms.

5.2 BASEBAND PROCESSING ARCHITECTURE AND PARAMETERIZATION

Baseband processing is computationally demanding and has historically required significant effort in hardware-software co-design by expert teams at telecommunication companies. In LTE uplink communication at the base station, incoming data is structured into subframes, each corresponds to 1 ms in the time domain. Every subframe consists of two adjacent slots, each lasting 0.5 ms and containing 14 symbols. The frequency domain is divided into subcarriers spaced 15 kHz apart. Each subframe contains data from up to 10 User Equipments (UEs), received by the base station's antennas. Each UE transmits an integer number of Physical Resource Blocks (PRBs), where a PRB is the smallest allocatable unit of data for a UE, corresponding to one subframe in the time domain and 12 subcarriers in the frequency domain [61].

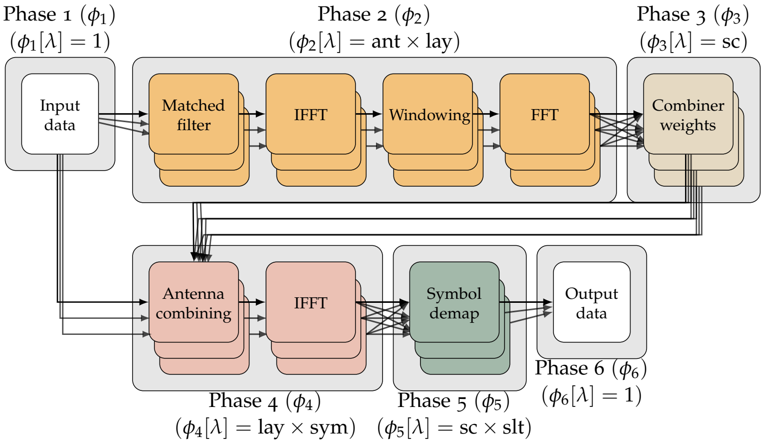


Figure 5.1: Block diagram of a baseband receiver

A typical model of a baseband receiver is shown in Figure 5.1. The physical layer of baseband processing consists of computational tasks grouped into phases ϕ, which reconstruct transmitted data before passing it to the upper layers of the communication stack. Typical tasks include a matched filter, Fast Fourier Transform (FFT), windowing, Inverse FFT (IFFT), antenna combining, combiner weights calculation, and soft symbol demapping. These tasks exhibit high runtime variation due to multiple parameters controlling their functionality. These include the number of UEs, the number of PRBs allocated per UE, the number of antennas (*ant*) at the base station, the modulation scheme, the number of concurrent data streams (layers, or *lay*), the number of slots (*slt*) per subframe (fixed to two in LTE but variable in 5G), the number of subcarriers (*sc*) per slot in the frequency domain, and the number of symbols (*sym*) per slot in the time domain. This parameterization supports modern and upcoming wireless communication standards, driving the trend towards more flexible software implementations over inflexible hardware.

These parameters, varying in runtime, influence the computational structure of the baseband processing chain for a single UE within a subframe. The parameters determine the data parallelism degree for each phase ϕ, where all tasks in a phase share the same level of parallelism, indicated by λ in Figure 5.1. Parameters such as the number of PRBs allocated to the UE and the modulation scheme also control the computational intensity of the task instances and thus the performance requirements. Each user in a subframe may be served by a different dataflow graph. 5G, operating at a higher bandwidth, exhibits even greater parameterization. In addition to Enhanced Mobile Broadband (eMBB), 5G introduces new use cases: Massive Machine-Type Communication (mMTC), and Ultra-Reliable Low-Latency Communication (URLLC) [216]. Each UE request in an upcoming subframe must be processed within its deadline; otherwise, it is discarded. Use cases have varying deadlines: 2.5 ms for eMBB and mMTC, and 0.5 ms for URLLC. Upcoming standards show a clear trend towards richer parameterization and a wider range of supported use cases.

For this work, the PHY benchmark [178] is utilized, featuring a structure similar to that in Figure 5.1. The benchmark is an implementation of an LTE baseband processing system and is suitable for handling realistic LTE workloads. The benchmark captures the parallelism of LTE through a configurable multi-threaded implementation.

5.3 TASK GRAPH WITH PHASE-SEQUENTIAL STRUCTURE

To formalize the application model, jobs are represented as task graphs with a *phase-sequential structure*, as shown in Figure 5.2. This model is suitable for applications that can be divided into distinct, data-parallel phases, as described in the previous section.

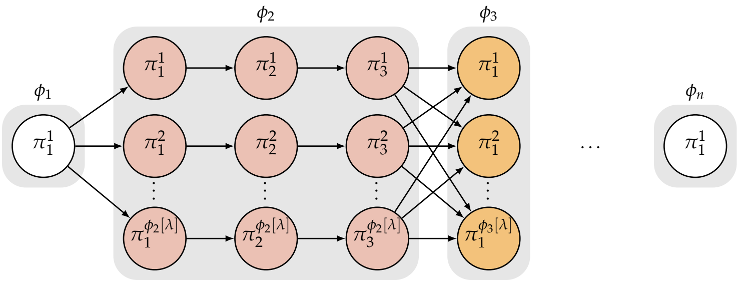


Figure 5.2: Task graph with a phase-sequential structure

A phase in this context represents a group of tasks executed sequentially but capable of data-level parallelization. Formally, a phase is defined as follows:

Definition 5.1 (Phase). A *phase* ϕ is a tuple $\phi := \phi\langle \Pi, \lambda \rangle$, where

- $\Pi = (\pi_1, \dots, \pi_{|\Pi|})$ is an ordered set of tasks connected in sequence. Specifically, for each $i \in \{1, \dots, |\phi[\Pi]| - 1\}$, a communication edge connects task π_i to task π_{i+1}.
- λ is the parallelization factor of the phase, indicating the number of data-parallel replicas.

Each phase is parallelized by replicating it λ times. Tasks in each replica, denoted by $\phi[\Pi^k] = \left(\pi_1^k, \dots, \pi_{|\phi[\Pi]|}^k\right)$ for $k \in \{1, \dots, \lambda\}$, are independent of tasks in other replicas of the same phase.

The phases in the task graph execute sequentially. The final tasks of a phase ϕ_i connect to the first tasks in all replicas of the subsequent phase ϕ_{i+1}. Formally, for all $k' \in \{1, \dots, \phi_i[\lambda]\}$ and $k'' \in \{1, \dots, \phi_{i+1}[\lambda]\}$, there is a communication edge from $\phi_i[\pi_{|\Pi|}^{k'}]$ to $\phi_{i+1}[\pi_1^{k''}]$.

The task graph with a phase-sequential structure is formalized as follows:

Definition 5.2 (Phased Task Graph)**.** A *phased task graph* model $\mathcal{M}^{\text{PTG}}$ is defined by the ordered set of phases $\Phi = (\phi_1, \dots, \phi_{|\Phi|})$, i.e., $\mathcal{M}^{\text{PTG}} := \mathcal{M}^{\text{PTG}}\langle\Phi\rangle$.

With the definitions above, the phased task graph application is specified as $A\langle\mathcal{M}^{\text{PTG}}, \Gamma, \mathcal{V}\rangle$, where Γ is the set of configuration parameters, and $\mathcal{V}$ is the set of all tasks in the application, including all replicas resulting from parallelization (cf. Definition 2.4). In the context of the baseband processing application presented in Section 5.2, the configuration parameters include $\Gamma = (\gamma_{\text{PRBs}}, \gamma_{\text{mod}}, \gamma_{\text{lay}}, \gamma_{\text{ant}})$, representing parameters such as the number of PRBs, modulation scheme, number of layers, and number of antennas. Based on these values, the parallelization degree $\phi[\lambda]$ are determined (as depicted in Figure 5.1).

Each task is annotated with performance and energy data. Specifically, each task π is associated with vector $\vec{\tau}$ and $\vec{\varepsilon}$, where $\pi[\tau_i]$ and $\pi[\varepsilon_i]$ denote the execution latency and energy consumption of task π on processing element type Ω_i (cf. Section 2.3.3). If a task cannot execute on a particular processing element type Ω_i, then $\pi[\tau_i] = \pi[\varepsilon_i] = \perp$.

5.4 EFFICIENT MAPPING ALGORITHM FOR PHASED TASK GRAPHS

For this specialized application model, an efficient mapping algorithm is proposed that leverages the phase-sequential structure of task graphs. The algorithm takes as input the task graph A with its inherent structure, the performance-energy data for each task $\pi\langle\vec{\tau}, \vec{\varepsilon}\rangle$, the sets of regular processor cores Ψ^{reg} and accelerators Ψ^{acc}. It maps tasks to the resources sequentially, processing each phase $\phi \in A[\Phi]$ in order.

The implementation of the mapping for a single phase is presented in Algorithm 5.1. The algorithm mainly consists of two steps: mapping

Algorithm 5.1 PHASED-FAST: Mapping a Single Phase onto Heterogeneous Processing Elements

```
Input: A task graph phase $\phi$, regular processing elements $\Psi^{\mathrm{reg}}$, accelerators $\Psi^{\mathrm{acc}}$ ($\Psi^{\mathrm{reg}} \cup \Psi^{\mathrm{acc}} = \mathcal{P}[\Psi]$)
Output: A mapping $\mu_\phi : \bigcup_{k=1}^{\phi[\lambda]} \phi[\Pi^k] \to \mathcal{P}[\Psi]$, phase latency $\tau_\phi$, phase energy $\varepsilon_\phi$
// Initialization
Initialize an empty mapping $\mu_\phi$
for all $\psi \in \Psi^{\mathrm{reg}}$ do
    $t[\psi] \leftarrow \sum_{\pi \in \phi[\Pi]} \pi[\tau_\psi]$          ▷ Latency of a single replica on $\psi$
    $T[\psi] \leftarrow 0$          ▷ Total latency on $\psi$
// Load Balancing on Regular Processing Elements
for $j \in \{1, \dots, \phi[\lambda]\}$ do
    $\psi^* \leftarrow \operatorname{argmin}_{\psi \in \Psi^{\mathrm{reg}}} (t[\psi] + T[\psi])$
    $T[\psi^*] \leftarrow T[\psi^*] + t[\psi^*]$
    for all $\pi \in \phi[\Pi]$ do
        $\mu_\phi[\pi^j] \leftarrow \psi^*$
// Initialization of Accelerators' Total Latency
for all $\psi \in \Psi^{\mathrm{acc}}$ do
    $T[\psi] \leftarrow 0$
    for all $\pi \in \phi[\Pi]$ do
        if $\pi[\tau_\psi] = \perp$ then
            $T[\psi] \leftarrow T[\psi] + \min_{\psi' \in \mathcal{P}[\Psi]} \pi[\tau_{\psi'}]$          ▷ Initial idle time
        else
            break
// Re-Balancing to Accelerators
for all $\pi \in \phi[\Pi]$ do
    $\Psi^{\mathrm{acc}}_\pi \leftarrow \{\psi \in \Psi^{\mathrm{acc}} \mid \pi[\tau_\psi] \neq \perp\}$
    if $\Psi^{\mathrm{acc}}_\pi = \emptyset$ then
        continue
    for $j \in \{1, \dots, \phi[\lambda]\}$ do
        $\Psi^{\mathrm{reg}}_\pi \leftarrow \{\psi \in \Psi^{\mathrm{reg}} \mid \exists k, \mu[\pi^k] = \psi\}$
        $\psi_{\mathrm{reg}} \leftarrow \operatorname{argmax}_{\psi \in \Psi^{\mathrm{reg}}_\pi} T[\psi]$
        $k \leftarrow \min\{k \in \{1, \dots, \phi[\lambda]\} \mid \mu[\pi^k] = \psi_{\mathrm{reg}}\}$
        $\psi_{\mathrm{acc}} \leftarrow \operatorname{argmin}_{\psi \in \Psi^{\mathrm{acc}}_\pi} T[\psi]$
        if $T[\psi_{\mathrm{reg}}] - \pi[\tau_{\psi_{\mathrm{reg}}}] < T[\psi_{\mathrm{acc}}] + \pi[\tau_{\psi_{\mathrm{acc}}}]$ then
            break
        $\mu_\phi[\pi^k] \leftarrow \psi_{\mathrm{acc}}$          ▷ Remap $\pi^k$ from $\psi_{\mathrm{reg}}$ to $\psi_{\mathrm{acc}}$
        $T[\psi_{\mathrm{reg}}] \leftarrow T[\psi_{\mathrm{reg}}] - \pi[\tau_{\psi_{\mathrm{reg}}}]$
        $T[\psi_{\mathrm{acc}}] \leftarrow T[\psi_{\mathrm{acc}}] + \pi[\tau_{\psi_{\mathrm{acc}}}]$
// Calculate Phase Latency and Energy
$\tau_\phi \leftarrow \max_{\psi \in \mathcal{P}[\Psi]} T[\psi]$
$\varepsilon_\phi \leftarrow \sum_{\pi \in \phi[\Pi], j \in \{1, \dots, \phi[\lambda]\}} \pi[\varepsilon_{\mu_\phi[\pi^j]}]$
return $\mu_\phi$, $\tau_\phi$, $\varepsilon_\phi$
```

to regular resources (Lines 5–9), and then reassigning some tasks to accelerators (Lines 17–30). A step-by-step visualization of the algorithm is provided in Figure 5.4, using Phase 2 of the baseband receiver as the input model of the task graph phase depicted in Figure 5.3.

The sequence of tasks in a single replica of a phase is independent of the sequences in the other replicas of that phase (cf. Figure 5.2). To reduce the communication overhead, it is preferable to map all tasks to a single regular processing element. At this stage, tasks in the replica are treated as virtually fused, with the replica latency $t[\psi]$ computed as the sum of the latencies of its tasks (Line 3). The total latency of already mapped processes on resource ψ is tracked in $T[\psi]$.

For each replica, the resource ψ that minimizes the total latency after adding the replica is determined, and the replica is mapped to it (Lines 5–9). This heuristic aims to balance the total latency across regular resources Ψ^{reg}.

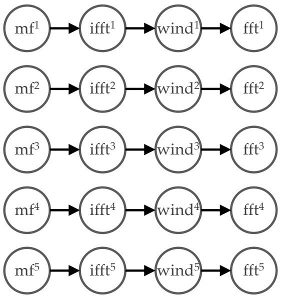
Figure 5.4 illustrates the working of the algorithm for a target platform with three processing elements: a *big* core ψ_{big}, a *little* core ψ_{little} and an *FFT/IFFT* accelerator ψ_{fft}. In this example, the mapping is performed on Phase 2 of the baseband processing application (see Section 5.2), for which the performance and energy data are shown in Figure 5.3. The sum of the latencies on a big core is 20, and on a little core is 32 (the setup and numbers are consistent with the platforms used in the evaluation in Section 5.6).

Initially, the algorithm maps the first line of replicas to the big core, as this decision minimizes the maximum delay after assignment (Figure 5.4a). Similarly, the second line of replicas is assigned to the little core (Figure 5.4b). After all replica lines are distributed among the regular resources, the total delays on both resources are roughly equal (Figure 5.4c).

Next, the algorithm reassigns some of the tasks to accelerators Ψ^{acc}. The latency, $T[\psi], \psi \in \Psi^{\text{acc}}$, is initialized with the minimum delay before the accelerator can receive the first task replica (Lines 10–16). In the example, the minimum delay before the accelerator can receive the first *ifft* task equals the minimum delay of *mf*, which is four units (Figure 5.4c).

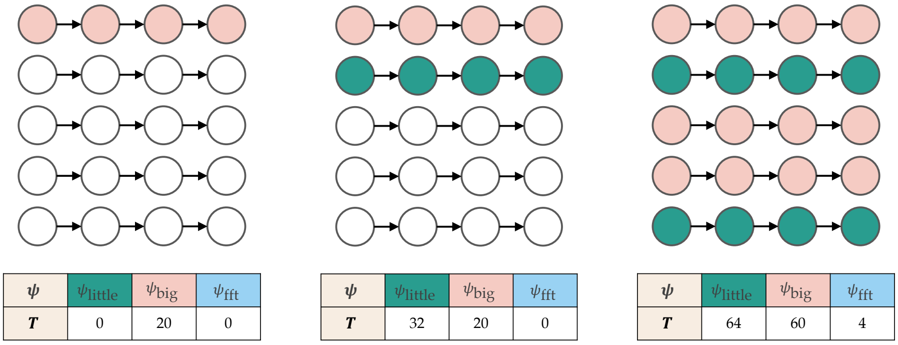
After initialization, the algorithm performs re-balancing of tasks to accelerators (Lines 22–30). For each task π, the algorithm searches for the regular resource ψ_{reg} with the largest total latency that has at least one replica of π (Line 23). Remapping occurs if the accelerator's total time remains below the largest total latency of regular resources (Line 26). If multiple accelerators are available, the task is assigned to the accelerator with the least total latency ψ_{acc} (Line 25).

In the example in Figure 5.4, the first candidate for remapping is selected from the little core, which has the highest total latency, $T[\psi]$. When the task is moved, the values of $T[\psi], \psi \in \mathcal{P}[\Psi]$ are update accordingly (Figure 5.4d). After all *ifft* tasks are migrated (Figure 5.4e),



$\Omega \setminus \pi$	mf	wind	fft/ifft
Ω_{little}	5	3	12
Ω_{big}	4	2	7
Ω_{fft}	-	-	3

Figure 5.3: Sample phase of the baseband receiver with the execution latencies of the tasks. The latency values on the resources are taken from measurements of the LTE PHY benchmark on the Odroid-XU4 and the FFT kernel on a Xilinx Ultra96 FPGA board (see Section 5.6).



ψ	ψ_{little}	ψ_{big}	ψ_{fft}
T	0	20	0

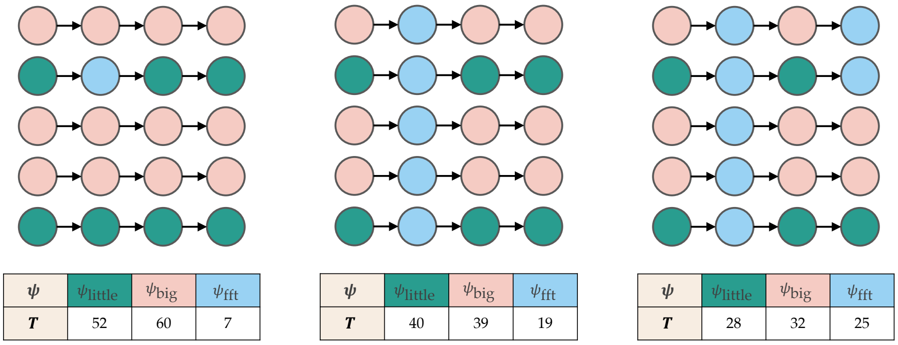
(a) Mapping of the first replica to ψ_{big}

ψ	ψ_{little}	ψ_{big}	ψ_{fft}
T	32	20	0

(b) Mapping of the second replica to ψ_{little}

ψ	ψ_{little}	ψ_{big}	ψ_{fft}
T	64	60	4

(c) Mapping of the remaining replicas and initializing $T[\psi_{fft}]$



ψ	ψ_{little}	ψ_{big}	ψ_{fft}
T	52	60	7

(d) The first *ifft* task is moved from ψ_{little} to ψ_{fft}

ψ	ψ_{little}	ψ_{big}	ψ_{fft}
T	40	39	19

(e) Migrating the remaining *ifft* tasks to the accelerator

ψ	ψ_{little}	ψ_{big}	ψ_{fft}
T	28	32	25

(f) Moving two *fft* tasks to the accelerator; the algorithm is completed

Figure 5.4: Step-by-step visualization of the fast mapping algorithm applied to a phase of the baseband receiver onto a platform with three resources.

the algorithm continues loading the accelerator with *fft* tasks until the stopping condition is reached: the total delay of the accelerator should not exceed the total delay of the regular processors (Figure 5.4f).

Finally, the phase latency is calculated as the maximum total latency across all resources (Line 31), and the energy consumption is calculated as the sum of energy consumptions of each task, according to the generated mapping (Line 32).

Once all phase mappings μ_ϕ have been computed, the final mapping is constructed $\mu \leftarrow \bigcup_{\phi \in A[\Phi]} \mu_\phi$, with the total latency and energy calculated as: $\bullet \leftarrow \sum_{\phi \in A[\Phi]} \bullet_\phi$.

Note that the algorithm attempts to distribute the workload over the resources evenly within each phase, so the platform will process the workload with an approximately constant progress rate. This property enhances predictability during runtime scheduling, particularly in scenarios involving job migrations.

generating pareto-optimal operating points The presented algorithm describes the generation of a single operating point. To generate the set of Pareto-optimal operating points, the algorithm is run for different subsets of the resources $\mathcal{P}[\Psi]$, varying the amount of cores of each particular type Ω_i. In the evaluation presented in Section 5.6, operating points are generated for all $\prod_{i=1}^{|\mathcal{P}(\Omega_i)|} (\mathcal{P}[\omega_i] + 1) - 1$ combinations of processing resources. Supplying the RM with fine-grained operating points can improve system utilization. However, if the platform is too large, this strategy may generate an excessive number of operating points, which, in turn, increases runtime selection overhead. To reduce runtime overhead, distillation heuristics could be applied.

limitations Algorithm 5.1 has several limitations. First, the algorithm does not consider communication delays, which is an intentional choice for the sake of simplicity. Second, the algorithm assumes that the platform has a regular power model; that is, the dynamic power usage of the cores is constant, and the energy consumption of tasks mapped to a specific core is proportional to latency. As a result, the mappings generated by this algorithm are not guaranteed to be Pareto-optimal. However, as demonstrated in Section 5.6, they nevertheless result in effective mappings.

5.5 spatio-temporal mapping reusing previous solutions

As described in Section 5.2, most UE requests must be processed within 2.5 ms, except for URLLC requests, which have a deadline of 0.5 ms. Given such tight deadlines, a resource management algorithm needs to be extremely fast to construct spatio-temporal mappings.

This section introduces an extension to the MMKP-MDF algorithm presented in Section 4.3.1. Recall that in MMKP-MDF, the RM selects the operating point for each job request including those that were previously accepted. This approach considerably increases runtime overhead, critical for the baseband processing application.

To address this issue, a modified version of the algorithm, called *MMKP-MDF-Reuse*, is proposed. This algorithm reduces overhead by reusing previously generated spatio-temporal mappings.

Algorithm 5.2 MMKP-MDF-Reuse: MMKP-MDF with reuse of prior solutions

```
Input: Job requests $\Sigma$, platform $\mathcal{P}$, Pareto-optimal operating points
       $\mathcal{O}_{\text{opt}}^{\sigma[A]}$ for each $\sigma \in \Sigma$, current spatio-temporal mapping $K_{\text{old}}$
Output: New spatio-temporal mapping $K$
 1: $\vec{J} \leftarrow \mathcal{P}[\vec{\omega}] \cdot \max(\sigma[\theta] \mid \sigma \in \Sigma)$        ▷ Initialize the capacities
 2: $K \leftarrow K_{\text{old}}$        ▷ Initialize $K$ with $K_{\text{old}}$ (for new jobs, $M[\sigma] = \perp$)
 3: for all $\sigma \in \Sigma$ do
 4:     $opc[\sigma] \leftarrow \perp$
 5:     for $i \in \{1, \ldots, |K|\}$ do
 6:         if $K[M_i][\sigma] \neq \perp$ then
 7:             $opc[\sigma] \leftarrow K[M_i][\sigma]$        ▷ Reuse operating point
 8:             $\vec{J} \leftarrow \vec{J} - (1 - \sigma[\rho]) \cdot opc[\sigma][\tau] \cdot opc[\sigma][\vec{\omega}]$
 9:             break
10: $\Sigma_{\text{rem}} \leftarrow \{\sigma \in \Sigma \mid opc[\sigma] = \perp\}$
11: while $\Sigma_{\text{rem}} \neq \emptyset$ do
12:     $\sigma^*, CL \leftarrow$ NextJobMDF($\vec{J}$, $\left\{\langle \sigma, \mathcal{O}_{\text{opt}}^{\sigma[A]} \rangle \mid \sigma \in \Sigma_{\text{rem}}\right\}$)
13:     while $\sigma^* \in \Sigma_{\text{rem}}$ do
14:         if $CL = \emptyset$ then
15:             $\Sigma_{\text{rem}} \leftarrow \Sigma_{\text{rem}} \setminus \sigma^*$        ▷ Reject $\sigma^*$
16:             continue
17:         $o^* \leftarrow \text{argmin}_{o \in CL}\{o[\varepsilon]\}$
18:         $opc^* \leftarrow opc; opc^*[\sigma^*] \leftarrow o^*$
19:         $K^* \leftarrow$ ConstructSTM($\Sigma, opc^*, \mathcal{P}$)
20:         if $K^* \neq$ null then
21:             $opc \leftarrow opc^*; K \leftarrow K^*; \Sigma_{\text{rem}} \leftarrow \Sigma_{\text{rem}} \setminus \sigma^*$
22:             $\vec{J} \leftarrow \vec{J} - (1 - \sigma^*[\rho]) \cdot o^*[\tau] \cdot o^*[\vec{\omega}]$
23:         else
24:             $CL \leftarrow CL \setminus o^*$
25: return $K$
```

Algorithm 5.2 outlines the modified approach. The algorithm generally follows the same steps as MMKP-MDF in Algorithm 4.1, with key differences in how it initializes and updates the operating points and spatio-temporal mappings.

First, the spatio-temporal mapping K is initialized with the current mapping K_{old} (Line 2). In Lines 3–9, the algorithm examines K_{old}, populates opc with selected operating points (Line 7), and updates resource-time capacities $\vec{J}$ (Line 8).

In Line 10, the set Σ_{rem} includes the newly arrived jobs (i.e., not previously mapped). These jobs are mapped in the main loop (Line 11), similar to the original algorithm described in Section 4.3.1. There is a difference in how the algorithm handles jobs for which no feasible spatio-temporal mapping is found. If no feasible operating point for a job σ^* is found, the algorithm excludes it from the set of remaining jobs (Line 15), effectively rejecting the job. Unlike MMKP-MDF, this algorithm does not terminate here but continues to map the remaining jobs.

5.6 EVALUATION

A virtual prototype was developed to evaluate the proposed algorithms for generating operating points and spatio-temporal mappings, leveraging domain-specific knowledge of the baseband processing application. Implemented as a plugin[1] to the Mocasin prototyping tool, the prototype integrates a virtual platform, an LTE physical layer uplink receiver application model, and an implementation of the hybrid mapping strategy described in the previous sections.

The prototype was created and validated using measurements obtained from realistic LTE traffic workloads. Using this virtual prototype, simulations were conducted to compare the hybrid strategy with other approaches. In addition to modeling the real platform, the virtual prototype enables evaluation on a wider range of virtual platforms by varying the number of cores or accelerators.

The remainder of this section is organized as follows. Section 5.6.1 details the platform models used. Section 5.6.2 outlines the workload model and LTE traffic trace characteristics. Section 5.6.3 evaluates the Phased-Fast algorithm for generating operating points, while Section 5.6.4 shows how hybrid approaches using the MMKP-MDF-Reuse mapping algorithm enhance performance and energy efficiency in baseband processing.

5.6.1 *Platform Setup*

The virtual prototype was developed and simulated using the Mocasin prototyping tool (described in Section 2.5). This framework allows the definition of virtual platforms and simulation of workload execution in an LTE uplink receiver using various mapping and scheduling strategies.

The platform model is based on the Hardkernel Odroid-XU4 board featuring a heterogeneous Exynos 5422 big.LITTLE chip with four ARM Cortex-A15 and four ARM Cortex-A7 cores, fixed at frequencies

---

[1] `https://github.com/tud-ccc/mocasin-fivegsim`

of 1.8 GHz and 1.5 GHz. Although this platform does not provide sufficient computational power to meet the high demands of general LTE base stations, it serves as a readily available heterogeneous platform for experimentation and extrapolation. Furthermore, the Odroid-XU4 platform has been used in power-efficient implementations of LTE femtocells [34].

For power modeling, each processor core is assumed to operate in one of two modes: *run* and *idle* [217]. If the core is not executing a task, it is placed in idle mode and consumes only static power. Tasks execute only in *run* mode, during which power consumption includes the dynamic component (see Section 2.1.3). Table 5.1 summarizes the power and frequency characteristics of the cores.

Table 5.1: Platform core frequency and power characteristics

Parameter	Cortex-A7	Cortex-A15	FFT Accelerator
Frequency	1500 MHz	1800 MHz	300 MHz
Static power	140.3 mW	214.8 mW	—
Dynamic power	320.2 mW	1319.6 mW	62.5 mW

To obtain a power model of the Odroid-XU4, power consumption measurements were conducted using a ZES Zimmer LMG450 Power Analyzer connected to the DC input with a readout rate of 20 Sa/s. CPU stressors from the `stress-ng` tool[2] were executed on different subsets of the platform's cores to measure dynamic power consumption. Static power was obtained from the SLX Tool Suite's commercial platform models, and an additional 763.3 mW was added to account for the static power consumption of other platform components.

Based on the data for individual cores, the original Odroid platform was extrapolated to create virtual platforms with a larger number of cores, up to eight cores of each type. To evaluate the impact of specialized accelerators, a virtual FFT accelerator was created and optionally included in the platform (up to two instances in the experiments). Performance and power characteristics of the FFT accelerator were derived from measurements on a Xilinx Ultra96 FPGA board. Due to overestimated idle power on the FPGA, the experiments report only dynamic energy consumption.

5.6.2 *Workload Model*

The workload in the virtual prototype is modeled as phased task graphs depicted in Figure 5.2 and derived from the PHY benchmark [178], an open-source LTE uplink receiver implementation. Each graph instance represents the workload for processing one UE, with

---

[2] `https://manpages.ubuntu.com/manpages/artful/man1/stress-ng.1.html`

size and structure depending on parameters such as the number of PRBs, layers, and modulation scheme.

Execution times are derived from measurements of each computational kernel in the PHY benchmark on the Odroid-XU4 for all parameter combinations. These measurements provide a detailed execution time model for each task.

During simulations, instances of the task graphs are dynamically generated based on workload traces that detail the UEs in each subframe. These graphs are then mapped to the available processing elements, and the execution of the workload is simulated.

To simulate realistic workloads, LTE traffic traces from real base stations were used [35]. Each trace captures a sequence of subframes over a specific time period, detailing the UEs processed in each subframe along with their parameters. These traces, recorded over 15 days, include real data with over 1.2 million Radio Network Temporary Identifiers (RNTIs) from different base stations.

Table 5.2: Characteristics of LTE traffic traces used for evaluation. Traces `trace1`-`trace4` are real LTE traffic traces, while `trace0` is constructed by removing heavy UEs.

Trace	Number of UEs	Non-empty Subframes	Average PRBs	Max. PRBs in Subframe
`trace0`	587	543	10.96	48
`trace1`	688	603	17.21	100
`trace2`	1473	1197	24.67	100
`trace3`	1577	1342	31.95	100
`trace4`	2091	1736	54.83	100

For simulations, the first 5000 subframes from four different base stations with varying levels of activity were used. Additionally, a synthetic trace (`trace0`) was constructed by modifying `trace1` to exclude UEs with a large number of PRBs, ensuring all UEs meet their deadlines on the evaluated platforms. The goal of this trace is to separate the energy savings obtained from rejecting UEs from those gained from more efficient resource utilization.

The main characteristics of the traces are summarized in Table 5.2. In all traces, the numbers of antennas and layers are fixed to four. As a result, all UEs have the same number of tasks: 150 (cf. Figure 5.1).

5.6.3 *Generation and Estimation of Operating Points*

In Section 5.4, an algorithm was introduced to generate (approximately Pareto-optimal) operating points for all valid combinations of processor resources in the platform. The algorithm also estimates the latency and energy of each generated mapping.

The proposed method is evaluated in two perspectives. First, the estimated latency and energy values are validated against simulations. Second, the algorithm is compared with other approaches regarding generation overhead and operating point quality.

5.6.3.1 *Energy-Performance Data Validation*

To validate performance and energy estimations, the generated values are compared with simulation results from Mocasin. Using Algorithm 5.1, a total of 12000 operating points were generated. This includes 24 operating points for each of 500 different types of UEs, varying the number of PRBs and the modulation scheme.

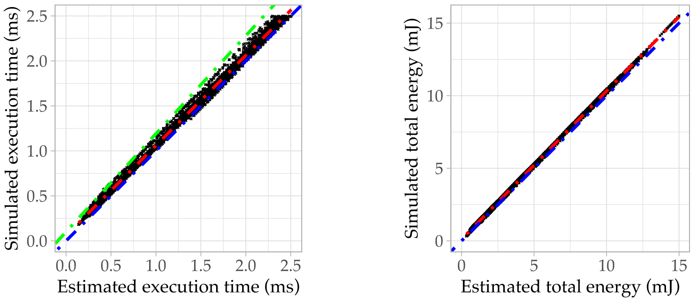


Figure 5.5: Validation of the estimated execution time and energy consumption values obtained with the Phased-Fast algorithm on the Odroid-XU4 platform.

Figure 5.5 shows the validation points on the Odroid-XU4 platform. The plots include only points where the simulated time does not exceed the highest deadline for processing a UE, namely 2.5 ms. The blue line indicates perfect correlation, and the red line shows the result of linear regression. The figure shows that the values obtained in the fast estimation exhibit high accuracy. The energy estimations are nearly ideal, whereas execution time estimations show slightly higher dispersion, which can be attributed to the lack of modeling of communication delays.

Since the Resource Manager (RM) relies on accurate execution time values, an effective execution time is calculated as a linear function of the estimated time: $\tau_{\text{eff}} = a \cdot \tau_{\text{est}} + b$. The RM is simulated on a sample trace with different values of a and b, and the values that minimize the number of UE rejections and deadline misses are selected. Figure 5.5 shows the effective time with green lines, closely aligning with the top edge of the validation points. This indicates that the formula for effective execution time slightly overestimates the actual execution times, thereby improving predictability at runtime.

5.6.3.2 *Comparison with State-of-the-Art Mapping Methods*

The proposed *Phased-Fast* heuristic is compared with other design-time mapping methods. To this end, different mapping algorithms are executed to generate static mappings for UEs with workloads ranging from 5 to 100 PRBs. For each of these UEs, five different mapping heuristics are applied:

- *Phased-Fast*: The proposed domain-specific heuristic.
- *Genetic*: A multi-objective evolutionary algorithm [58].
- *Simulated Annealing*: A heuristic based on [141].
- *Tabu Search*: A heuristic based on [121].
- *Static CFS*: A simple but fast load-balancing heuristic based on a static version of the Linux CFS [142]

The mapping algorithms are evaluated by three measures:

1. *Exploration Time*: The execution time of the mapping algorithm itself, representing the design-time overhead.
2. *Mapping Runtime*: The execution time of the UE in the simulation, measuring the quality of the mapping.
3. *Mapping Energy Consumption*: The energy consumption of the UE in the simulation, representing another measure of quality in this multi-objective setting.

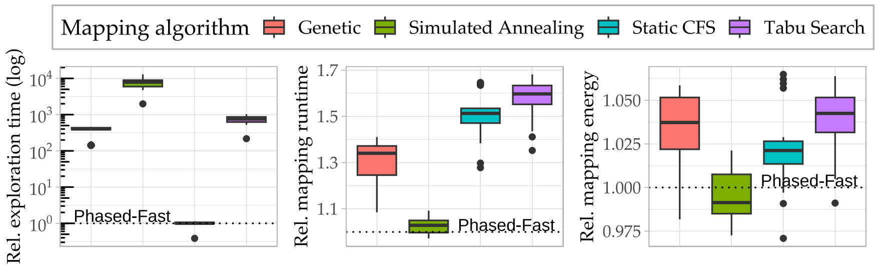


Figure 5.6: Comparison of design-time mapping algorithms for the baseband processing application relative to the Phased-Fast heuristic as baseline.

Figure 5.6 presents the comparison of the design-time mapping heuristics, using Phased-Fast as the baseline (indicated by the dotted line). By showing the relative values, this approach allows for a uniform comparison across different workloads of the UEs. In terms of exploration time, Phased-Fast is the fastest heuristic, taking just 0.36 s in median. Among the other heuristics, only static CFS is not a metaheuristic (i.e., it does not iteratively evaluate mappings). As a result, it is the only one that is comparatively fast, being approximately 0.5 % slower than Phased-Fast in median.

In terms of mapping quality, only the simulated annealing heuristic produces results comparable to the domain-specific Phased-Fast heuristic. While the mappings from Phased-Fast are slightly faster, they also consume slightly more energy. Overall, the mapping quality of these two heuristics is comparable and significantly better than the other state-of-the-art heuristics for this problem. However, the simulated annealing heuristic, which produces comparable results, is approximately four orders of magnitude slower than Phased-Fast. Due to the large parameter space of UEs and the enormous number of tasks in the task graphs, fast mapping heuristics are crucial, even at design time.

5.6.4 *Energy-Efficient Runtime Mapping*

Runtime resource management algorithms are evaluated based on their ability to execute UEs within real-time constraints, energy consumption, and resource utilization. Resource utilization is measured as the number of processing elements required to achieve a given performance level.

In these evaluations, the design-time mapping algorithm is fixed to Phased-Fast, and the following runtime algorithms are compared:

- *Work-Stealing*: A baseline scheduler similar to one used in the PHY benchmark [178]. Tasks are initially assigned to per-core queues of general-purpose cores. Worker threads execute tasks from their own queues, and when empty, they steal tasks from other queues. Accelerators operate without dedicated queues and always steal tasks of a specific type (e.g., FFT) from the general-purpose resources' queues.

- *MMKP-LR*: A spatial mapping algorithm using Lagrangian Relaxation (with up to 1000 iterations) [207]. The spatio-temporal mapping is constructed by applying this algorithm segment by segment.

- *MMKP-MDF*: The fixed-point spatio-temporal mapping algorithm described in Section 4.3.1.

- *MMKP-MDF-Reuse*: The proposed variant of MMKP-MDF that reuses solutions from previous algorithm runs (Section 5.5).

First, all methods are evaluated on two fixed platforms, namely Odroid-XU4 and a virtual platform with two FFT accelerators. Then, resource utilization is evaluated by varying platform sizes. The section concludes with an analysis of runtime overhead.

5.6.4.1 *Evaluation on Fixed Platforms*

Figure 5.7 presents results for the Odroid-XU4 platform using LTE traces from Table 5.2. The left plot shows the percentage of UEs successfully executed within the real-time deadline. In general, all three hybrid approaches outperform the Work-Stealing scheduler, especially under high workloads. On `trace4`, MMKP-MDF-Reuse reaches up to 29.8 % better success rate than Work-Stealing. Compared to the other hybrid schedulers, MMKP-MDF-Reuse performs slightly better on three out of four real LTE traces; however, the differences are minimal.

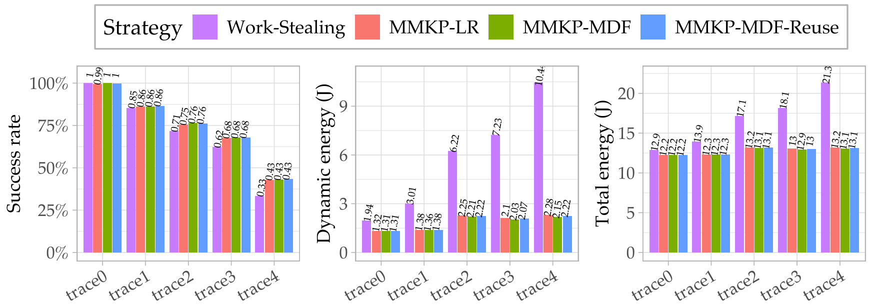


Figure 5.7: Successfully executed UEs and energy efficiency on Odroid-XU4

In terms of energy efficiency, the hybrid approaches exhibit substantial dynamic energy savings over Work-Stealing, ranging from 54.1 % to 78.7 % on the real LTE traces. Even on `trace0`, where all UEs meet their deadlines, MMKP-MDF-Reuse demonstrates a 32.7 % energy savings by selecting energy-efficient configurations. The differences between the hybrid approaches are minor, with MMKP-MDF-Reuse achieving a maximum of 3.2 % better energy efficiency under the most intensive workload.

Total energy savings (right plot of Figure 5.7) follow a similar pattern, with hybrid approaches saving between 4.9 % and 38.5 % compared to Work-Stealing, depending on the workload intensity. These savings highlight the effectiveness of hybrid approaches in leveraging energy-efficient configurations and reducing rejected requests under high workloads.

On a virtual platform augmented with two FFT accelerators (Figure 5.8), the hybrid approaches also deliver comparable results. MMKP-MDF-Reuse accepts 4.3 % fewer requests than the baseline on the least demanding real trace, `trace1`. However, on all other traces, MMKP-MDF-Reuse outperforms the baseline, achieving improvements of up to 24.6 % in Quality of Service (QoS), while reducing dynamic energy consumption by 43.5 % to 75.5 % across the traces. Due to the absence of an accurate static power model for the accelerators, total energy savings are not reported.

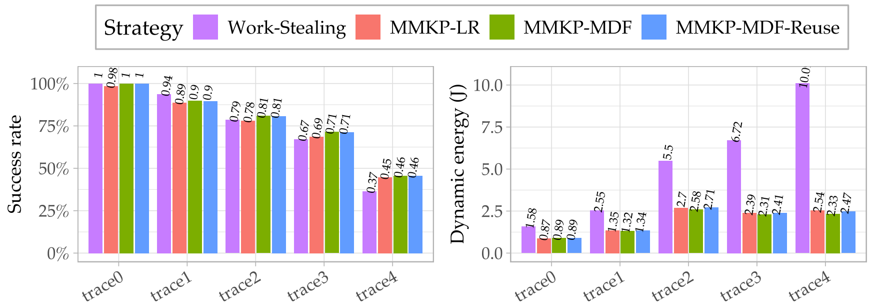


Figure 5.8: Successfully executed UEs and energy efficiency on a virtual platform with accelerators

5.6.4.2 *Varying the Amount of Resources*

The success rates reported in Figures 5.7 and 5.8 are unacceptably low for production-level LTE baseband processing. Even with increased cores in a virtual platform, these configurations fail to meet the demands of modern LTE systems.

In real base stations, QoS requirements stipulate a minimal success rate that must be met. Therefore, comparing the success rate of the approaches is not conclusive. To address this, in this experiment, the success rate is controlled by increasing the number of Cortex-A7 and Cortex-A15 in the virtual platform. To compare the approaches, instances with similar success rates (within a difference of 2 %) were matched and analyzed for relative energy consumption and resource utilization. This method yielded 112 distinct points, grouped in pairs that achieved the same performance with different numbers of resources and energy consumption values.

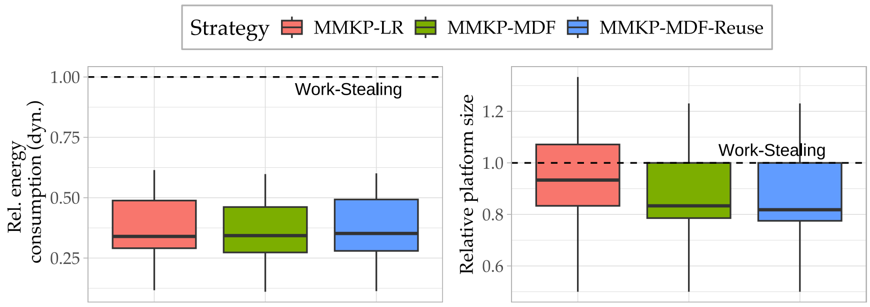


Figure 5.9: Relative dynamic energy consumption and resource utilization of the hybrid mapping approach while controlling for the success rate.

Figure 5.9 compares the hybrid mapping algorithms to Work-Stealing, with the baseline set to one. The MMKP-MDF-Reuse algorithm achieves a median dynamic energy consumption that is only 35.2 % relative to Work-Stealing, with a comparable resource utilization of

82 %. This result demonstrates that the hybrid mapper's ability to deliver the same performance with slightly fewer resources while consuming significantly less dynamic energy.

Compared to MMKP-MDF-Reuse, MMKP-LR and MMKP-MDF achieve slightly less dynamic energy but require 11 % and 1 % more resources, respectively. The performance differences are negligible, but the main advantage of MMKP-MDF-Reuse lies in its reduced runtime overhead, further evaluated in Section 5.6.4.3.

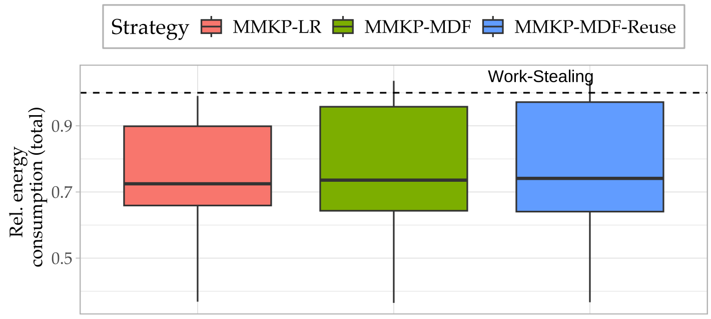


Figure 5.10: Relative total energy consumption of the hybrid mapping approach while controlling for the success rate on virtual platforms without accelerators.

Due to the absence of a static energy model for the accelerators, Figure 5.10 presents results only for virtual platforms without accelerators. Here, MMKP-MDF-Reuse achieves a median total energy consumption of 74.1 % compared to Work-Stealing, underscoring its energy efficiency even without specialized hardware.

5.6.4.3 *Runtime Overhead Analysis*

The hybrid approaches significantly improve the energy efficiency of baseband processing. However, the more complex spatio-temporal mapping incurs additional runtime overhead.

Figure 5.11 presents the average execution time per RM activation on the Odroid-XU4 for all hybrid approaches. These values are not directly comparable with the execution times of the benchmark; they are measured on a prototype implementation written in Python 3 using MOCASIN and executed on a system with a 3.40 GHz Intel Core i7-6700 CPU and 32 GB RAM. However, the relative execution times between the heuristics are comparable since they are all implemented within the same framework.

As shown in the figure, the mapping overhead of MMKP-MDF and MMKP-MDF-Reuse is significantly smaller than that of MMKP-LR, exhibiting at least 99.5 % reduction across all LTE traces. Comparing the MMKP-MDF-based approaches, the proposed MMKP-MDF-Reuse algorithm requires less execution time. Moreover, the speedup grows with workload intensity, from 20.8 % on the lightest `trace0` to 37.7 %

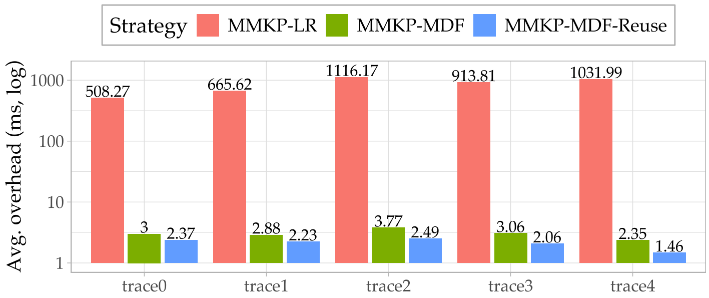


Figure 5.11: Average runtime overhead per RM activation on Odroid-XU4

on the heaviest `trace4`. These results highlight the positive impact of the proposed modifications to the spatio-temporal mapping algorithm.

While the proposed algorithm outperforms other hybrid approaches, the absolute values of the runtime overhead remain too high for a real baseband processing system. As mentioned above, the current prototype is executed with the Python interpreter, causing significant overhead. It is expected that a production-ready mapping algorithm implemented in a low-level programming language could execute 10–200 × faster.

5.7 SYNOPSIS

This chapter demonstrates the potential of HAM methodologies to address the demands for flexibility and energy efficiency in baseband processing. By leveraging the computational structure of task graphs and refining the scheduling methodology, the proposed domain-specific approach effectively scales to the complexities of baseband processing kernels.

The evaluations highlight significant improvements. At design time, the Phased-Fast mapping algorithm generates mappings of comparable quality to those obtained by the state-of-the-art simulated annealing algorithm, but those mappings were generated almost four orders of magnitude faster. At runtime, hybrid approaches, including the proposed MMKP-MDF-Reuse algorithm, consume about a third of the dynamic energy compared to a work-stealing scheduler, while maintaining the same success rates on equivalent hardware resources. Furthermore, MMKP-MDF-Reuse achieves a runtime overhead reduction of up to 37.7 % compared to other hybrid mapping strategies.

This work serves as a proof-of-concept, showing the applicability and benefits of HAM methodologies in baseband processing. To fully demonstrate its viability, future research could focus on further optimizing compile-time Pareto point selection to reduce runtime overhead, exploring integration of voltage-frequency scaling for additional energy saving, and evaluating the approach on more advanced 5G and beyond baseband workloads, as well as higher-performing platforms.

6

EXTENDING KAHN PROCESS NETWORKS WITH ADAPTIVITY

This chapter shifts the focus from adaptivity at the resource management level to adaptivity at the application level. To fully utilize available resources, applications have to adapt their execution to align with the decisions made by the RM. General HAM methodologies typically focus on thread-to-core pinning, leaving applications agnostic to the concrete mapping. Additionally, many applications lack mechanisms for adaptivity, limiting their ability to make internal adjustments and leading to suboptimal resource utilization.

As discussed in Section 2.3.1, Kahn Process Network (KPN) is a widely used dataflow Model of Computation (MoC) in embedded systems due to its expressiveness and deterministic execution guarantees. However, KPNs have limitations in adaptivity. Their deterministic properties, based on Kahn-MacQueen execution semantics, restrict scheduling flexibility and enforce rigid communication patterns. Furthermore, KPNs are ill-suited to expose data-level parallelism, as their application topology is statically defined and cannot be changed at runtime.

To address these issues, this chapter proposes an extension that introduces implicit parallelism into the KPN execution model while preserving its formal semantics. By relaxing the Kahn-MacQueen execution semantics and reverting to the original semantics of KPNs, runtime adaptivity can be enhanced without sacrificing determinism. This proposed extension, termed *Adaptive Process Network* (APN), introduces malleable runtime adaptivity and implicit parallelism, improving application scalability and resource utilization.

The chapter begins with a motivational example in Section 6.1, illustrating the adaptivity limitations of KPN applications. Section 6.2 introduces the application model of APNs. Section 6.3 presents the DPM library, which provides runtime support for APN applications by managing configurations and enabling communication with the RM. Section 6.4 evaluates the benefits of APNs in improving execution on HMAs. The chapter ends with a synopsis in Section 6.5.

A Note on Publications and Contributions

An early version of the implicit data-parallel extension to KPNs, including the motivational example, application model (featuring parallel processes), figures, and results, was previously published in Khasanov, Goens, and Castrillon, “Implicit Data-Parallelism in Kahn

Process Networks: Bridging the MacQueen Gap," 2018 [99]. Since then, the application model has been extended to include parallel regions, and the DPM library was developed. While the main ideas, the APN application model, and the DPM runtime library were conceptualized and developed by the author of this thesis, the implementation of the DPM library was carried out by Fabius Mayer-Uhma and Dylan Gageot.

6.1 LIMITATIONS OF KPNS: A MOTIVATIONAL EXAMPLE

Consider a KPN application that calculates the Mandelbrot set, depicted in Figure 6.1. In this process network, a source process (`SRC`) divides the complex plane into lines and sends these lines to four worker processes ($\texttt{W}^1$-$\texttt{W}^4$). Each worker calculates the convergence or divergence of points in its assigned lines and sends the results to a sink process (`SNK`), which reassembles the lines to output the complete Mandelbrot set.

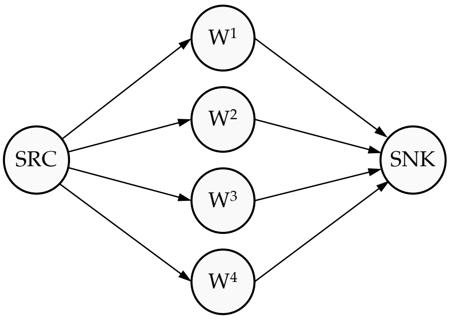


Figure 6.1: KPN application calculating the Mandelbrot Set with four worker processes.

While this example showcases how the KPN model can express parallelism, it also highlights significant limitations regarding adaptivity.

The first limitation is that KPNs lack implicit data-level parallelism semantics. To achieve data parallelism, application developers have to explicitly define the parallel structure of the process network at compile-time. In the Mandelbrot example, the four worker processes ($\texttt{W}^1$-$\texttt{W}^4$) represent this explicit parallelism.

The second limitation arises from the deterministic execution of each process. In the example, the source process (`SRC`) operates according to a deterministic program that specifies the sequence of write events to each of its output FIFO channels. While the workload is usually distributed evenly, this can lead to load imbalances if workload units differ in computational complexity or when worker processes run on heterogeneous processing elements.

Similarly, the sink (`SNK`), which reads the results from the worker processes, follows a deterministic sequence of read events. This constraint is more stringent due to the Kahn-MacQueen blocking-read semantics. Under these semantics, a process with multiple input channels must read in a predetermined static order. If the first channel in this order does not have data available, the process blocks, even if other channels have data ready to be processed. This behavior leads to suboptimal execution and poor load balancing among processes.

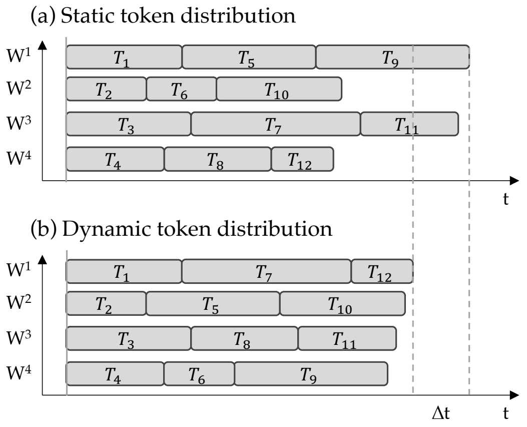


Figure 6.2: Gantt charts illustrating (a) suboptimal scheduling in the static KPN model and (b) improved scheduling with dynamic workload distribution

These limitations are illustrated in Figure 6.2, which shows a Gantt chart of the execution of the four workers calculating 12 different tokens (lines). In the static KPN model (Figure 6.2a), the deterministic nature and fixed workload distribution lead to poor load balanced among workers. Some workers may finish their assigned tasks earlier and remain idle while other are still processing. In contrast, Figure 6.2b shows a different workload distribution achieved by assigning tasks to workers as they become available. Such a strategy allows for better load balancing and faster execution.

These limitations have several consequences:

- *Reduced Portability*: Programmers have to generate a specific network topology for each target platform, considering the number of available processors and their characteristics.
- *Rigidity in Resource Utilization*: Explicit network topologies are rigid (Section 3.4) and cannot adapt to variations in available resources at runtime. If the number of processing elements changes because of other applications, the static process network cannot adjust.
- *Inefficient Execution on Heterogeneous Platforms*: Even in homogeneous systems, variation in token processing times can cause imbalances, requiring dynamic workload distribution. On HMAs, where cores differ in performance and energy characteristics, this need extends to a broader class of applications, including those with uniform processing times.

- *Limited Compiler Optimization*: The inability of the KPN model to capture semantic information about data-parallel execution makes it challenging to design a holistic approach for optimizing parallelism. As a result, tuning data parallelism cannot be integrated into the compiler and falls on the programmer's responsibility.

Given that data parallelism is a common type of parallelism, addressing these issues is crucial for leveraging the full capabilities of modern systems.

6.2 ADAPTIVE PROCESS NETWORK

To overcome the limitations of KPNs, the *Adaptive Process Network* (APN) model is proposed to introduce implicit data-level parallelism and enhance adaptivity while preserving deterministic execution of KPNs. The APN model consists of several components: *parallel regions*, which are special nodes in the process network that encapsulate a subnetwork of process and can be replicated at runtime; and *parallel channels*, which facilitate communication between the replicated parallel regions and the rest of the network. By employing a consistent strategy for distributing and collecting tokens, the APN model achieves deterministic semantics. Furthermore, the dynamic distribution strategy offers two key adaptivity features: adaptivity at workload distribution and malleability.

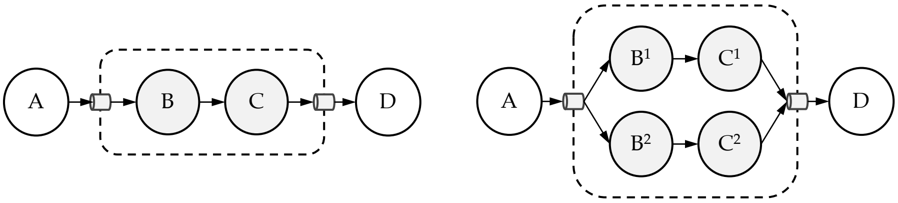
6.2.1 *Parallel Regions*

The Adaptive Process Network (APN) model extends the standard KPN by introducing a new type of node called a *parallel region*. From an external viewpoint, a parallel region functions like a regular KPN process: it has input and output ports to which channels are connected, and the entire region reads tokens from input channels and produces tokens on output channels.

Internally, however, a parallel region contains its own subnetwork of processes and communication channels. This internal subnetwork comprises processes connected by channels that are classified as either *intra-region channels* — connecting processes within the region — or *boundary channels* — connecting internal processes to the region's input and output ports.

At runtime, a parallel region can be replicated, allowing the workload to be distributed among multiple instances of the region. Each replica operates independently, with no communication channels between processes in different replicas. This replication enables implicit data-level parallelism in the process network.

Figure 6.3 illustrates an example of an APN application with a parallel region. In Figure 6.3a, the application model shows a parallel



(a) An application model with a parallel region consisting of two processes.

(b) A runtime topology where the region is duplicated into two replicas.

Figure 6.3: An APN application with a parallel region consisting of two processes: an application model and a runtime topology.

region containing two processes, while in Figure 6.3b, the runtime topology depicts the parallel region duplicated into two replicas that can execute in parallel.

Not all processes can be included within a parallel region. The execution of regular KPN processes may depend on the data read from their input channels, and their communication patterns may vary during execution. Moreover, KPN processes can be *stateful*, meaning that the processing of a token may depend on the processing of previous tokens. Including such processes in a replicated parallel region could lead to incorrect behavior due to unintended dependencies across replicas.

Therefore, specific requirements must be met for a process to be included within a parallel region:

1. *Constant Input and Output Rates*: The process must behave as an actor with constant rates, similar to actors in dataflow models with firings [107]. This means that the process consumes and produces a fixed number of tokens on each of its input and output channels during each execution. The exact number of tokens does not need to be the same across different channels but must remain constant for each channel.

2. *Statelessness*: The process must be stateless, implying that its output depends only on the current input tokens and not on any internal state or previous executions. This ensures that there are no dependencies between different iterations of the process, allowing replicas to process data concurrently without interference.

Processes meeting these conditions can be safely replicated and executed in parallel within a parallel region without causing inconsistencies or violating the deterministic semantics. Actors in Synchronous Dataflow (SDF) [109] naturally meet these requirements, as they are stateless and have constant production and consumption rates. While these conditions are sufficient, they are not strictly necessary. The approach could, in principle, be extended to accommodate actors with cyclic rates, such as those in Cyclo-Static Dataflow (CSDF) [21].

With above requirements, the subnetwork within a parallel region becomes an open SDF network. For correct operation, the SDF subnetwork must be *consistent*. This means it can execute repeatedly without accumulating unbounded tokens on any channel or causing deadlocks [108].

Since the processes in a parallel region may have different firing rates on different channels, it is necessary to determine the minimum workload unit that is independent from other workload units. Every consistent SDF graph can be transformed into a HSDF graph, in which actors consume and produce one token on each input and output channel at each firing [186]. During this transformation, a *repetition vector* $\vec{q} = [q_1, q_2, \ldots, q_n]^\top$ is calculated, where each element q_i specifies the number of times actor π_i must fire in one full iteration of the subnetwork, so that the network returns to its initial state. Using this repetition vector, it is possible to determine the number of tokens on each input and output boundary channel that belongs to a minimum workload unit assignable to a single replica of the parallel region.

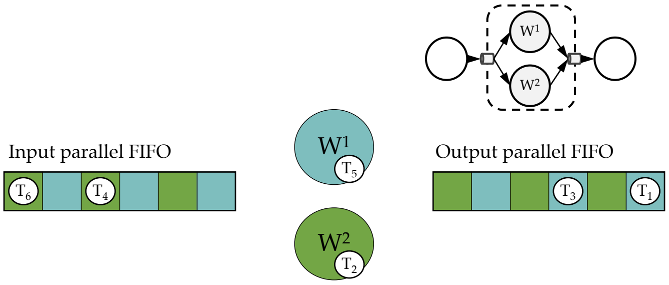
6.2.2 *Parallel Channels and Workload Distribution*

When a parallel region is replicated into multiple instances, these instances need to be connected with the predecessors and successors of the region. In the standard KPN model, each FIFO channel connects to a single process on each side. However, with multiple instances of a parallel region, the channels connecting to these instances involve multiple processes on one side, requiring a different approach.

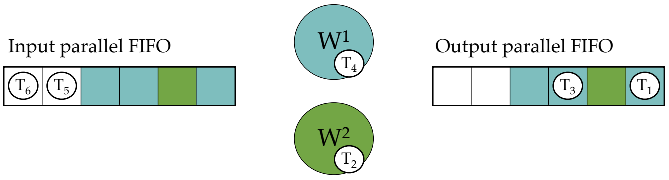
To handle this, *split* and *interleave* primitives are introduced. Split primitives distribute tokens among the instances of the parallel region, while interleave gather tokens from all instances back into a single channel. The split-interleave functionality can be implemented in several ways: as additional processes, within the predecessors and successors of the parallel region, or as part of the channels themselves.

In the APN execution model, special *parallel channels* are introduced, which implement the split and interleave primitives. These parallel channels are specialized FIFO channels that allow tokens to be read or written based on their identifier, potentially in *out-of-order* fashion.

To maintain the original KPN semantics and preserve token order, coordinated access to parallel channels at the boundaries of parallel regions is essential. In the APN execution model, a shared *region manager* keeps track of the assignment of workload units to replicas. When a process in a parallel replica is ready to process the next workload unit, it requests its identifier from the region manager using its replica identifier. The process then determines which tokens to read from or write to in the parallel channels based on this identifier, identifying their exact positions in the channel buffers. This mechanism ensures



(a) Static token distribution



(b) Dynamic token distribution

Figure 6.4: Distribution strategies for split-interleave primitives

that the consumer of the output channel reads tokens in the correct order, as if only a single replica were executing, thus preserving the deterministic behavior of the application.

Figure 6.4 illustrates static and dynamic strategies to distribute tokens among replicas. Figure 6.4a depicts a static assignment of tokens to workers in a round robin fashion. In this example, tokens T_1 and T_3 have already been processed by worker $\mathtt{W}^1$, which is now processing T_5. In contrast, $\mathtt{W}^2$ is slower and still executing T_2, causing a load imbalance.

Figure 6.4b illustrates a dynamic distribution strategy. In this case, there is no static assignment of tokens. Instead, tokens are assigned to replicas dynamically as they become available. Since $\mathtt{W}^2$ takes a long time executing T_2, tokens T_3 and T_4 are assigned dynamically to $\mathtt{W}^1$. This allows for better load balancing, as faster workers can process more tokens, leading to improved overall performance.

The dynamic distribution of tokens to replicas introduces a key adaptivity feature in the APN model, which is particularly beneficial on heterogeneous hardware platforms like HMAs or when some cores are shared with other applications. This adaptivity is also advantageous when the workload depends on the data, causing significant variation in execution time between firings.

In both static and dynamic distribution scenarios, the interleave primitive maintains the correct ordering of tokens in the output. This demonstrates that, despite the relaxation introduced by parallel channels, the control provided by the region manager guarantees the deterministic execution of the entire APN application. This property is valuable, as many assumptions used in the many analysis methodologies (see Section 2.5 and [42, 147, 150]) implicitly rely on determinism. For example, it enables compatibility with trace-based simulation approaches, such as described in Section 2.3.3.

6.2.3 *Malleability*

Another important adaptivity feature offered by APN with dynamic distribution is *malleability* (see Section 3.4 and [63]). In the context of parallel applications, a regular KPN with static data parallelism is considered *rigid*, as the number of parallel processes is specified by the user at compile time. An APN application with static distribution enhances flexibility by allowing the system to decide the parallelization degree at startup, making the application *moldable*. However, static distribution strategy does not allow changing the number of replicas during execution.

APN application with a dynamic distribution policy removes this restriction, enabling malleability. Since tokens are not pre-assigned to replicas in advance, creating and terminating replicas becomes straightforward. New replicas can join and begin processing workload units from the input channels, while terminating replicas can complete their current workload units and stop requesting new ones.

To ensure stability and correctness, reconfigurations must be performed safely. In addition to maintaining the distribution strategy, the region manager is responsible for managing the replica instances. When new instances need to be created, the region manager generates runtime instances of the processes and intra-region channels according to the template network defined in the parallel region and connects the parallel channels to the newly created processes. The threads executing the processes in the new replicas are started, and they begin pulling workload units.

When a replica needs to be terminated, abrupt termination is undesirable, as processes may be in the middle of processing a workload unit, leading to data loss and inconsistent states. Instead, the region manager marks the replica for deletion. Marked replicas are allowed to complete their current workload units but will not receive new ones. When the processes in this replica attempt to request a new workload unit, they receive a terminating signal, exit their processing loops, and terminate. Once all processes in the replica have finished, the replica is removed from the runtime application graph.

This approach ensures that the application can dynamically adjust its parallelization degree while maintaining correct output. In Section 6.3, the runtime library providing an interface to the RM is introduced. This library allows dynamic changes to the number of replicas, thereby making the application malleable.

6.3 DYNAMIC PROCESS MANAGER (DPM) LIBRARY

The *Dynamic Process Manager* (DPM) is a C++ runtime library designed for managing and executing Adaptive Process Networks (APNs). It provides essential functions enabling the creation, management, and execution of APN applications.

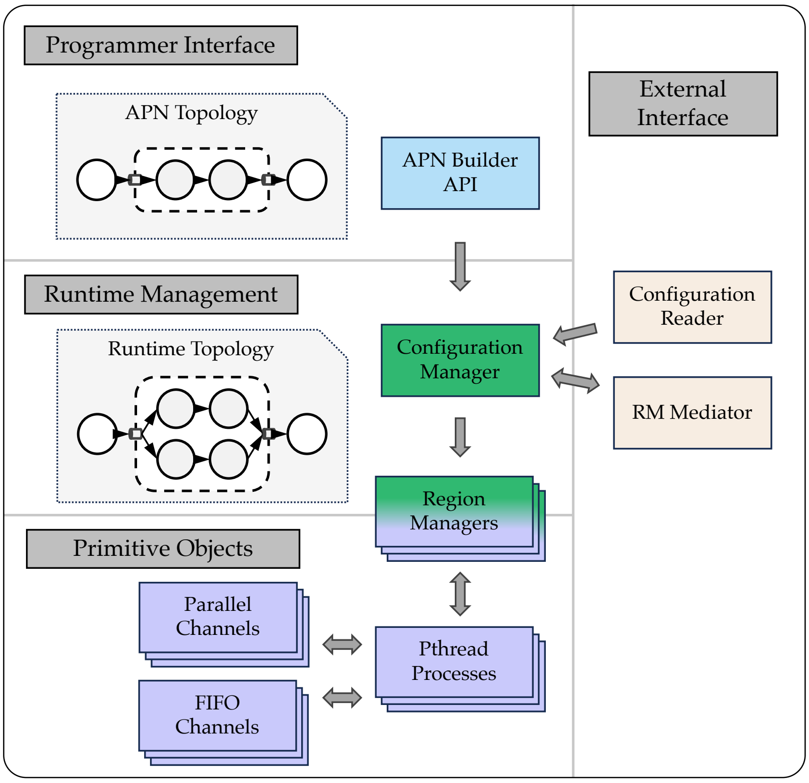


Figure 6.5: Overview of the Dynamic Process Manager (DPM) library

Figure 6.5 provides an overview of the DPM library's component and functionality. First, it offers a C++ Application Programming Interface (API) for developers to define an APN application, specifying their topologies and process body implementations. This API is detailed in Section 6.3.1.

Second, DPM provides the primitives needed to build and manage the runtime topology of the application. This includes implementations of FIFO and parallel channels, process wrappers using pthreads[1], and region managers to handle parallel regions. The runtime topology construction is described in Section 6.3.2.

Finally, DPM acts as a runtime manager, handling application configurations, which specify the number of replica and the mapping of processes to cores. A central *configuration manager* coordinates with region managers and maintains information about available configurations. It also interacts with an external Resource Manager (RM)

---

[1] `https://man7.org/linux/man-pages/man7/pthreads.7.html`

to align application configurations with the RM decisions. This is described in Section 6.3.3.

6.3.1 *Programmer Interface*

DPM provides a C++ API for defining and implementing APN applications, including both their topologies and the logic of individual processes. Listing 6.1 shows an example implementing the topology of an APN application that calculates the Mandelbrot set.

Listing 6.1: Building APN topology with DPM

```
 1 #include <Dpm/Manager.h>
 2 using namespace Dppm;
 3 
 4 // Process body definitions are hidden
 5 
 6 int main(int argc, char* argv[]) {
 7   auto manager = Manager::GetInstance();
 8   auto testApp = manager->CreateApplication();
 9 
10   // Construct APN topology
11   auto mainRegion = testApp->MainRegion();
12 
13   auto in1 = mainRegion->AddChannel<int>();
14   auto out1 = mainRegion->AddChannel<std::array<int, WIDTH>>();
15 
16   mainRegion->AddKpnProcess("Init", Init, {}, {in1});
17 
18   auto parallelRegion = mainRegion->AddParallelRegion(
19       "pr1", InputChannels{in1}, OutputChannels{out1});
20   parallelRegion->AddSdfProcess("CalcLines", CalcLines,
21       {parallelRegion->InputChannel<0>()},
22       {parallelRegion->OutputChannel<0>()});
23 
24   mainRegion->AddKpnProcess("Display", Display, {out1}, {});
25 
26   // Start APN
27   manager->RunApplication(testApp.get());
28 
29   return 0;
30 }
```

In this code, Lines 7–8 retrieve the DPM manager singleton and create a new application. The application topology construction begins with a main region (Line 11), where communication channels (`in1` and `out1`) are created (Lines 13–14). These channels carry tokens of type `int` and `std::array<int, WIDTH>`, respectively.

In Line 16, the KPN process `Init` is added to the main region via `AddKpnProcess()`. The arguments include the process name, a pointer to the process function, and lists of input and output channels. Similar steps apply to the `Display` process (Line 24).

Line 18 creates a parallel region `pr1` with one input port (connected to `in1`) and one output port (connected to `out1`). Inside the region, an SDF process `CalcLines` is added and connected to the region's boundary channels via `InputChannel<0>()` and `OutputChannel<0>()` (Line 20).

Listing 6.2: KPN process definition with DPM

```
1 void Init(Application* app, InputRTChannels<>, OutputRTChannels<
      int> out) {
2   auto [outChannel] = out;
3   for (int i = 0; i < HEIGHT; i++) {
4     outChannel->Push(i);
5   }
6 }
```

In addition to defining the topology, the programmer must also provide the logic of the processes. Listing 6.2 shows an example of a KPN process definition. The KPN process signature includes a pointer to the application, and wrappers for input and output channels. Channel wrappers (`InputRTChannels<>` and `OutputRTChannels<int>`) use templates to specify the types of connected channels.

In this example, `Init` has no input channels and one output channel of type `int`. Within the function body, Line 2 extracts the runtime output channel from the wrapper, and the function writes a sequence of integers into the output channel by calling `Push`.

Listing 6.3: SDF process definition with DPM

```
std::tuple<std::array<int, WIDTH>> CalcLines(int y) {
  std::array<int, WIDTH> retVal;
  for (int x = 0; x < WIDTH; x++) {
    retVal[x] = calc(x, y);
  }
  return {retVal};
}
```

Listing 6.3 shows the definition of an SDF process, `CalcLines`. In SDF processes, arguments specify input tokens, while return values (in a `std::tuple`) define output tokens. This structure is different from KPN processes, where the function signature includes runtime channels, and the process explicitly pulls and pushes token from and to these channels. SDF processes are restricted to a single firing operation per call, ensuring stateless behavior and constant rates as discussed in Section 6.2.1. Consequently, explicit pull and push operations are not required in the process code; instead, these operations are handled by the SDF wrapper functions.

The current DPM implementation limits SDF processes to reading and writing a single token per firing, making them effectively HSDF actors. Future extensions may support SDF with different firing rates.

This example demonstrates how to define APN applications using the DPM library. By relying on C++ templates and compile-time type checking, the DPM library ensures that channel types and process signatures match. Programmers only need to specify the topology and process logic, while DPM handles creating the runtime topology and managing application configurations.

6.3.2 *Runtime Topology*

Once the APN application is built, DPM's manager creates the runtime application graph. For each regular process in the model, a corresponding runtime process is created. For each channel connecting regular processes, a FIFO channel is instantiated.

For each parallel region, DPM creates a region manager and spawns the required number of replica instances according to the chosen configuration (see Section 6.3.3). Each replica instance includes runtime processes and FIFO channels corresponding to the subnetwork defined within the parallel region. At the boundaries of parallel regions, DPM creates parallel channels (as discussed in Section 6.2.2) to connect replicas to external processes.

When the application starts executing, region managers and parallel channels coordinate workload distribution across replicas. With the dynamic distribution strategy, the application efficiently balances load, adapts to variations in performance, and preserves deterministic semantics.

6.3.3 *Configuration Management*

DPM also manages application's runtime configurations, including the number of replicas and process-to-core assignments. These configurations are provided in a YAML file, as illustrated in Listing 6.4.

In the `mapping_template` of the YAML file, the programmer lists the top-level processes and the parallel regions with their internal processes. The `mappings` section then lists distinct runtime configurations. Each mapping entry specifies (1) the process-to-core assignments for the non-parallel processes (under `processes`), and (2) the number of replicas and the process-to-core assignments for each replica of the parallel regions (under `par_regions`).

Listing 6.4: Sample YAML configuration file

```yaml
application: mandelbrot
platform: odroid
type: dpm
mapping_template:
  processes: [Init, Display]
  par_regions:
    pr1: [CalcLines]
  metadata: [execution_time, energy]
mappings:
  - name: mapping1
    processes: [ARM00, ARM00]
    par_regions:
      pr1:
        - [ARM00]
    metadata: [100, 250]
  - name: mapping2
    processes: [ARM00, ARM00]
    par_regions:
      pr1:
        - [ARM01]
        - [ARM02]
        - [ARM03]
    metadata: [60, 260]
  - name: mapping3
    processes: [ARM00, ARM00]
    par_regions:
      pr1:
        - [ARM00]
        - [ARM01]
        - [ARM02]
        - [ARM03]
        - [ARM04]
        - [ARM05]
        - [ARM06]
        - [ARM07]
    metadata: [20, 300]
```

In the example, `mapping1` allocates one replica of the `pr1` region mapped onto core `ARM00`. `mapping2` defines three replicas for `pr1`, each assigned to different cores (`ARM01`, `ARM02`, `ARM03`). `mapping3` increases this further to eight replicas for `pr1`, achieving the maximum resource utilization on the Odroid-XU4 platform.

The configuration file may also provide extra metadata such as execution times and energy consumption estimations for each mapping. Although DPM itself does not directly utilize these metadata fields, it can forward them to an external Resource Manager (RM) to guide runtime decisions.

Users can specify a particular mapping from this file at startup. If connected to an RM, DPM passes the configuration data through an *RM Mediator* component (depicted in Figure 6.5). When the RM decides to switch mappings (i.e., select a different operating point), it sends a command back to DPM, which then applies the corresponding reconfiguration to the running application. In this work, DPM integrates with HARP, the RM introduced in the next Chapter 7, demonstrating how both components cooperate to enable adaptive, energy-efficient, and performance-optimized APN applications at runtime.

6.4 EVALUATION

The effectiveness of the proposed APN application model is evaluated with a focus on its impact on performance and adaptivity across varying execution conditions.

6.4.1 *Experimental Setup*

All experiments were conducted on a Hardkernel Odroid-XU4 board featuring a Samsung Exynos 5422 big.LITTLE chip with four Cortex-A15 (big) cores and four Cortex-A7 (little) cores, running at 2.0 GHz and 1.4 GHz, respectively, with 2 GB of LPDDR3 RAM.

The benchmark used is the calculation of the Mandelbrot set, as described in Figure 6.1 and Listing 6.1. It consists of two KPN processes (source and sink) and a parallel region containing a single SDF process. The processes in the parallel region calculate points in an iterative manner, where the number of iterations depends on the point itself, i.e., data-dependent. This characteristic makes the benchmark suitable for evaluation data parallelism more thoroughly, since the execution time per point varies. Figure 6.6 shows the number of iterations needed for each point, visually revealing the Mandelbrot set structure.

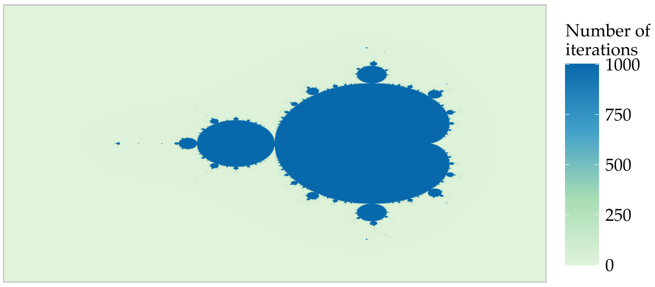


Figure 6.6: Iterations per point in the Mandelbrot set

For comparison, two versions of the application are used:

- *KPN Version*: Implemented with manually introduced data parallelism and hard-coded split/interleave functionality in the processes, using static workload distribution. Different degrees of parallelization are achieved by creating separate versions with different number of replicas.

- *APN Version*: Implemented using parallel channels and the dynamic workload distribution strategy.

Two problem sizes are used: 4000 × 3000 for initial experiments with static configurations (Section 6.4.2) and 8000 × 6000 for adaptivity-related experiments (Section 6.4.3). In all scenarios, parallel workers are assigned to distinct cores, while source and sink remain unpinned.

6.4.2 *Performance Scalability with Parallelization*

The first set of experiments evaluates how both implementations scale on the Odroid-XU4 when varying the number and type (big vs. little) of parallel workers.

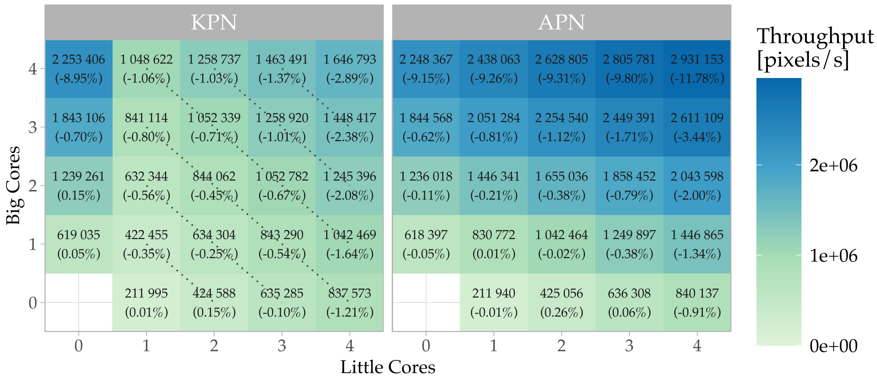


Figure 6.7: Throughput on Odroid-XU4 for different configurations of little and big cores as parallel workers. The approximated errors are shown in parentheses.

Figure 6.7 presents the throughput (in pixels/second) for all combinations of big and little cores assigned to parallel workers. In homogeneous scenarios, where only big or only little cores are used, both the KPN and APN versions achieve similar throughput. However, when mixing big and little cores, the APN version outperforms the static KPN version by up to 78.0 %.

A clear illustration of the static approach's limitations arises when comparing a configuration four big workers to a configuration with four big and four little workers. Despite using fewer total cores, the four-big-worker configuration outperforms the mixed-core configuration by 36.8 %. In the static approach, slower workers become stragglers, reducing the entire system's throughput.

To quantify this observation, consider the following approximations for total throughput. For the static KPN:

$$TH_{\text{total}}^{\text{stat}} \approx \begin{cases} TH_{\text{little}} \times (b+l) & \text{if } l > 0 \\ TH_{\text{big}} \times b & \text{if } l = 0 \end{cases} \tag{6.1}$$

Here, l and b are the number of little and big workers, respectively.

For the dynamic version, in which all cores are utilized fully:

$$TH_{\text{total}}^{\text{dyn}} \approx TH_{\text{little}} \times l + TH_{\text{big}} \times b \tag{6.2}$$

In Figure 6.7, the relative error of this approximation is shown in parentheses, and dashed lines indicate equally-performing configurations for the static KPN. For configurations in which the big cluster is not fully utilized, the relative error does not exceed 3.44 %. However, when the big cluster is fully utilized, the relative error increases to 11.79 %, largely due to thermal constraints preventing the big cluster from sustaining high frequencies.

These results demonstrate that in the static approach, a rigid execution model allows slower workers to impede overall performance, while the dynamic approach offered by APN mitigates this issue by adapting the workload distribution at runtime.

6.4.3 *Runtime Adaptivity*

After examining static configurations, the next experiments focus on runtime adaptivity, including both workload distribution adaptivity and malleability.

6.4.3.1 *Workload Distribution Adaptivity*

In this scenario, the benchmark runs on four little cores. At runtime, a second application (a sorting task) starts after 15 s, causing contention by sharing one of the little cores. Another instance of this sorting application starts 17 s later.

Figure 6.8 shows throughput over time. For reference, the dotted lines depict performance without contention. The KPN version suffers a dramatic throughput drop as soon as contention arises, because one worker becomes a severe bottleneck that slows all the other workers. In contrast, the APN version only sees a slight reduction in performance. Its dynamic distribution lets faster workers handle more tokens, mitigating the impact of the slowed-down worker. Note that the high throughput at the beginning and the end of the execution corresponds to computing regions far from the Mandelbrot set (see Figure 6.6).

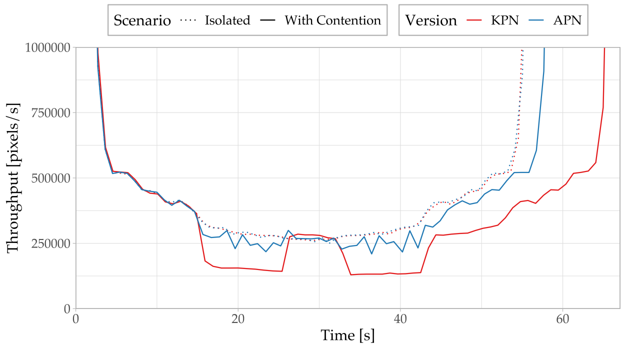


Figure 6.8: Effect of contention on performance. Dotted lines indicate performance without contention.

6.4.3.2 *Malleability: Changing Parallelization at Runtime*

Finally, the ability of the APN application to adjust its parallelization degree during execution is tested. This feature is not available in the KPN version.

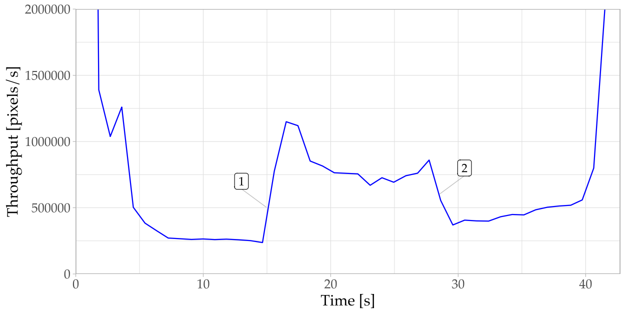


Figure 6.9: Dynamic reallocation of resources at runtime

Figure 6.9 shows the evolution of throughput over time. The execution begins with two little workers. At 15 s, four big workers are added (marked by ①). 13 s later, three big and one little worker are removed (②).

The plot illustrates how the APN application seamlessly adapts to resource availability changes. It fully utilizes resources when present and scaling efficiently when they are no longer available.

6.5 SYNOPSIS

This chapter introduced the Adaptive Process Network (APN) application model, an extension to KPNs that enables adaptive data-parallelism while maintaining deterministic execution. The core contribution is the introduction of *parallel regions*, which encapsulate process subnetworks. These regions can be replicated at runtime, and their communication with the rest of the application is handled through *parallel channels*, which allow out-of-order token access by parallel replicas. Deterministic semantics are preserved through split-interleave strategies, carefully controlled by region managers.

Additionally, the DPM library was presented. This library provides a C++ API for programmers to define APN applications; it constructs the runtime topology using built-in process wrappers, channels, and region managers; and it manages application configurations.

The evaluation demonstrated that this approach can improve performance by up to 78.0 % on a heterogeneous Odroid-XU4 platform. The dynamic nature of the APN model ensures efficient resource utilization, avoiding bottlenecks caused by straggling processes and overcoming the rigidity of standard KPN semantics.

By extending adaptivity to the application level, the APN model complements resource-level adaptivity. The next Chapter 7 introduces HARP, an RM solution that integrates these capabilities to achieve coordinated adaptivity at both the resource management and application levels.

7

COORDINATING ADAPTIVITY IN GENERAL-PURPOSE ENVIRONMENTS

In the previous chapters, the focus primarily centered on embedded scenarios, reflecting the early adoption of Heterogeneous Multi-core Architectures (HMAs) in that domain. Today, however, HMAs have entered the domain of powerful desktop and server computers, necessitating research into resource management for modern operating systems. Although OS schedulers have begun adapting to heterogenous CPUs, as discussed in Section 3.2.2, they typically rely on simple cost-based thread allocation strategies and fail to fully utilize the potential of this hardware.

This chapter introduces HARP (Heterogeneity-aware Adaptive Resource Partitioning), a Linux-integrated resource management framework designed to optimize execution on heterogeneous processors. A key contribution is a simple interface that enables efficient two-way communication between the Resource Manager (RM) and applications. This interface allows HARP to learn the application characteristics through online monitoring that generates Pareto-optimal operating points at runtime, leveraging principles inspired by HAM methodologies, which rely on these operating points for optimization. With this knowledge, HARP can allocate resource more effectively and informs applications about its decisions, enabling them to further adapt to the assigned resources.

The chapter begins by motivating the necessity of two-way communication channel between the RM and applications in Section 7.1. Next, Section 7.2 discusses the challenges of applying HAM methodologies to general-purpose environments. Section 7.3 presents the HARP system architecture and `libharp` client library, which facilitates communication between the RM and diverse application models. Section 7.4 details the runtime exploration of operating points for previously unknown workloads. Section 7.5 provides an extensive evaluation of HARP on two HMAs: the Odroid-XU3-E and the Intel Raptor Lake Core i9-13900K. The chapter ends with a synopsis in Section 7.6.

A Note on Publications and Contributions

Most of the content in this chapter, including discussion, HARP design, runtime exploration of operating points, figures, and results, was previously published in Smejkal, Khasanov, Castrillon, and Härtig, *E-Mapper: Energy-Efficient Resource Allocation for Traditional Operating Systems on Heterogeneous Processors*, 2024 [180].

The work presented in this chapter is a collaborative effort with Till Smejkal (Chair of Operating Systems, TU Dresden). The system's design and overall approach were developed collaboratively, with each contributor focusing on different core components. Till Smejkal designed and developed the HARP and `libharp` components, establishing the two-way communication protocol, integrating support for Intel TBB and OpenMP, and implementing performance and energy monitoring.

The author of this thesis focused on the resource allocation algorithm, online exploration of operating points (including regression model evaluation and selection), integrating `libharp` within the DPM library to support KPN and APN applications, and developing a custom wrapper for TensorFlow (implementation of the TensorFlow wrapper by Marc Dietrich).

Both contributors continuously refined the integrated solution through iterative discussions and joint evaluations.

7.1 NEED FOR TWO-WAY COMMUNICATION

The emergence of HMAs in general-purpose desktop and server systems poses challenges for resource management within operating systems. Such management must be fast and efficient, which is why modern OS schedulers typically rely on heuristics (Section 3.2.2).

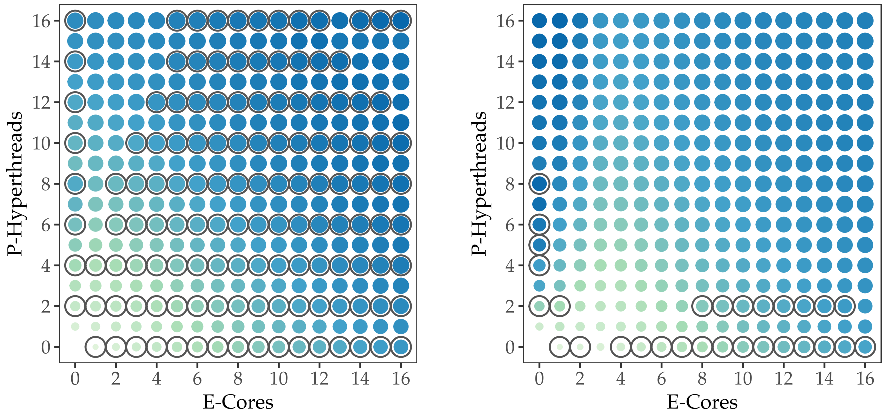
As discussed in Section 1.2, there are two key facets of adaptivity:

1. Resource managers must account for an application behavior characteristics to optimize core allocation.
2. Applications must dynamically adapt their execution to changing resource allocations at runtime.

Regarding the application behavior characteristics, Figure 7.1 illustrates that different applications benefit from different core types and their combinations. The figure presents two NAS Parallel Benchmark applications running on an Intel Raptor Lake Core i9-13900K.

For `bt.C` (Figure 7.1a), performance and power consumption scale smoothly with increasing core counts, particularly towards the upper right corner, indicating benefits from using both P-cores and E-cores. In contrast, `cg.C` (Figure 7.1b) does not benefit from more resources, especially when they are heterogeneous core combinations. In fact, its power consumption (size of the dot) increases with higher core counts without increasing performance (color of the dot). Here, `cg.C` performs significantly better on configurations that primarily use a homogeneous subset of the cores.

Conventional OS schedulers (Section 3.2.2) generally overlook such fine-grained behavioral differences. They allocate resources at the



(a) bt.C

(b) cg.C

Figure 7.1: Performance and power consumption of two applications on an Intel Raptor Lake Core i9-13900K with varying configurations. Each dot represents an application configuration with its thread distribution across E-cores (x-axis) and P-Hyperthreads (y-axis). Dot size reflects power consumption, and color indicates performance (darker blue for higher performance). Points highlighted with black circles represent Pareto-optimal configurations based on resource usage and an energy-utility cost metric (as defined later in Section 7.3.2.2).

thread level and often ignore the collective impact on the application performance. For heterogeneous platforms, a more holistic approach is needed — one that integrates knowledge of application behavior directly into its resource allocation strategy.

On the other side of the adaptivity challenge is the application itself. To fully leverage available resources in a dynamic environment, applications must adapt their execution at runtime. For data-parallel applications like those in Figure 7.1, optimal performance is typically achieved when the number of threads matches the allocated cores. Running more threads than cores introduces scheduling overhead and potential issues such as lock-holder preemption, while running fewer threads leaves resources underutilized.

An additional example of the need for adaptivity is demonstrated in the previous chapters with KPN applications, which cannot adjust their parallelization degree dynamically. Furthermore, it was shown that static distribution strategy also fail to balance parallel workers running across heterogeneous cores. As a result, energy-efficient cores take longer to execute, leaving the high-performance cores underutilized. The proposed APN application model address both these issues.

Beyond these examples, applications might exploit other forms of adaptivity: redistributing cores across different pipeline stages, selecting different algorithms based on the core types, or employing other application specific adaptivity features [155, 156, 166]. While the adaptivity features vary widely, a common requirement emerges: applications must be aware of their assigned resources to make informed adaptation decisions.

Enabling these scenarios requires coordination between a RM and applications. Ideally, the RM would dynamically provide information on resource allocations to applications, enabling them to optimize their resource usage proactively. At the same time, the RM should be informed of the application's behavior characteristics and adaptivity capabilities to better allocate the resources. This two-way communication between RMs and applications is essential to orchestrate both system-level and application-level optimizations.

7.2 ADAPTING HAM METHODOLOGIES

The proposed resource management approach, HARP, draws inspiration from Hybrid Application Mapping (HAM) methodologies. As explored in this thesis, HAM leverages application behavior characteristics through provided Pareto-optimal operating points (see Section 2.4.1). These points include information about application configurations, mappings, and non-functional characteristics that reflect application behavior.

However, applying HAM to general-purpose environments introduces several new challenges.

GENERATING OPERATING POINTS In traditional HAM approaches, applications are often known a priori, enabling Design Space Exploration (DSE) to generate operating points during the design stage. While the previous section argued that applications should provide their behavior characteristics (i.e., operating points) to the RM through a communication channel, it cannot be assumed that all applications will do so. Operating points may not be available for every application, and it is unclear whether users or developers should generate them. Developers would face the impractical challenge of performing DSE across varying system configurations, while users cannot be expected to pre-execute applications to generate these points.

EXECUTION-TIME VARIABILITY Applications behave differently depending on their inputs, affecting execution time and energy consumption. Estimating these non-functional characteristics for every input is impractical. Although some embedded systems approaches distinguish between "data scenarios" [183, 184], predicting completion times for general applications is as intractable as solving the halting problem. Furthermore, applications that run continuously, such as web browsers, require a shift from metrics like execution time and energy consumption that depend on the input size to *instant* metrics such as utility (e.g., Instructions Per Second (IPS)) and power consumption. This leads to the next issue.

DEFINING UTILITY Defining a universal utility metric for different applications is challenging. From the OS perspective, treating applications as black boxes limits utility to measurable metrics like executed instructions, which may not accurately reflect performance. For instance, busy-waiting loops can inflate instruction count without contributing useful work. More meaningful performance metrics, such as transactions per second, would need to be provided directly by the application.

HARP addresses these issues by utilizing the HAM methodology in a following way. First, it leverages instant non-functional characteristics that do not depend on data input size. Second, it provides a mechanism to obtain application-specific utility information via the communication channel between HARP and applications (as discussed in Section 7.1). Finally, it introduces a novel component that performs quick exploration of application configurations through an online monitoring tool and generates operating points at runtime. This approach effectively transitions the design-time component of HAM to runtime.

7.3 HARP DESIGN

The *Heterogeneity-aware Adaptive Resource Partitioning* (HARP) resource management framework introduced here builds upon principles from Hybrid Application Mapping (HAM) methodologies. Like HAM, HARP utilizes application descriptions in the form of operating points. Unlike traditional HAM, however, HARP does not rely solely on design-time analysis; instead, it also generates operating points through online monitoring.

To coordinate resource management across diverse application models, HARP provides a `libharp` library, which works in tandem with the HARP Resource Manager (RM). Figure 7.2 shows the HARP's management approach. A single instance of the HARP Resource Manager (RM) oversees all managed applications, while each application links with `libharp`, a client library that mediates communication between the RM and the adaptation.

The HARP RM makes high-level decisions on resource allocations and core affinities, working alongside the OS scheduler, which handles lower-level thread scheduling according to these decisions. The RM adjusts allocations dynamically whenever applications start, exit, or other system events occur. These decisions are guided by hardware descriptions (❶) and by application operating points, either parsed from application descriptions (❷) by `libharp` or derived at runtime via HARP's fast exploration heuristics (❺), further detailed in Section 7.4.

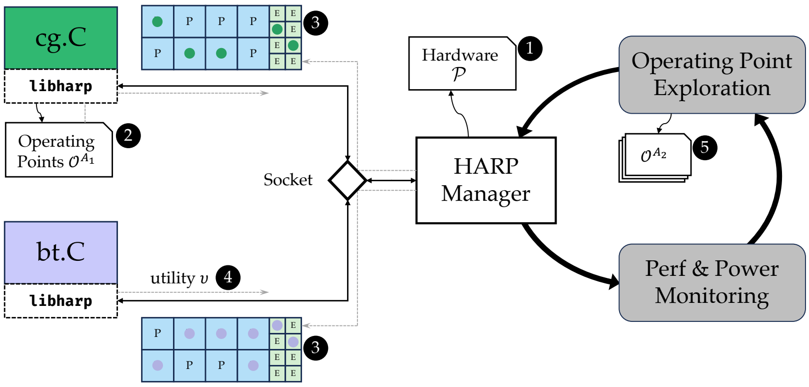


Figure 7.2: Overview of the HARP management approach and system design with two example applications.

7.3.1 *Application Support via `libharp`*

`libharp` serves as an intermediary between the RM and applications, handling tasks like application registration, adjusting application configurations based on the RM triggers, and providing utility metrics. While it uses a standardized communication protocol to ensure effective runtime adaptations, `libharp` can manage applications differently based on their supported resource types (Section 7.3.1.2) and adaptivity (Section 7.3.1.3).

7.3.1.1 *Communication Protocol*

`libharp` communicates with the RM through a simple interface that is handled using `protobuf`[1] messages over Unix sockets. Figure 7.3 illustrates a typical interaction between the managed application and the HARP RM:

1. *Registration Request*: Upon startup, `libharp` initializes the connection with the RM by sending a registration request through the RM's Unix socket. This request includes the application's PID and the supported resource type (Section 7.3.1.2). At this point, `libharp` also creates a dedicated Unix socket for receiving push messages from the RM.

2. *Operating Points and Utility Subscription*: After registration, the application sends available operating points from the application description file (②) to the RM. Along with that, the library indicates further available functionality depending on the application adaptivity type (Section 7.3.1.3), for example that it can provide real-time utility metrics.

3. *Operating Point Activation*: After completing the initial setup, the RM generates a new resource allocation. It communicates

---

[1] `https://protobuf.dev/`

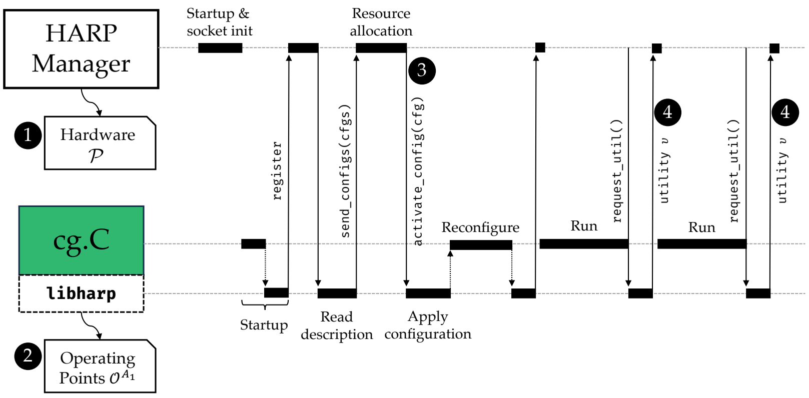


Figure 7.3: Typical control flow between a managed application and the HARP resource manager.

this decision to `libharp`, specifying the selected operating point and its allocation on the processing resources (③). The library will accordingly update the application configuration to the provided resources. Such reconfiguration messages from the RM not only happen at the startup of the application but can also occur whenever the RM is triggered.

4. *Utility Feedback*: If the utility feedback feature was indicated, the RM periodically requests the current utility from `libharp` (④) to refine its operating point models and improve allocations (⑤).

7.3.1.2 *Resource Allocation Types*

`libharp` supports two types of resource allocations, differing by mapping granularity.

- *Coarse-grained allocations*: Define an *extended resource vector* indicating the number of cores (and their hardware threads) allocated per core type. For example, if an application uses 4 E-cores and 3 P-cores (which support simultaneous multithreading), where two of the P-cores use two hardware threads each and the third only one, the extended resource vector would be $[P^{ST}, P^{HT}, E]^\top = [1, 2, 4]^\top$. This extended resource vector is defined for the entire application, implying a uniform thread distribution.
- *Fine-grained allocations*: Define more detailed mappings by assigning each application thread to a specific core in the system. These points may also include application configurations that can be adjusted at runtime via *adaptivity knobs*.

The distinction between coarse- and fine-grained resource allocations stems from their differing strengths and trade-offs. Coarse-grained allocations are simpler since only different core combinations have to be distinguished, offering a fixed and uniform structure across applications. This simplicity allows the HARP RM to explore operating points at runtime (Section 7.4). In contrast, fine-grained allocations define a much larger and more complex search space, making online exploration impractical.

At the same time, fine-grained mappings are specialized for more complex application structures and may encode internal application configurations. Notably, the RM does not receive a detailed information about thread-to-core mapping and configurations. Instead, `libharp` communicates only the extended resource vector containing the required hardware resources, just as with coarse-grained allocations.

7.3.1.3 *Application Adaptivity Types*

In HARP, applications are categorized into three adaptivity types: static, scalable, and custom.

STATIC APPLICATIONS Static applications lack runtime adaptation mechanisms and have fixed internal structures, making them *rigid* (Section 3.4). Both fine-grained and coarse-grained allocation types can be applied to such applications. Fine-grained allocation maps individual threads to specific cores, while coarse-grained restricts the entire application to a subset of the cores.

`libharp` provides native support for static applications, with automatic registration to the RM upon library load. Thread management is achieved by intercepting `pthread_*()` functions. A drawback, however, is that performance can degrade if the number of threads exceeds the allocated cores, as the OS scheduler will time-multiplex the assigned cores among the threads.

SCALABLE APPLICATIONS Scalable applications support some form of runtime scaling. Libraries such as OpenMP and Intel TBB enable implicit data-level parallelism, making these applications inherently *scalable*. Since parallel worker threads in these applications usually perform the same operations on different data, they naturally support coarse-grained resource allocations.

Typically, the parallelization degree is fixed at the launch, classifying these applications as *moldable* (Section 3.4). `libharp` extends them to be *malleable*, allowing their parallelization degree to be adjusted at runtime using a built-in adaptivity knob. For example, in OpenMP, `libharp` hooks into the `GOMP_parallel` function to modify the `num_threads` variable, ensuring that the number of worker threads matches the hardware threads allocated by HARP RM. This adjustment prevents time-sharing processor cores, which the static applications may suffer from.

Additionally, `libharp` can manage applications with their own scalability knob in a similar manner. For example, a custom wrapper library developed for TensorFlow [56], utilized in the evaluation (Section 7.5), implements this functionality on the application side.

CUSTOM APPLICATIONS Custom applications extend `libharp` for application-specific adaptations, typically employing fine-grained allocations. A prime example is Adaptive Process Network (APN) applications, which use the DPM library for runtime support (Chapter 6). Linked to `libharp`, DPM receives the selected configuration from HARP RM through the RM mediator component (Figure 6.5). This integration enables dynamic scaling specific parallel regions and fine-grained resource allocation across different regions.

Other custom extensions might leverage the communicated resource allocations by selecting alternative algorithms, using specialized code paths, or handling ISA-extension differences between core types. This flexibility makes `libharp` suitable to a wide range of application-specific adaptation mechanisms.

7.3.1.4 *Making Applications HARP-ready*

Most static and scalable applications are supported by `libharp` out-of-the-box, without requiring additional modifications. The library automatically detects supported runtime libraries and application threads, adapting them through function hooks.

For custom adaptations, developers need to extend a `libharp`-internal interface to handle resource allocation updates and implement the reconfiguration logic based on the provided information. `libharp` invokes this mechanism, passing the operating point and resource allocation to the custom adaptation extension for reconfiguration. Similarly, developers can also provide utility metric feedback by implementing a corresponding interface in the library.

7.3.2 *Resource Allocation*

The HARP RM allocates heterogeneous resources to concurrently running applications, balancing their resource needs while optimizing system energy efficiency.

7.3.2.1 *Non-Functional Characteristics*

As discussed in Section 7.2, HARP relies on *instant* metrics — utility and power consumption — annotated in each operating point (cf. Figure 7.1). The *power consumption* metric, $\eta[p]$, represents the average power attributed to the application, while the *utility* metric, $\eta[v]$, quantifies the useful work performed by the application.

Utility can be measured using generic metrics such as *Instructions Per Second* (IPS), obtained from `perf`, or application-specific metrics like transactions per second, which more accurately reflect meaningful work. Utility data may come from description files or from a running application via the utility feedback interface.

To account for varying interpretations and magnitudes of utility across applications, HARP normalizes utility by dividing it by the maximum observed value for the application, denoted as $\eta[v^*]$.

7.3.2.2 *Energy-Efficient Resource Allocation*

HARP generates a spatial multi-application mapping as defined in Section 2.4.2.2. Similar to state-of-the-art spatial resource allocation techniques (Section 3.3), HARP formulates this as an optimization problem, as described in Definition 2.14.

To balance energy efficiency and performance, HARP uses an *energy-utility cost* $\eta[\zeta]$, which is derived from the traditional Energy-Delay Product (EDP) formula [124, 144]. Assuming utility is inversely proportional to delay, the energy-utility cost is defined as:

$$\eta[\zeta] = \left(\frac{\eta[p]}{\eta[v^*]}\right) \cdot \left(\frac{1}{\eta[v^*]}\right) \tag{7.1}$$

Before solving the optimization problem, HARP applies Pareto filtering to the operating points, retaining only the non-dominated subset with respect to the following objectives: the values in the extended resource vector (as defined in Section 7.3.1.2) and the energy-utility cost.

The optimization problem is then solved using a state-of-the-art approximation algorithm based on Lagrangian relaxation. This approach solves the problem under relaxed constraints, selects the final operating points meeting resource constraints, and finally assigns resources to applications, ensuring no overlap. The implementation is inspired by the work of Wildermann et al. [206, 207].

LIMITATIONS The algorithm may fail to find suitable operating points for some applications due to prior allocations. While it optimizes resource costs to avoid exceeding available resources, such situations are unavoidable when the number of managed applications exceeds the number of available resources. In such cases, HARP temporarily relaxes the resource constraint (Equation (2.16)), allowing applications to execute in co-allocation. Since co-allocation adversely affects performance, HARP disables performance monitoring during these periods.

7.4 RUNTIME EXPLORATION OF OPERATING POINTS

As highlighted in Section 7.2, applications running on desktops and servers usually lack predefined application descriptions with operating points, or available points may be imprecise due to hardware variations. To address this, HARP integrates runtime exploration of operating points, a synergy between Design Space Exploration and Runtime Resource Management. This integration addresses several challenges.

First, accurate online measurements of utility and power consumption are essential for determining operating point characteristics during application execution. For applications without specific utility metrics, Instructions Per Second (IPS) serves as a generic utility measure. Both utility and power metrics, however, are inherently variable due to factors such as measurement noise and the dynamic behavior of applications, including shifts between sequential and parallel stages or between compute- and memory-intensive stages. These fluctuations necessitate periodic re-evaluation to ensure robust and reliable operating points. This online monitoring component is discussed in Section 7.4.1.

Second, exhaustive exploration of all possible operating points at runtime is impractical. Instead, HARP relies on both actual measurements and approximations for unexplored operating points. By employing a regression model, HARP accelerates the identification of the Pareto front with a limited number of measurements. The model is continuously refined as new data becomes available, efficiently ensuring robustness of the identified Pareto-optimal operating points. Section 7.4.2 provides an evaluation of different regression models used in this process.

Third, runtime exploration must seamlessly integrate into the HARP RM, ensuring that it does not compete with other concurrently running applications for processor cores. This integration requires allocating sufficient resources to new applications, providing them with substantial solution space for exploration without compromising the performance of existing workloads.

To achieve this, HARP RM incorporates operating point exploration directly into its resource allocation algorithm. The algorithm distributes resources between applications, ensuring that those in the exploration stage have enough cores to evaluate potential configurations. Additionally, the exploration process guides the selection of operating points for measurement within the allocated resources, using continuous performance monitoring and regression models to dynamically adjust configurations. This approach accelerates the identification of robust Pareto-optimal operating points while maintaining efficient resource utilization and optimizing overall system performance. Details on the exploration algorithm are presented in Section 7.4.3.

7.4.1 *Runtime Performance and Power Monitoring*

Accurate runtime performance monitoring is essential for refining Pareto fronts. When applications provide their own utility metric, HARP RM bases its algorithm on these values. Otherwise, HARP utilizes the Linux `perf` subsystem[2] to monitor IPS as a generic utility measure. With appropriate configuration, `perf` can automatically multiplex measurements across applications.

For power consumption, HARP relies on built-in power sensors available in modern systems, such as the Running Average Power Limit (RAPL) counters on Intel machines. Since these counters typically measure system-wide energy rather than per application, HARP builds atop EnergAt [80], which estimates per-application power usage by combining RAPL data with thread execution metrics.

To handle heterogeneous systems, the EnergAt approach was extended to account for different core types. This extension introduces power coefficients ($P^P = \gamma \cdot P^E$, determined offline) to attribute total energy consumption (E_Δ^{CPU}) to P-cores (E_Δ^P) and E-cores (E_Δ^E), based on CPU time:

$$E_\Delta^{CPU} = E_\Delta^P + E_\Delta^E = T_{total}^P \cdot P^P + T_{total}^E \cdot P^E \tag{7.2}$$

where $T_{total}^{P|E}$ represents the total CPU time on P-cores and E-cores, respectively. Using this model, HARP estimates E_Δ^P and E_Δ^E, then applies EnergAt's approach to further attribute energy to applications running on homogeneous subsets of cores.

To smooth the inherent variability in measured utility and power, HARP applies an Exponential Moving Average (EMA) with a smoothing factor of $\alpha = 0.1$, updating values as follows:

$$value_{new} = value_{measured} \cdot \alpha + value_{old} \cdot (1 - \alpha) \tag{7.3}$$

This approach stabilizes short-term fluctuations while adapting to significant shifts in application behavior, ensuring accurate and responsive profiling of performance and power characteristics.

7.4.2 *Selection of the Regression Model*

To approximate utility and power consumption for unexplored operating points, HARP employs a regression model to predict these metrics based on coarse-grained resource allocations represented by the extended resource vector (Section 7.3.1.2).

---

[2] Accessed via the system call `perf_event_open`

To determine the most suitable regression model, Polynomial Regression (degrees 1 to 3), Neural Network (NN), and Support Vector Machine (SVM) models were evaluated. The evaluation utilized pre-measured data from 15 applications on the Intel Raptor Lake Core i9-13900K. Each model was trained on subsets of training data with varying sizes, with experiments repeated across 10 random seeds to ensure robustness.

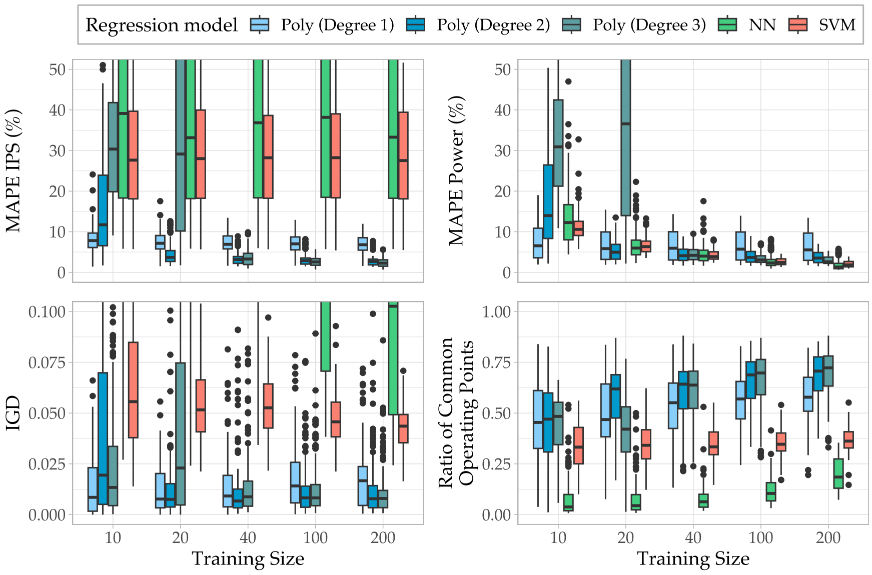


Figure 7.4: Evaluation of regression models for runtime exploration of operating points. Metrics include Mean Absolute Percentage Error (MAPE) for IPS and Power (lower is better), Inverted Generational Distance (IGD) (lower is better), and the ratio of common operating points in the Pareto-front (higher is better), evaluated across 15 different applications on the Intel Raptor Lake Core i9-13900K.

As shown in the top two plots of Figure 7.4, Polynomial Regression improves in accuracy for both utility and power as training set size increases. Higher-degree polynomial models achieve greater accuracy at larger training sizes, albeit requiring more data points to converge. Conversely, Neural Network (NN) and Support Vector Machine (SVM) models perform better for power predictions at larger training sizes but struggle with utility (IPS) predictions.

The bottom two plots in Figure 7.4 evaluate the alignment between predicted and reference Pareto fronts (derived from measured configurations) using two metrics: Inverted Generational Distance (IGD) [50] and the ratio of common operating points.

- IGD measures the average distance from points on the reference Pareto front to the nearest point on the predicted front, indicating coverage.
- The ratio of common operating points measures the overlap between the predicted and reference fronts.

Polynomial regression consistently outperforms SVM and NN in aligning with the reference Pareto front. Among polynomial models, second- and third-degree models show better alignment than the first-degree model. While the second- and third-degree models achieve similar accuracy, the second-degree model is more efficient, requiring only 20 training points to converge and generate robust Pareto fronts.

Based on these findings, HARP adopts a second-degree polynomial regression model for runtime exploration due to its balance of accuracy and efficiency.

7.4.3 *Runtime Exploration Algorithm*

The runtime exploration of operating points integrates seamlessly with the resource allocation algorithm, ensuring that cores assigned to applications undergoing exploration do not overlap with cores used by other applications. At the same time, sufficient resources are allocated for efficient exploration without adversely affecting the performance of other concurrently running applications.

To guide the exploration process, the maturity of application operating points is classified into three stages:

1. *Initial Stage*: There are insufficient measured operating points, making approximations unreliable.
2. *Refinement Stage*: An intermediate number of measured points exist, but model accuracy is still limited.
3. *Stable Stage*: A sufficient number of points have been measured, enabling reliable approximations.

Figure 7.5 illustrates how the runtime exploration of operating points is integrated with the resource allocation algorithm. The process consists of two stages: it begins with spatial resource allocation, followed by exploration within the bounds of assigned resources. As shown in the figure, the exploration strategy in the second stage varies depending on the maturity stage of each application's operating points.

To generate a multi-application mapping, the mapper uses a Pareto-optimal set of operating points comprising both measured and approximated configurations. If unassigned processor cores remain, they are evenly distributed among applications in the initial or refinement stages, allowing them to explore a broader configuration space. Applications in the stable stage execute on the designated cores, without additional exploration.

Applications in the initial and refinement stages apply distinct exploration techniques within the set of assigned cores.

In the *initial stage*, when measured points are insufficient to create even a preliminary model, the objective is to evenly distribute the

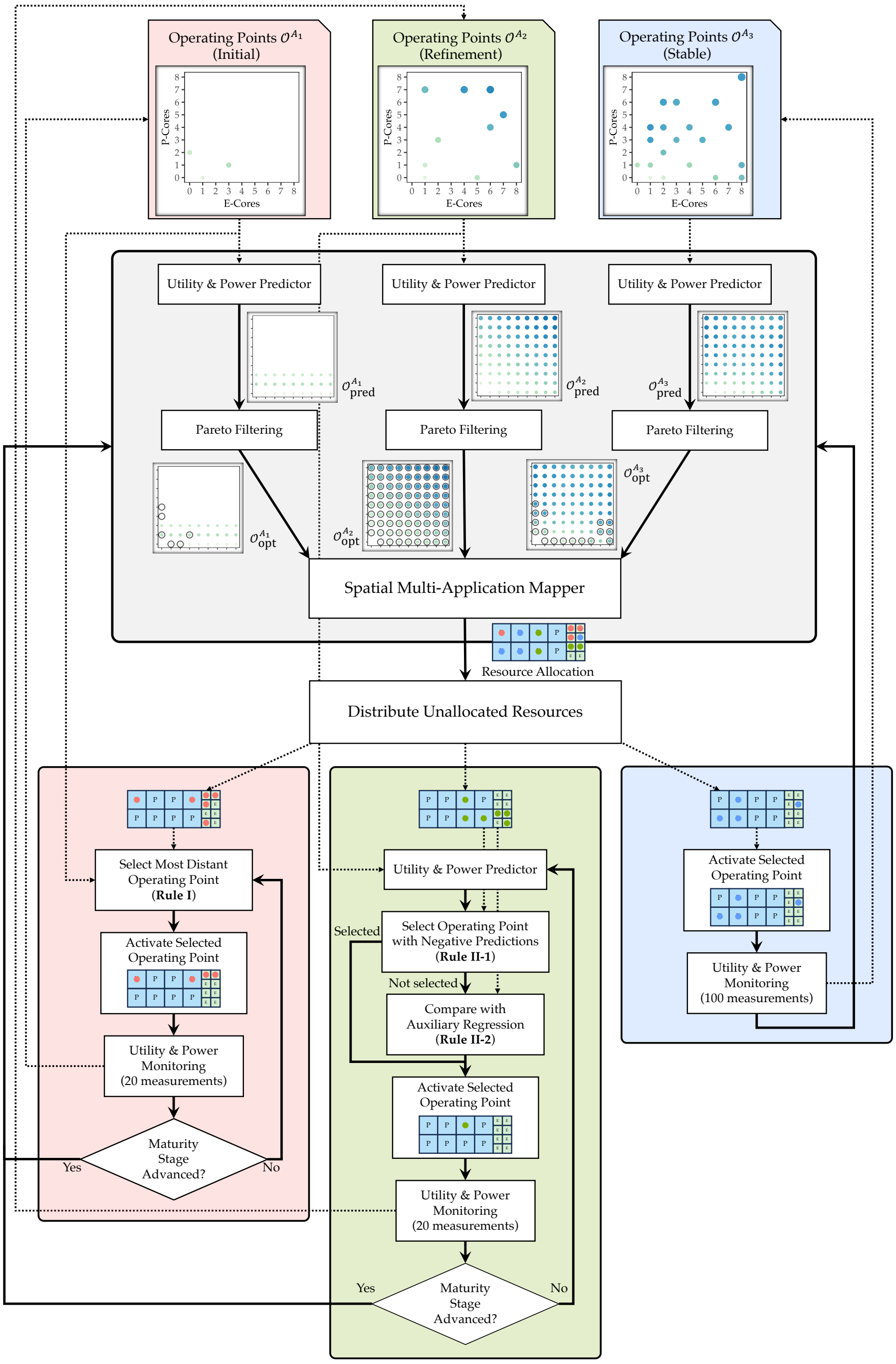

Operating Points $\mathcal{O}^{A_1}$ (Initial)
Operating Points $\mathcal{O}^{A_2}$ (Refinement)
Operating Points $\mathcal{O}^{A_3}$ (Stable)
P-Cores
E-Cores
Utility & Power Predictor
$\mathcal{O}^{A_1}_{\text{pred}}$
$\mathcal{O}^{A_2}_{\text{pred}}$
$\mathcal{O}^{A_3}_{\text{pred}}$
Pareto Filtering
$\mathcal{O}^{A_1}_{\text{opt}}$
$\mathcal{O}^{A_2}_{\text{opt}}$
$\mathcal{O}^{A_3}_{\text{opt}}$
Spatial Multi-Application Mapper
Resource Allocation
Distribute Unallocated Resources
Select Most Distant Operating Point (Rule I)
Activate Selected Operating Point
Utility & Power Monitoring (20 measurements)
Maturity Stage Advanced?
Yes
No
Utility & Power Predictor
Selected
Select Operating Point with Negative Predictions (Rule II-1)
Not selected
Compare with Auxiliary Regression (Rule II-2)
Activate Selected Operating Point
Utility & Power Monitoring (20 measurements)
Maturity Stage Advanced?
Yes
No
Activate Selected Operating Point
Utility & Power Monitoring (100 measurements)
P
E

Figure 7.5: Integration of runtime exploration of operating points with resource allocation. Exploration strategies are tailored to the maturity stage of each application's operating points: initial, refinement, or stable.

measurements across the configuration space. This ensures a diverse dataset to quickly build a model. To achieve this, the heuristic selects the next operating point that is maximally distant from all previously measured configurations in the extended resource vector space, thereby maximizing diversity (*Rule I*).

In the *refinement* stage, a preliminary regression model based on earlier measurements becomes available, but it may still contain inaccuracies and anomalies, such as negative utility and power predictions. The focus at this stage is on refining the model's accuracy by strategically selecting configurations for measurements. The selection heuristic operates as follows:

1. Configurations with negative utility and power predictions are prioritized. Among those, the configuration with the largest negative value — calculated as the geometric mean of both utility and power "errors" (with positive values treated as zero) — is selected (*Rule II-1*).
2. If no configurations have negative predictions, the heuristic compares the primary regression model and an auxiliary one anchored with a "zero" point (representing zero utility and power). The next operating point is then selected based on the largest geometric mean of relative discrepancies in predicted utility and power values (*Rule II-2*).

Each selected operating point undergoes a predefined number of measurements (in the evaluation, 20 measurements at 50 ms intervals). Once measurements for an operating point are completed, a new operating point is selected for evaluation. This iterative process repeats until the application has explored 25 different configurations and transitions to the *stable stage*. When an application's operating point maturity advances (e.g., from initial to refinement, or refinement to stable), a new resource allocation is generated.

In the stable stage, active exploration ends, and the application executes under the operating point selected by the resource allocation algorithm (Section 7.3.2). The resource allocation process is then invoked with at longer intervals (e.g., every 100 measurements) to reassess and, if necessary, update allocations.

This integrated design ensures that applications in the initial and refinement stages receive sufficient resources for exploration, minimize interference with concurrently running applications, and benefit from strategically guided exploration that efficiently improves prediction accuracy.

7.5 EVALUATION

To demonstrate how HARP enhances the management of energy-aware tasks on heterogeneous processors, the proposed resource management approach was evaluated on two distinct platforms — highlighting its generality across diverse types of heterogeneous hardware — and across a range of application models and execution scenarios.

7.5.1 *Experimental Setup*

The experimental setup comprises two heterogeneous platforms, a diverse set of benchmark applications and comparative resource management strategies. The following describes the hardware configurations, application characteristics, and methodology used throughout the evaluation.

7.5.1.1 *Evaluated Platforms*

The evaluation is conducted on two platforms, representing embedded and desktop computing domains.

- *Odroid-XU3-E* [76]: Representing the embedded domain, this board features a Samsung Exynos 5422 processor with an Arm big.LITTLE architecture, comprising a four-core Cortex-A15 (big) island and a four-core Cortex-A7 (LITTLE) island. It is similar to the Odroid-XU4 board used in the previous evaluations (Sections 4.3.2, 4.4.3, 5.6 and 6.4), but includes internal energy sensors for the core islands, the memory, and GPU. A custom-compiled Linux 6.6 kernel with full support for the Linux Energy-Aware Scheduler (EAS) [117, 145] is used in the experiments.
- *Intel Raptor Lake Core i9-13900K*: Representing the desktop domain, this processor comprises 8 high-performance P-cores with SMT and 16 energy-efficient E-cores without SMT. It is equipped with 128 GB of memory. Energy measurements are captured using RAPL counters [54]. A custom-compiled Linux 6.4 kernel is used, based on the default Debian Testing kernel, with a patch-set that adds preliminary support for ITD [137]. The patch-set is further extended to provide user-space access to ITD classifications of threads and reference Instructions Per Cycle (IPC) per thread class.

Both platforms use the `performance` frequency governor, with maximum frequencies capped to prevent thermal throttling: 4.6 GHz for P-cores and 3.8 GHz for E-cores on the Intel Raptor Lake, and 1.2 GHz for LITTLE and 1.8 GHz for big cores on the Odroid.

7.5.1.2 *Benchmarks*

To evaluate the features of HARP, different sets of applications are employed. For testing dynamic adaptability, the OpenMP implementations of the NAS Parallel Benchmarks [11], version 3.4.2, are used.

Since the Intel platform is more powerful than the Odroid board, different classes of the benchmarks are used: class *A* for the Odroid platform and class *C* for the Intel platform.

On the Intel system, additional evaluations are conducted using a selection of benchmarks from Intel Thread Building Blocks (Intel TBB) [146] and two TensorFlow [1] applications. The Intel TBB benchmarks include `binpack`, `fractal`, `pp` (`parallel-preorder`), `pi`, `primes`, and `seismic`, chosen for their comprehensive coverage of the building blocks provided by the framework. TensorFlow, a widely used open-source framework for machine-learning algorithms, is evaluated with the HARP-enabled wrapper library of TensorFlow Lite [192], which supports runtime parallelism scaling. Two image recognition models, VGG [173] and AlexNet [104], are used in the experiments.

Additionally, two embedded dataflow applications are employed to evaluate the custom extensions of HARP: `mandelbrot` for calculating the Mandelbrot set (Section 6.4), and `lms` implementing Leighton-Micali Signatures [127]. Each application is used in two versions: one with a static application topology, based on the original KPN model (denoted as `static`), and another implemented as an APN application, with adaptive data-parallelism and scalability features (Section 6.2). These configurations demonstrate the dynamic adaptation capabilities of HARP. Since dataflow applications target embedded platforms, these applications are evaluated only on the Odroid system.

7.5.1.3 *Evaluated Approaches*

Each scenario is executed with HARP and compared against the following baselines, depending on the platform:

- For Intel Raptor Lake:
  - *Completely Fair Scheduler (CFS)*: The default Linux scheduler that uses built-in work-distribution techniques.
  - *ITD-Based Allocation (ITD)*: An extended allocator inspired by Saez et al. [162], which leverages hardware-provided ITD classifications to allocate processor cores to application threads.
- For Odroid-XU3-E:
  - *Energy-Aware Scheduler (EAS)*: The Linux Energy-Aware Scheduler (*EAS*), which leverages a power model of the heterogeneous processor on the Arm big.LITTLE system.

The evaluation includes several configurations of HARP:

- *HARP*: The standard version of HARP, with enabled adaptivity features, using *stable* operating points generated through online monitoring.

- *HARP (Offline)*: A version utilizing pre-generated operating points, showcasing the potential benefits of prior design-time exploration.
- *HARP (No Scaling)*: A version with application adaptation disabled, emphasizing the impact of application adaptivity on HMAs.

On the Odroid-XU3-E platform, simultaneous performance counter tracking for the big and LITTLE clusters is not supported, making online monitoring infeasible. Consequently, the evaluation of HARP on this platform is limited to the version using pre-generated operating points (*HARP (Offline)*).

For the Intel Raptor Lake system, additional details on the runtime exploration process for operating points are provided in Section 7.5.4.

7.5.2 *Intel Raptor Lake Evaluation*

The evaluation on the Intel Raptor Lake system examines the performance and energy consumption of the benchmarks in both single and multi-application scenarios. For each scenario, the overall execution time (makespan) and total energy consumption are measured, with average results reported over ten repetitions. Error bars are displayed for cases where relative errors exceed 5 %.

Figure 7.6 shows the results for all benchmarks, along with geometric means for both single and multi-application scenarios. Results are expressed as improvement factors over the baseline (*CFS*), where an improvement factor of F indicates $F\times$ faster execution or $F\times$ lower energy consumption compared to the baseline. Naturally, higher improvement factors indicate better outcomes.

7.5.2.1 *Single-Application Scenarios*

In single-application scenarios, HARP effectively reduces energy consumption, often with a slight trade-off in execution time. This behavior aligns with HARP's cost function, which balances performance and energy efficiency and often prioritizes configurations that reduce resource utilization while maintaining acceptable performance. On average, HARP achieves an improvement factor of 0.96 for execution time and 1.32 for energy consumption.

One notable outlier is `binpack`, which exhibits significant improvements of 6.91 in makespan and 1.29 in energy consumption. In this scenario, `binpack` achieves a $10\times$ higher IPS with lower thread counts. Unlike the baseline and ITD-based allocator, HARP successfully scales the application down to exploit this behavior.

The ITD-based allocator, by contrast, shows only minor improvements over CFS, with average improvement factors of 1.02 for execu-

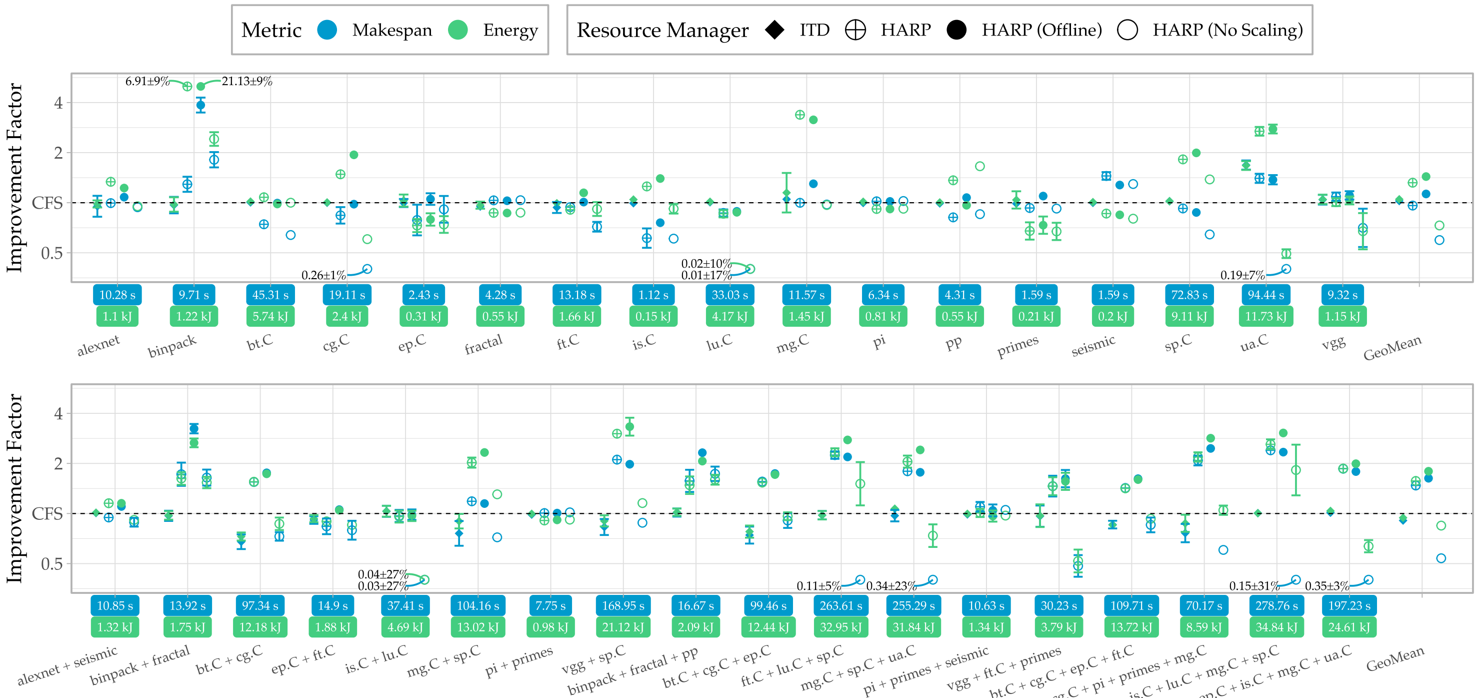


Figure 7.6: Relative improvement factor of HARP and Intel Thread Director (ITD) over CFS on the Intel Raptor Lake Core i9-13900K (higher y-value is better). Results also include HARP with offline-generated operating points and HARP without application adaptation. Blue and green boxes indicate the average makespan and energy consumption for each scenario when executed with CFS.

tion time and 1.05 for energy consumption, both within the margin of error.

HARP with offline-generated operating points demonstrates the advantages of prior design-time exploration, achieving improvements of 1.29 for execution time and 1.43 for energy consumption. Offline DSE provides additional insights into application behavior, enabling better resource allocation.

In contrast, HARP without application adaptation performs very poorly, with a geometric mean of 0.60 for execution time and 0.73 for energy consumption. These results indicate the critical role of dynamic application adaptation for effective resource management in heterogeneous systems.

7.5.2.2 *Multi-Application Scenarios*

In multi-application scenarios, HARP delivers significant performance gains, with average improvement factors of 1.47 for execution time and 1.57 for energy consumption. Most scenarios benefit from HARP, with only a few cases showing results close to the baseline.

The ITD-based allocator does not show improvements in multi-application scenarios, similar to the single-application results. Most scenarios achieve performance close to the baseline, with minor gains observed in the scenarios `is + lu` and `bt + ep + is + mg + ua`. Overall, ITD achieves improvement factors of 0.91 for execution time and 0.94 for energy consumption.

As with single-application scenarios, HARP with offline-generated operating points achieves superior results, with improvement factors of 1.63 for execution time and 1.79 for energy consumption.

Finally, HARP without application adaptation again shows very poor results, with improvement factors of 0.54 for execution time and 0.84 for energy consumption. This highlights the necessity of runtime application adaptivity in dynamic heterogeneous systems.

7.5.3 *Odroid-XU3-E Evaluation*

As mentioned in Section 7.5.1.3, the Odroid-XU3-E does not support simultaneous performance counter tracking for the big and the LITTLE clusters. Consequently, online monitoring is not feasible on this platform. For this evaluation, only *HARP (Offline)* is used, utilizing pre-generated operating point tables. This approach aligns with the common practice of offline DSE in embedded systems (Section 3.3).

The baseline for comparison is the Linux Energy-Aware Scheduler (EAS), which leverages a power model of the heterogeneous processor. The results of this evaluation are shown in Figure 7.7.

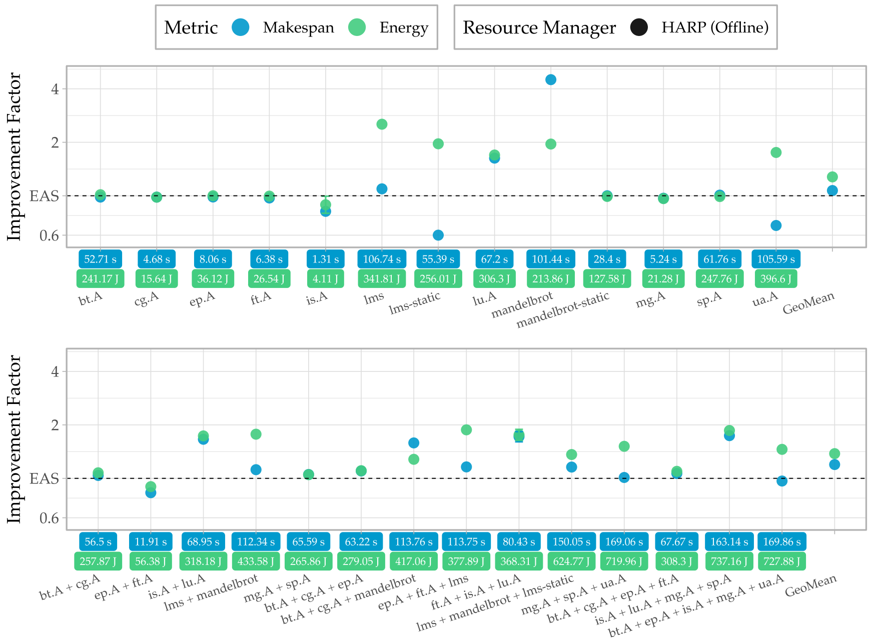


Figure 7.7: Relative improvement factor of HARP over EAS on the Odroid-XU3-E (higher y-value is better). Blue and green boxes indicate the average makespan and energy consumption for each scenario when executed with EAS.

7.5.3.1 *Single-Application Scenarios*

In single-application scenarios on the Odroid-XU3-E, HARP achieves an average improvement factor of 1.07 for makespan and 1.27 for energy consumption. The results for OpenMP benchmarks generally reflect those from the Intel Raptor Lake experiment, where HARP reduces energy consumption at the expense of longer execution times. For instance, the `ua` benchmark exemplifies this trade-off.

The `is` benchmark, however, shows increases in both execution time and energy consumption. As a short-running application, `is` is significantly impacted by the initialization overhead of HARP, particularly the communication with the RM. During this phase, the application executes in a suboptimal configuration, affecting its overall performance. In contrast, `lu`, a longer-running benchmark, benefits significantly from HARP.

APN applications with internal adaptivity mechanisms show notable improvements with HARP. By leveraging application-level adaptivity, HARP effectively adjusts configurations to align with available hardware resources. This results in reduced energy consumption for `lms` and improvements in both execution time and energy consumption for `mandelbrot`.

Static KPN versions of these applications, however, do not benefit much from HARP. The behavior of `mandelbrot-static` is similar to the baseline, while `lms-static` exhibits reduced energy consumption at the cost of increased execution time.

7.5.3.2 *Multi-Application Scenarios*

In multi-application scenarios, HARP optimizes resource usage, achieving average improvement factors of 1.20 for execution time and 1.38 for energy consumption. These results indicate that, similar to the Intel platform, HARP delivers better performance and energy efficiency when managing multiple applications.

An exception is the `ep + ft` scenario, which shows slight degradation in both metrics compared to EAS. This outcome is attributed to thread migrations between the big and LITTLE clusters during executions due to resource reassignments done by HARP. Despite this, the overall results emphasize HARP's effectiveness in optimizing resource allocation in embedded multi-application environments.

7.5.4 *Evaluation of the Learning Process*

To examine the behavior of HARP during the learning process in scenarios where no prior application knowledge is available, an additional series of experiments was conducted on the Intel Raptor Lake system.

Each scenario starts with no predefined application behavior data and begins with a warm-up phase, during which HARP monitors application execution and learns their behavioral characteristics until operating points stabilize.

To analyze the learning process, operating points are captured as snapshots every 5 s. These snapshots are used to determine the quality of each learning step. Figures 7.8 and 7.9 present the results for 17 single-application and 18 multi-application scenarios, respectively. Each data point represents an improvement factor for makespan and energy consumption based on the operating points at the corresponding snapshot. Shaded areas in the figures indicate the duration of the learning stage.

7.5.4.1 *Single-Application Scenarios*

In single-application scenarios (Figure 7.8), most applications demonstrate fluctuating performance and energy efficiency during the training stages (either the initial or refinement stages, as defined in Section 7.4.3). Once stable operating points are reached, performance stabilizes, though minor fluctuations persist. This is expected, as HARP continues to refine operating points even in the stable stage (Section 7.4.3). On average, the stable stages are achieved within 35.7 ± 9.4 s.

However, two applications, `binpack` and `ep`, exhibit notable deviations. `binpack` exhibits persistent fluctuations even in the stable

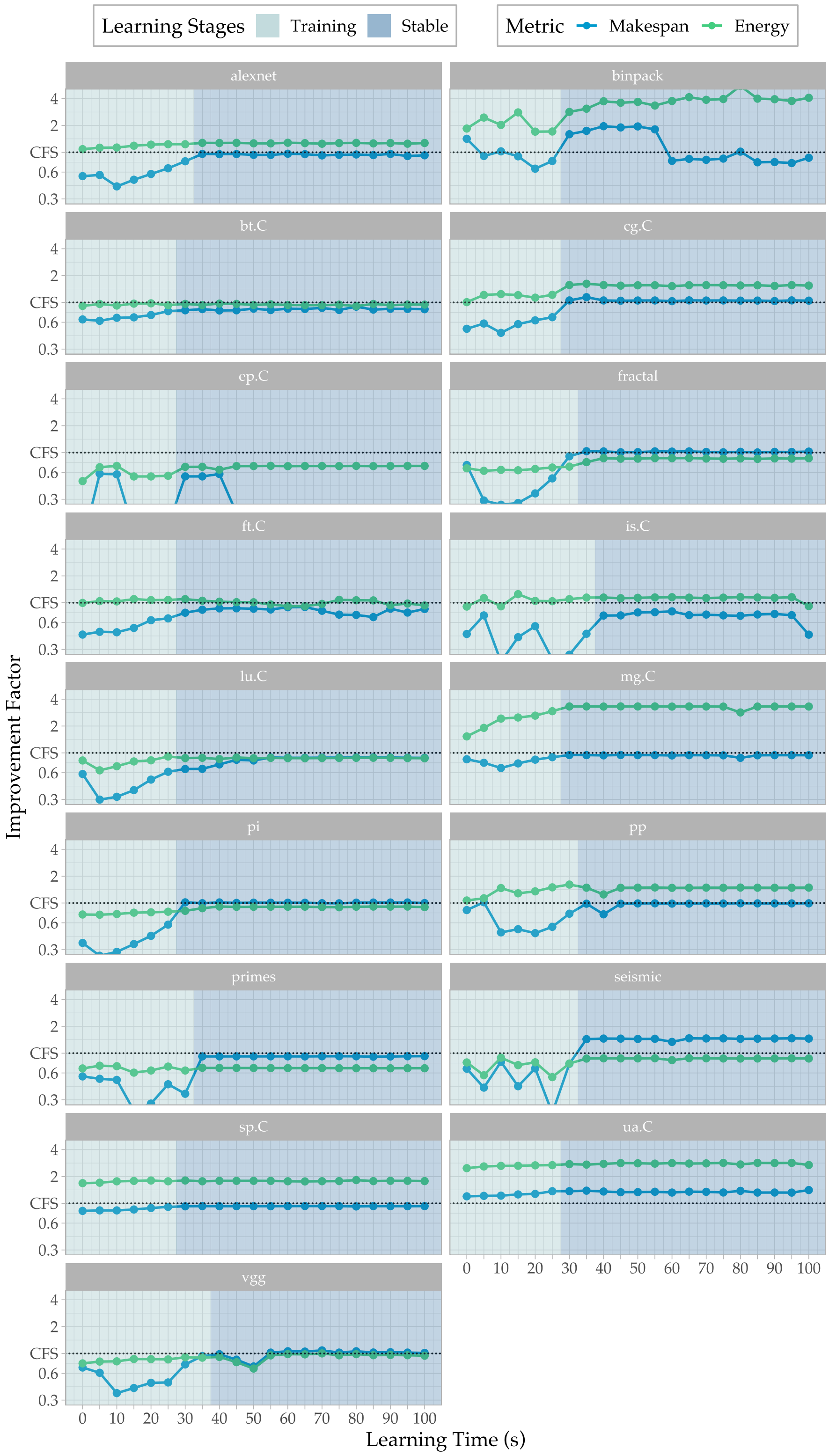


Figure 7.8: Relative improvement factor of HARP over CFS during the learning stage on single-application scenarios on the Intel Raptor Lake Core i9-13900K (higher y-value is better).

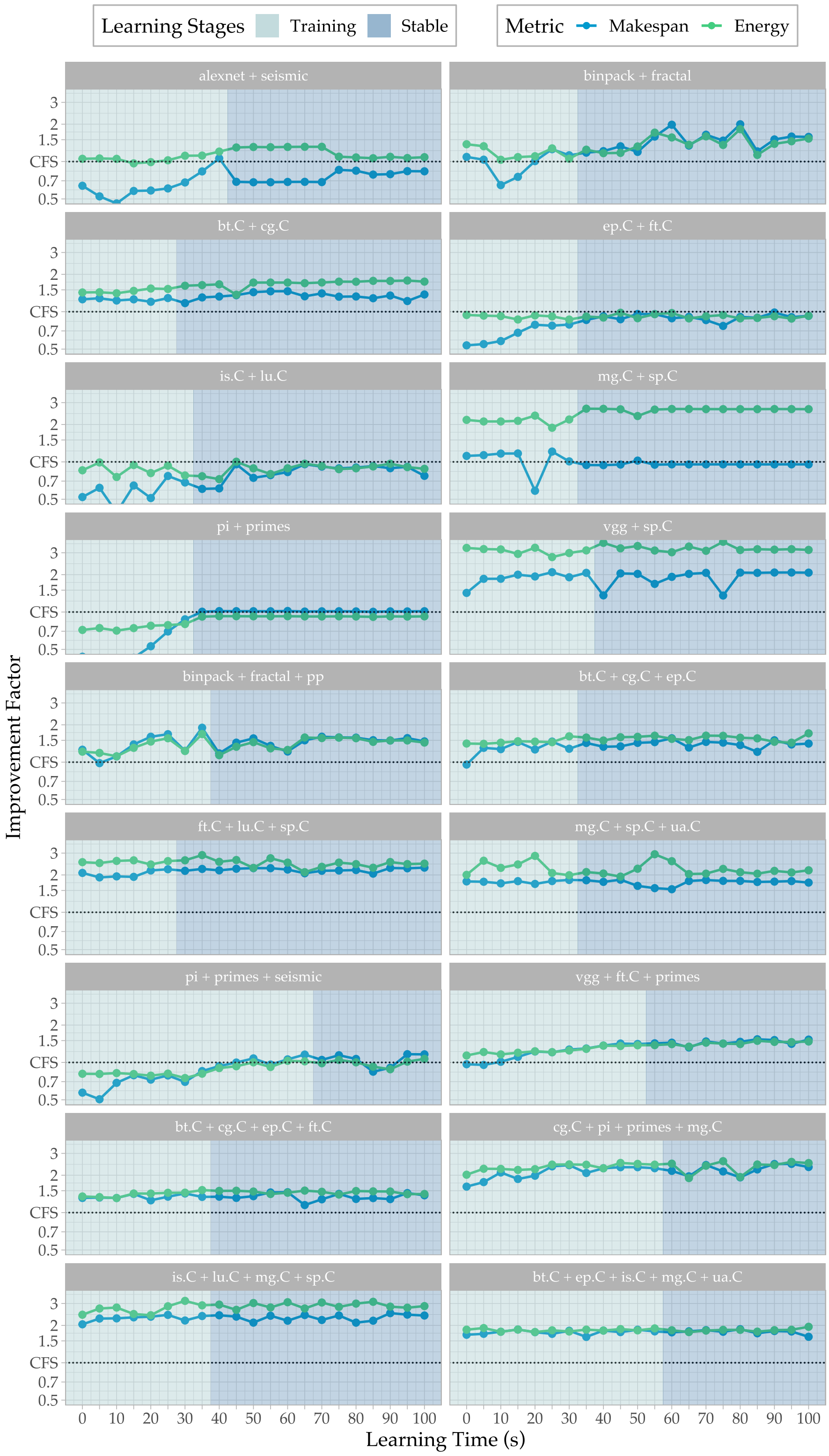


Figure 7.9: Relative improvement factor of HARP over CFS during the learning stage on multi-application scenarios on the Intel Raptor Lake Core i9-13900K (higher y-value is better).

stage. This behavior is likely due to misalignment with the regression model, causing HARP to continue exploring configurations to refine the Pareto front. The second outlier, `ep`, shows poor results even when compared to the main evaluation in Figure 7.6, despite being trained in isolation in both cases. This discrepancy may be attributed to the very short execution time of the application (2.43 s), which can result in limited and potentially unreliable measurements for certain configurations. As training progresses, the application is executed repeatedly, but the exploration heuristic may rely on these unreliable measurements when selecting subsequent operating points, introducing non-deterministic behavior into the learning process.

7.5.4.2 *Multi-Application Scenarios*

In multi-application scenarios (Figure 7.9), HARP shows a similar trend: fluctuating performance and energy efficiency during the training stage, followed by stabilization in the stable stage. Compared to single-application scenarios, fluctuations in the stable stage are more frequent. Stable operating points are achieved on average within 37.6 ± 9.3 s, slightly longer than in single-application scenarios due to limited resources available to each application, resulting in delays in the refinement stage.

Interestingly, as the number of applications increases, the performance and energy efficiency during the training stage approach those in the stable stage. For example, in scenarios with four or five applications, the results observed during training are already similar to the final stable outcomes. This behavior can be attributed to the limited search space available to each individual application under resource constraints. During the training stage, the exploration heuristic selects the configurations that utilize fewer resources, and the resource allocation algorithm in the stable stage also tends towards such configurations. Consequently, the outcomes of the training and stable stages become closely aligned.

Across both single- and multi-application scenarios, HARP effectively learns stable configurations that often outperforms CFS. However, specific limitations, such as the persistent fluctuations observed in `binpack` and the non-deterministic results for `ep`, highlight opportunities to refine the exploration heuristic, especially for applications with short execution times or irregular behaviors. Future work aims to address these issues by enhancing the exploration algorithm, ensuring more robust and consistent learning outcomes.

7.5.5 *Performance Overhead of HARP*

Resource management in modern operating systems must be swift and lightweight to avoid negatively impacting the system performance.

To evaluate the performance overhead introduced by HARP with its full functionality active, additional measurements were conducted on the Intel Raptor Lake system.

All application scenarios were executed using HARP with its full functionality enabled: runtime exploration, configuration communication, and, if supported by applications, utility updates. To isolate overhead, configuration assignment messages were ignored by `libharp`, leaving applications unadapted and thus scheduled as with the baseline CFS scheduler. This setup allows for an evaluation of the cumulative overhead introduced by HARP's components, including performance monitoring via `perf`, energy estimation and attribution, the resource selection algorithm, and communication between the RM and applications.

The evaluation indicates that HARP introduced less than 1 % overhead when managing a single application, and 2.5 % in multi-application scenarios. These minimal overheads are outweighed by the performance and energy efficiency gains achieved with HARP's resource allocation strategy and demonstrated in Section 7.5.2.

7.6 SYNOPSIS

This chapter introduces HARP, a resource management system designed for energy-aware applications on Heterogeneous Multi-core Architectures (HMAs). One of HARP's primary design goals is to enable seamless coordination between the RM and applications, allowing them to adapt their execution based on the allocated resources. At the same time, HARP RM receives the information about application behavior characteristics through pre-generated operating points or a real-time feedback of utility metrics. This two-way communication, facilitated via a simple and uniform interface, coordinates the adaptivity mechanisms of the RM and applications, addressing a core challenge of resource management in HMAs.

As heterogeneous processors become common even in general-purpose desktop and server domains, HARP is designed to operate effectively in such environments. It supports a broad spectrum of application models, from static to scalable, and even custom-specific ones with adaptivity features like reconfiguration options and algorithmic variations. Crucially, HARP does not require detailed knowledge of each application's adaptivity mechanisms. Instead, all applications interact with the RM through the same uniform interface. Extensions to `libharp` allow application-specific adaptivity features to be incorporated, as demonstrated with APN applications and a custom wrapper for TensorFlow applications.

Building on Hybrid Application Mapping (HAM) methodologies, HARP addresses its main limitation: reliance on pre-generated operating points. To overcome this, HARP integrates an online operating point exploration component that monitors application executions under different configurations and strategically guides the exploration process. This process leverages regression models to rapidly construct robust operating point models while ensuring minimal interference with concurrently running applications.

Evaluation on two heterogeneous systems shows that HARP significantly enhances application execution. Improvements of 16 % in execution time and 31 % in energy consumption were observed on the Intel Raptor Lake system, while the Odroid platform achieved enhancements of 12 % and 25 %, respectively. Notably, multi-application scenarios demonstrated better results than single-application scenarios, highlighting HARP's efficiency in dynamic and complex workloads.

The evaluation also highlights the rapidity of HARP's online operating point exploration, achieving stable configurations within 36.7 ± 9.3 s. Furthermore, HARP introduces minimal performance overhead, which is typically outweighed by the resulting gains.

In summary, this work demonstrates the value of a simple yet effective interface for communicating allocation decisions between applications and the OS, leveraging adaptivity at both the resource management and application levels. By aligning these mechanisms, HARP significantly enhances overall system performance and energy efficiency.

8

CONCLUSIONS AND OUTLOOK

The emergence of Heterogeneous Multi-core Architectures (HMAs) in both embedded and general-purpose systems presents new opportunities and challenges. These architectures combine heterogeneous cores with distinct performance-energy characteristics while sharing a common ISA. This design enables systems to optimize performance and energy efficiency for varying workloads and requirements. However, traditional execution models and resource allocation strategies may fail to fully leverage the potential of HMAs.

This thesis examined the adaptivity challenges posed by HMAs at two levels: resource management and application behavior. At the resource management level, dynamic allocation of processor cores to applications and the necessity to account for heterogeneous core types demand novel strategies. At the application level, applications must adapt their execution (e.g., parallelization degree or workload distribution) to align with the allocated resources, especially in the presence of heterogeneous cores.

While no single solution can fully resolve the adaptivity challenges of HMAs, this thesis proposed several approaches that enhance adaptivity in different scenarios, addressing both resource management and application behavior.

For resource management adaptivity, the thesis leverages the Hybrid Application Mapping (HAM) methodology, which has shown benefits in the embedded systems domain. HAM integrates design-time exploration to identify optimal configurations with runtime resource management that dynamically adapts resource allocation based current workloads. Building on these strengths, this thesis enhances HAM methodologies to further improve adaptivity.

First, Chapter 4 addressed the challenge of efficiently mapping parallel applications in dynamic real-time environments where workloads may change sporadically. The thesis introduced *spatio-temporal* multi-application mappings, an extension of conventional spatial mapping models, that can account for *foreseen* workload changes. Two algorithms, *MMKP-MDF* and *FFEMS*, were presented to generate spatio-temporal mappings within milliseconds. Additionally, a memetic algorithm, *STEM*, was introduced as a baseline to evaluate solutions generated by these rapid heuristics. These methods demonstrated significant improvements in resource utilization, energy efficiency, and Quality of Service (QoS), showcasing the effectiveness of spatio-temporal mapping strategies.

Second, Chapter 5 focused on baseband processing, a critical component of wireless communication systems, demanding high flexibility to handle heterogeneous and dynamic workloads. The chapter introduced a *domain-specific HAM methodology* tailored for the phase-sequential structure of baseband processing applications. A novel mapping algorithm was proposed to generate near-optimal operating points for hundreds of tasks within the baseband receiver kernel. Additionally, the runtime resource allocation algorithm was refined to leverage prior decisions, reducing overhead highly critical for baseband systems.

Beyond resource management, this thesis also focused on adaptivity challenges at the application level. Chapter 6 addressed the limitations of Kahn Process Network (KPN), a widely used dataflow MoC in embedded systems for signal processing and streaming application. To enhance adaptivity, the chapter introduced *Adaptive Process Network* (APN), an extension of KPNs that enables malleable parallelization — adapting to varying resource availability — and dynamic workload distribution to address system heterogeneity, while preserving the key feature of *determinism*. The chapter also introduced the Dynamic Process Manager (DPM) library, which provides runtime support for APN applications, a programmer interface for developing such applications, and a configuration manager to facilitate application adaptations and communication with external resource managers.

The final contribution of this thesis addressed the challenge of *coordinated adaptivity at both resource management and application levels*. Chapter 7 introduced *Heterogeneity-aware Adaptive Resource Partitioning* (HARP), a Linux-integrated resource management framework that enables seamless coordination between the RM and applications. A key innovation of HARP is its *two-way communication interface*, which enables applications to both receive resource allocation decisions — allowing dynamic adaptation to assigned resources — and provide real-time feedback on their execution behavior to the RM. This simple and uniform communication interface allows HARP to support diverse application models, including OpenMP, Intel TBB, TensorFlow and dataflow applications.

HARP marks a significant step towards supporting general-purpose systems, where workloads are dynamic and unpredictable. While building on HAM principles, it overcomes a critical limitation: reliance on pre-generated operating points. To address this, HARP integrates an *online operating point exploration* component that monitors application behavior under various configurations. By leveraging a preselected regression model, the exploration process strategically guides measurements to rapidly build robust operating point models with minimal interference to concurrently running applications.

This thesis demonstrated that adaptivity challenges in HMAs can be effectively addressed through coordinated strategies at both resource management and application levels. While the proposed approaches significantly improve energy efficiency and performance, several opportunities for future work remain.

Incorporating Dynamic Voltage-Frequency Scaling into HAM

The resource management algorithms presented in this thesis primarily focus on spatial (and spatio-temporal) resource allocation to optimize energy efficiency, without considering core frequency adjustments. Integrating Dynamic Voltage Frequency Scaling (DVFS) into HAM presents an opportunity for additional energy savings, although this is not a novel direction. Prior works have explored methods such as pruning operating points across the spatial and frequency components [140] and applying DVFS during runtime resource management (e.g., [15, 60, 184, 211]). However, combining DVFS with online exploration of operating points in HARP presents an interesting research challenge.

Addressing Shared Resource Contention

The resource management algorithms presented in this thesis do not account for contention of shared resources such as memory components. As a result, actual application performance in multi-application scenarios may deviate from the behavior observed during DSE. Accounting for contention for shared resources within HAM methodologies requires advanced techniques, such as Memguard [215] and PALLOC [214], to accurately model these effects and ensure predictable performance in dynamic workloads.

Application Behavior Descriptions across Platforms

While HARP's online operating point exploration effectively handles unforeseen applications, evaluations demonstrated that pre-generated operating points obtained at design time yield superior results. With the adoption of the HARP approach, manually generating these descriptions across diverse hardware platforms is a nontrivial task for application developers. Developing methodologies to approximate application behavior across different hardware platforms would simplify this process and encourage broader adoption of the HARP approach.

Design Space Exploration for Adaptive Process Networks

The deterministic nature of the APN application model makes it well-suited for model-based analysis, such as trace-based simulations (Section 2.3.3) and Design Space Exploration (DSE) in frameworks like Mocasin (Section 2.5). To enable DSE for APN applications, a dedicated workflow must be developed, integrating the DPM library with Mocasin.

This workflow would begin with process trace generation. By leveraging the functional equivalence of replicas in parallel regions, traces could be generated using a single replica, simplifying this step. These traces would then be fed to Mocasin, which would require extensions to support the APN application model. In particular, its simulation component needs to be extended to accurately emulate parallel channels and distribution strategies.

The final step of the workflow would involve mapping generation. APN-specific mapping algorithms would need to determine the optimal number of replicas, in addition to conventional process-to-core assignments. Both metaheuristic and heuristic approaches could be explored for this purpose.

Broader Opportunities for Application-Level Adaptivity

This thesis primarily focuses on application adaptivity within the APN application model. Additionally, Chapter 7 demonstrates the parallelization adaptivity for OpenMP and Intel TBB. Further research could focus on extending the adaptivity mechanisms to other application models, enabling them to integrate with the frameworks proposed in this thesis.

For example, Multi-Alternative Process Network (mAPN) supporting algorithm-switching capabilities could dynamically switch algorithms within processes or even process subnetworks [26, 166]. This could benefit, for instance, real-time streaming applications such as video encoding/decoding, which could adjust codec complexity or resolution based on available resources and network conditions. mAPN could leverage the DPM runtime library and the HARP resource management framework to dynamically select the algorithms best suited for current resource allocations and workload characteristics.

In the domain of machine learning, adaptivity mechanisms for Convolutional Neural Networks (CNNs), such as model pruning and early exits [102, 113, 191], provide opportunities for resource-aware execution. These methods allow CNNs to trade off between performance, energy efficiency, and accuracy based on available resources and could be supported within HARP for dynamic optimization.

The field of approximate computing offers further opportunities for exploration. Techniques include software-level approximations such as reduced-precision algorithms, loop perforation, and algorithm-level simplifications [132]. Additionally, specialized accelerators for reduced-precision arithmetic [13] or task-specific units, such as Tensor Processing Unit (TPU) [90], further enhance the potential of approximate computing. Extending HARP to support such hardware would also represent a substantial addition to the presented methodology.

Managing Evolving Application Phases

The presented approaches, particularly within HARP, assume constant application behavior throughout execution. However, some applications exhibit *evolving* phases, such as transitions between sequential and parallel stages, or compute-intensive and memory-intensive parts. Two potential directions could be explored to address this challenge.

First, applications could explicitly communicate their phases and associated performance-energy characteristics, as well as phase-switch events, via the communication interface presented in HARP. For instance, in OpenMP, entry and exit of parallel loops could be detected by hooking to `GOMP_parallel` function. The main limitation of this approach is the reliance on applications to provide such information.

Second, for applications that do not provide explicit phase information, more advanced techniques would be needed to dynamically detect the phase transitions. Additionally, to avoid repeatedly regenerating operating points, the system should quickly recognize previously explored phases and reuse their configurations.

GLOSSARY

ACRONYMS

API	Application Programming Interface
APN	Adaptive Process Network
ASIC	Application-Specific Integrated Circuit
ASIP	Application-Specific Instruction-Set Processor
BBU	Baseband Unit
BDF	Boolean Dataflow
CFS	Completely Fair Scheduler
CGRA	Coarse-Grained Reconfigurable Array
CNN	Convolutional Neural Network
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit
cRAN	Cloud RAN
CSDF	Cyclo-Static Dataflow
DAG	Directed Acyclic Graph
DDF	Dynamic Dataflow
DPM	Dynamic Process Manager
DSE	Design Space Exploration
DSP	Digital Signal Processor
DSRH	Domain-Specific Reconfigurable Hardware
DVFS	Dynamic Voltage Frequency Scaling
EA	Evolutionary Algorithm
EAS	Energy-Aware Scheduler
EDF	Earliest Deadline First
EDP	Energy-Delay Product
EFT	Earliest Finishing Time
EMA	Exponential Moving Average
eMBB	Enhanced Mobile Broadband
EPN	Expandable Process Network
FCFS	First-Come, First-Served
FFEMS	Fast Flexible Energy-Minimizing Scheduler
FFT	Fast Fourier Transform

FIFO	First In, First Out
FPGA	Field-Programmable Gate Array
GA	Genetic Algorithm
GPP	General-Purpose Processor
GPU	Graphics Processing Unit
HAM	Hybrid Application Mapping
HARP	Heterogeneity-aware Adaptive Resource Partitioning
HMA	Heterogeneous Multi-core Architecture
HSDF	Homogeneous SDF
IFFT	Inverse FFT
IGD	Inverted Generational Distance
ILP	Integer Linear Programming
IoT	Internet of Things
IPC	Instructions Per Cycle
IPS	Instructions Per Second
ISA	Instruction Set Architecture
ITD	Intel's Thread Director
KPN	Kahn Process Network
LTE	Long-Term Evolution
MA	Memetic Algorithm
mAPN	Multi-Alternative Process Network
MDF	Maximum Difference First
MILP	Mixed-Integer Linear Programming
MMKP	Multiple-choice Multidimensional Knapsack Problem
mMTC	Massive Machine-Type Communication
MoC	Model of Computation
MPSoC	Multi-Processor Systems-on-Chip
NN	Neural Network
NoC	Network-on-Chip
OS	Operating System
PE	Processing Element
PELT	Per-Entity Load Tracking
PRB	Physical Resource Block
PN	Process Network
QoS	Quality of Service
RAN	Radio Access Network

RAPL	Running Average Power Limit
RDF	Reconfigurable Dataflow
RM	Resource Manager
RRH	Remote Radio Head
SA	Simulated Annealing
SDF	Synchronous Dataflow
SDR	Software-Defined Radio
SIMD	Single Instruction Multiple Data
SMT	Simultaneous Multithreading
SVM	Support Vector Machine
STEM	Spatio-Temporal Evolutionary Mapping
TLP	Thread-Level Parallelism
TDP	Thermal Design Power
TPU	Tensor Processing Unit
UE	User Equipment
URLLC	Ultra-Reliable Low-Latency Communication
vRAN	Virtualized RAN

NOTATION

$X^C\langle\ldots\rangle$	Object X with attributes; C is an optional context
$X[a]$	Attribute a of object X
Ω	Processor type, $\Omega := \Omega\langle \text{isa}, \text{cm}, X\rangle$
(Ω_i)	Sequence of processor types, $(\Omega_i) = (\Omega_1, \ldots, \Omega_{\lvert(\Omega_i)\rvert})$
ψ	Processing element, $\psi := \psi\langle\Omega\rangle$
$\mathcal{P}$	Platform, $\mathcal{P} := \mathcal{P}\langle\Psi, \mathcal{IC}\rangle$
$\mathcal{P}[\Psi]$	Set of all processing elements in platform $\mathcal{P}$
$\mathcal{P}[\omega_i]$	Number of processing elements of type Ω_i in platform $\mathcal{P}$
$\mathcal{P}[\vec{\omega}]$	Resource vector of platform $\mathcal{P}$
A	Application, $A := A\langle\mathcal{M}, \Gamma, \mathcal{V}\rangle$
$A[\mathcal{M}]$	Application model
$A[\Gamma]$	Set of all application configurations
$A[\mathcal{V}]$	Set of all application primitives
$A[\mathcal{V}_\gamma]$	Application primitives active under configuration γ

o	Operating point, $o := o^A\langle \gamma, \mu, \eta \rangle$
$o[\mu]$	Mapping, $\mu : A[V_\gamma] \to \mathcal{P}[\Psi]$
$o[\eta]$	Non-functional characteristics, $\eta := \eta\langle \tau, \varepsilon \rangle$ or $\eta := \eta\langle \upsilon, p \rangle$
$\eta[\tau]$	Execution time
$\eta[\varepsilon]$	Energy consumption
$\eta[\upsilon]$	Utility
$\eta[p]$	Power consumption
$\mu[\Psi]$	Resource allocation of μ
$\mu[\vec{\omega}]$	Resource vector of μ
$\vec{F}(o)$	Objective values of operating point o
$\vec{u} \preceq \vec{v}$	Domination: $\vec{u}$ dominates $\vec{v}$
$\mathcal{O}^A_{\text{OPT}}$	Pareto Optimal Set of operating points for application A
$\mathcal{F}^A_{\text{OPT}}$	Pareto Front
σ	Job request, $\sigma := \sigma\langle A, t_{\text{arr}}, \theta, \rho \rangle$
$\sigma[t_{\text{ARR}}]$	Arrival time
$\sigma[\theta]$	Relative deadline
$\sigma[\rho]$	Current progress ratio, $\rho \in [0, 1]$
Σ	Set of all job requests.
M	Spatial multi-application mapping
$M[\sigma]$	Assignment of the job request σ, $M[\sigma] := M[\sigma]\langle o \rangle$
K	Spatio-temporal multi-application mapping
$K[M_i]$	Multi-application mapping at the i-th mapping segment
$K[\delta_i]$	Duration the i-th mapping segment
$K[\Delta_n]$	Cumulative duration of first n segments
$K[\varepsilon]$	Energy consumption of the spatio-temporal multi-application mapping
$\mathcal{M}^{\text{PTG}}$	Phased task graph, $\mathcal{M}^{\text{PTG}} := \mathcal{M}^{\text{PTG}}\langle \Phi \rangle$, where $\Phi = (\phi_1, \ldots, \phi_{\lvert\Phi\rvert})$
ϕ	Phase in a task graph with phase-sequential structure, $\phi := \phi\langle \Pi, \lambda \rangle$

$\phi[\Pi]$	Ordered set of tasks connected in sequence, $\Pi := \left(\pi_1, \ldots, \pi_{\lvert\Pi\rvert}\right)$
$\phi[\lambda]$	Parallelization factor of the phase
$\phi[\pi_i^k]$	The *i*-th process in the *k*-th replica of the phase ϕ
π	Task with latency τ and energy consumption ξ for each resource type Ω_i, $\pi := \pi\langle\vec{\tau}, \vec{\xi}\rangle$

LIST OF FIGURES

Figure 1.1	Samsung Exynos 5422 CPU implementing the Arm's big.LITTLE architecture	3
Figure 1.2	Labeled die shot of an Intel Raptor Lake Core i9-13900K	3
Figure 1.3	Adaptivity challenges at resource and application levels	5
Figure 2.1	Mapping, scheduling, and spatio-temporal mapping of two dataflow applications	13
Figure 2.2	JPEG Image Encoding application as a KPN graph	22
Figure 2.3	Overview of Hybrid Application Mapping	26
Figure 2.4	Overview of the Mocasin framework architecture	36
Figure 4.1	Gantt charts for resource management scenarios in the motivational example	50
Figure 4.2	Overview of multi-application mapping strategies	51
Figure 4.3	Acceptance rate of different algorithms for test cases with tight deadlines	58
Figure 4.4	Monotonic distribution curves of the relative energy consumption	59
Figure 4.5	Box plots and average values summarizing the mapping overhead for different algorithms	60
Figure 4.6	STEM algorithm flow	61
Figure 4.7	Crossover operators in STEM	63
Figure 4.8	Acceptance rates of resource management algorithms across varying job counts	71
Figure 4.9	Relative energy consumption of different resource management algorithms	71
Figure 4.10	Monotonic distribution curves of the relative energy consumption across resource management algorithms	73
Figure 4.11	Runtime overhead of resource management algorithms	73
Figure 5.1	Block diagram of a baseband receiver	78
Figure 5.2	Task graph with a phase-sequential structure	79
Figure 5.3	Sample phase of the baseband receiver with the execution latencies	83
Figure 5.4	Step-by-step visualization of the fast mapping algorithm	83
Figure 5.5	Validation of the estimated execution time and energy consumption values	89

Figure 5.6 Comparison of design-time mapping algorithms for the baseband processing application 90
Figure 5.7 Successfully executed UEs and energy efficiency on Odroid-XU4 92
Figure 5.8 Successfully executed UEs and energy efficiency on a virtual platform with accelerators 93
Figure 5.9 Relative energy and utilization while controlling for success rate 93
Figure 5.10 Relative total energy consumption while controlling for success rate on virtual platforms without accelerators 94
Figure 5.11 Average runtime overhead per RM activation . 95
Figure 6.1 KPN application calculating the Mandelbrot Set 98
Figure 6.2 Gantt charts illustrating static and dynamic workload scheduling 99
Figure 6.3 APN application with a parallel region 101
Figure 6.4 Distribution strategies for split-interleave primitives 103
Figure 6.5 DPM library overview 105
Figure 6.6 Iterations per point in the Mandelbrot set . . . 110
Figure 6.7 Throughput on Odroid-XU4 across core configurations 111
Figure 6.8 Effect of contention on performance 113
Figure 6.9 Dynamic reallocation of resources at runtime . 113
Figure 7.1 Performance and power consumption of two applications on Intel Raptor Lake 117
Figure 7.2 HARP system overview 120
Figure 7.3 Control flow between application and HARP manager 121
Figure 7.4 Evaluation of regression models for runtime exploration of operating points 127
Figure 7.5 Integration of runtime exploration of operating points with resource allocation 129
Figure 7.6 Relative improvement of HARP over CFS on Intel Raptor Lake 134
Figure 7.7 Relative improvement of HARP over EAS on Odroid-XU3-E 136
Figure 7.8 Relative improvement of HARP over CFS during learning on single-application scenarios . . 138
Figure 7.9 Relative improvement of HARP over CFS during learning on multi-application scenarios . . 139

LIST OF TABLES

Table 4.1 Job parameters for the motivational example . 48
Table 4.2 Operating points for applications in the motivational example 49
Table 4.3 Number of test cases by jobs and deadline level 57
Table 4.4 Relative energy consumption across number of jobs and deadline levels 59
Table 4.5 Factor range bounds for workload generation . 68
Table 4.6 Acceptance, energy, and overhead of RM algorithms on `4B4L` 70
Table 4.7 Acceptance, energy, and overhead of RM algorithms on `8B8L` 70
Table 5.1 Platform core frequency and power characteristics 87
Table 5.2 Overview of LTE trace characteristics 88

LIST OF ALGORITHMS

Algorithm 4.1 Main Procedure of the MMKP-MDF Algorithm 54
Algorithm 4.2 ConstructSTM with EDF policy 55
Algorithm 4.3 Main Procedure of the FFEMS Algorithm . . . 65
Algorithm 4.4 GenerateJobTail procedure 66
Algorithm 5.1 Phased-Fast algorithm 81
Algorithm 5.2 MMKP-MDF with prior solution reuse 85

LIST OF LISTINGS

Listing 6.1 Building APN topology with DPM 106
Listing 6.2 KPN process definition with DPM 107
Listing 6.3 SDF process definition with DPM 107
Listing 6.4 Sample YAML configuration file 109

BIBLIOGRAPHY

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. "TensorFlow: a system for large-scale machine learning." In: *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*. OSDI'16. Savannah, GA, USA: USENIX Association, 2016, pp. 265–283. ISBN: 9781931971331.

[2] Mohiuddin Ahmed, Raihan Seraj, and Syed Mohammed Shamsul Islam. "The k-means algorithm: A comprehensive survey and performance evaluation." In: *Electronics* 9.8 (2020). ISSN: 2079-9292. DOI: `10.3390/electronics9081295`. URL: `https://www.mdpi.com/2079-9292/9/8/1295`.

[3] Waheed Ahmed, Muhammad Shafique, Lars Bauer, and Jörg Henkel. "Adaptive resource management for simultaneous multitasking in mixed-grained reconfigurable multi-core processors." In: *Proceedings of the Seventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. CODES+ISSS '11. Taipei, Taiwan: Association for Computing Machinery, 2011, pp. 365–374. ISBN: 9781450307154. DOI: `10.1145/2039370.2039426`.

[4] Adrian Alexandrescu, Ioan Agavriloaei, and Mitică Craus. "A Genetic Algorithm for mapping tasks in heterogeneous computing systems." In: *15th International Conference on System Theory, Control and Computing*. 2011, pp. 1–6.

[5] Ali Alnoman and Alagan Anpalagan. "Towards the fulfillment of 5G network requirements: technologies and challenges." In: *Telecommunication Systems* 65.1 (May 2017), pp. 101–116. ISSN: 1018-4864. DOI: `10.1007/s11235-016-0216-9`.

[6] Murali Annavaram, Ed Grochowski, and John Shen. "Mitigating Amdahl's law through EPI throttling." In: *32nd International Symposium on Computer Architecture (ISCA'05)*. 2005, pp. 298–309. DOI: `10.1109/ISCA.2005.36`.

[7] Apple. *Apple unleashes M1*. Available at: `https://www.apple.com/newsroom/2020/11/apple-unleashes-m1/` [Online; accessed 08-April-2025]. Nov. 2020.

[8] Arm Limited. *big. LITTLE technology: The future of mobile*. White Paper. Available at: `https://armkeil.blob.core.windows.net/developer/Files/pdf/white-paper/big-little-technology-the-future-of-mobile.pdf` [Online; accessed 08-April-2025]. 2013.

[9] Arm Limited. *Arm DynamIQ Shared Unit 120 Technical Reference Manual*. Available at: `https://developer.arm.com/documentation/102547/0100` [Online; accessed 08-April-2025]. 2023.

[10] Giuseppe Ascia, Vincenzo Catania, and Maurizio Palesi. "Multi-objective mapping for mesh-based NoC architectures." In: *Proceedings of the 2nd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. CODES+ISSS '04. Stockholm, Sweden: Association for Computing Machinery, 2004, pp. 182–187. ISBN: 158113 9373. DOI: `10.1145/1016720.1016765`.

[11] D.H. Bailey, E. Barszcz, J.T. Barton, D.S. Browning, R.L. Carter, L. Dagum, R.A. Fatoohi, P.O. Frederickson, T.A. Lasinski, R.S. Schreiber, H.D. Simon, V. Venkatakrishnan, and S.K. Weeratunga. "The Nas Parallel Benchmarks." In: *Int. J. High Perform. Comput. Appl.* 5.3 (Sept. 1991), pp. 63–73. ISSN: 1094-3420. DOI: `10.1177/109434209100500306`.

[12] Felice Balarin, Massimiliano Chiodo, Paolo Giusto, Harry Hsieh, Attila Jurecska, Luciano Lavagno, Claudio Passerone, Alberto Sangiovanni-Vincentelli, Ellen Sentovich, Kei Suzuki, and Bassam Tabbara. *Hardware-software co-design of embedded systems: the POLIS approach*. USA: Kluwer Academic Publishers, 1997. ISBN: 0792399366.

[13] A. S. Baroughi, S. Huemer, H. S. Shahhoseini, and N. TaheriNejad. "AxE: An Approximate-Exact Multi-Processor System-on-Chip Platform." In: *2022 25th Euromicro Conference on Digital System Design (DSD)*. 2022, pp. 60–66. DOI: `10.1109/DSD57027.2022.00018`.

[14] Luiz André Barroso and Urs Hölzle. "The Case for Energy-Proportional Computing." In: *Computer* 40.12 (2007), pp. 33–37. DOI: `10.1109/MC.2007.443`.

[15] Karunakar R. Basireddy, Amit Kumar Singh, Bashir M. Al-Hashimi, and Geoff V. Merrett. "AdaMD: Adaptive Mapping and DVFS for Energy-Efficient Heterogeneous Multicores." In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39.10 (2020), pp. 2206–2217. DOI: `10.1109/TCAD.2019.2935065`.

[16] Michela Becchi and Patrick Crowley. "Dynamic thread assignment on heterogeneous multiprocessor architectures." In: *Proceedings of the 3rd Conference on Computing Frontiers*. CF '06. Ischia, Italy: Association for Computing Machinery, 2006, pp. 29–40. ISBN: 1595933026. DOI: `10.1145/1128022.1128029`.

[17] Sandro Belfanti, Christoph Roth, Michael Gautschi, Christian Benkeser, and Qiuting Huang. "A 1Gbps LTE-advanced turbo-decoder ASIC in 65nm CMOS." In: *2013 Symposium on VLSI Circuits*. Jan. 2013, pp. C284–C285. ISBN: 978-1-4673-5531-5.

[18] L. Benini and G. De Micheli. "Networks on chips: a new SoC paradigm." In: *Computer* 35.1 (2002), pp. 70–78. DOI: `10.1109/2.976921`.

[19] C. H. (Kees) van Berkel. "Multi-core for mobile phones." In: *Proceedings of the Conference on Design, Automation and Test in Europe*. DATE '09. Nice, France: European Design and Automation Association, 2009, pp. 1260–1265. ISBN: 9783981080155.

[20] Carlos Bilbao, Juan Carlos Saez, and Manuel Prieto-Matias. "Flexible system software scheduling for asymmetric multicore systems with PMCSched: A case for Intel Alder Lake." In: *Concurrency and Computation: Practice and Experience* 35.25 (2023), e7814. DOI: `10.1002/cpe.7814`.

[21] Greet Bilsen, Marc Engels, Rudy Lauwereins, and Jean Peperstraete. "Cycle-static dataflow." In: *IEEE Transactions on Signal Processing* 44.2 (1996), pp. 397–408. DOI: `10.1109/78.485935`.

[22] Enrico Bini, Giorgio Buttazzo, Johan Eker, Stefan Schorr, Raphael Guerra, Gerhard Fohler, Karl-Erik Arzen, Vanessa Romero, and Claudio Scordino. "Resource Management on Multicore Systems: The ACTORS Approach." In: *IEEE Micro* 31.3 (2011), pp. 72–81. DOI: `10.1109/MM.2011.1`.

[23] Tobias Blickle and Lothar Thiele. "A Comparison of Selection Schemes Used in Evolutionary Algorithms." In: *Evolutionary Computation* 4.4 (Dec. 1996), pp. 361–394. ISSN: 1063-6560. DOI: `10.1162/evco.1996.4.4.361`.

[24] Alessio Bonfietti, Luca Benini, Michele Lombardi, and Michela Milano. "An efficient and complete approach for throughput-maximal SDF allocation and scheduling on multi-core platforms." In: *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*. 2010, pp. 897–902. DOI: `10.1109/DATE.2010.5456924`.

[25] Hasna Bouraoui, Jeronimo Castrillon, and Chadlia Jerad. "Comparing Dataflow and OpenMP Programming for Speaker Recognition Applications." In: *Proceedings of the 10th and 8th Workshop on Parallel Programming and Run-Time Management Techniques for Many-Core Architectures and Design Tools and Architectures for*

Multicore Embedded Computing Platforms (PARMA-DITAM 2019). Valencia, Spain: Association for Computing Machinery, 2019. ISBN: 9781450363211. DOI: `10.1145/3310411.3310417`.

[26] Hasna Bouraoui, Chadlia Jerad, and Jeronimo Castrillon. "Towards Adaptive Multi-Alternative Process Network." In: *12th Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures and 10th Workshop on Design Tools and Architectures for Multicore Embedded Computing Platforms (PARMA-DITAM 2021)*. Ed. by João Bispo, Stefano Cherubin, and José Flich. Vol. 88. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, 1:1–1:11. ISBN: 978-3-95977-181-8. DOI: `10.4230/OASIcs.PARMA-DITAM.2021.1`.

[27] Björn B. Brandenburg, John M. Calandrino, and James H. Anderson. "On the Scalability of Real-Time Scheduling Algorithms on Multicore Platforms: A Case Study." In: *2008 Real-Time Systems Symposium*. 2008, pp. 157–169. DOI: `10.1109/RTSS.2008.23`.

[28] C. Brandolese, W. Fornaciari, F. Salice, and D. Sciuto. "Source-level execution time estimation of C programs." In: *Ninth International Symposium on Hardware/Software Codesign. CODES 2001 (IEEE Cat. No.01TH8571)*. 2001, pp. 98–103. DOI: `10.1145/371636.371694`.

[29] Alexander J. Branover, Benjamin Tsien, and Elliot H. Mednick. "Method of Task Transition Between Heterogeneous Processors." 20210173715. US Patent. June 2021. URL: `https://www.freepatentsonline.com/y2021/0173715.html`.

[30] Eduardo Wenzel Brião, Daniel Barcelos, and Flavio Rech Wagner. "Dynamic Task Allocation Strategies in MPSoC for Soft Real-time Applications." In: *2008 Design, Automation and Test in Europe*. 2008, pp. 1386–1389. DOI: `10.1109/DATE.2008.4484934`.

[31] David Brooks, Vivek Tiwari, and Margaret Martonosi. "Wattch: a framework for architectural-level power analysis and optimizations." In: *Proceedings of 27th International Symposium on Computer Architecture (IEEE Cat. No.RS00201)*. 2000, pp. 83–94.

[32] Simone Casale Brunet, Marco Mattavelli, and Jorn W. Janneck. "Buffer optimization based on critical path analysis of a dataflow program design." In: *2013 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2013, pp. 1384–1387. DOI: `10.1109/ISCAS.2013.6572113`.

[33] Joseph Tobin Buck. "Scheduling dynamic dataflow graphs with bounded memory using the token flow model." AAI9431898. PhD thesis. EECS Department, University of California, Berkeley, 1993.

[34] Nishant Budhdev, Mun Choon Chan, and Tulika Mitra. "PR3: Power Efficient and Low Latency Baseband Processing for LTE Femtocells." In: *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*. 2018, pp. 2357–2365. DOI: `10.1109/INFOCOM.2018.8486276`.

[35] Nishant Budhdev, Mun Choon Chan, and Tulika Mitra. *Iso-RAN: Isolation and Scaling for 5G RANvia User-Level Data Plane Virtualization*. 2020. arXiv: `2003.01841 [cs.NI]`.

[36] Ewerson Luiz de Souza Carvalho, Ney Laert Vilar Calazans, and Fernando Gehm Moraes. "Dynamic Task Mapping for MP-SoCs." In: *IEEE Design & Test of Computers* 27.5 (2010), pp. 26–35. DOI: `10.1109/MDT.2010.106`.

[37] Jeronimo Castrillon, Karol Desnos, Andrés Goens, and Christian Menard. "Dataflow Models of Computation for Programming Heterogeneous Multicores." In: *Handbook of Computer Architecture*. Ed. by Anupam Chattopadhyay. Singapore: Springer Nature Singapore, Jan. 2023, pp. 1–40. ISBN: 978-981-15-6401-7. DOI: `10.1007/978-981-15-6401-7_45-2`.

[38] Jeronimo Castrillon, Rainer Leupers, and Gerd Ascheid. "MAPS: Mapping Concurrent Dataflow Applications to Heterogeneous MPSoCs." In: *IEEE Transactions on Industrial Informatics* 9.1 (2013), pp. 527–545. ISSN: 1551-3203. DOI: `10.1109/TII.2011.2173941`.

[39] Jeronimo Castrillon, Stefan Schürmans, Anastasia Stulova, Weihua Sheng, Torsten Kempf, Rainer Leupers, Gerd Ascheid, and Heinrich Meyr. "Component-based Waveform Development: The Nucleus Tool Flow for Efficient and Portable Software Defined Radio." In: *Analog Integrated Circuits and Signal Processing* 69.2 (Dec. 2011), pp. 173–190. ISSN: 1573-1979. DOI: `10.1007/s10470-011-9670-1`.

[40] Jeronimo Castrillon, Andreas Tretter, Rainer Leupers, and Gerd Ascheid. "Communication-aware mapping of KPN applications onto heterogeneous MPSoCs." In: *Proceedings of the 49th Annual Design Automation Conference*. DAC '12. San Francisco, California: Association for Computing Machinery, 2012, pp. 1266–1271. ISBN: 9781450311991. DOI: `10.1145/2228360.2228597`.

[41] Jeronimo Castrillon, Ricardo Velásquez, Anastasia Stulova, Weihua Sheng, Jianjiang Ceng, Rainer Leupers, Gerd Ascheid, and Heinrich Meyr. "Trace-based KPN Composability Analysis for Mapping Simultaneous Applications to MPSoC Platforms." In: *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*. Dresden, Germany, Mar. 2010, pp. 753–758. ISBN: 978-3-9810801-6-2. DOI: `10.1109/DATE.2010.5456950`.

[42] Jeronimo Castrillon Mazo and Rainer Leupers. *Programming Heterogeneous MPSoCs: Tool Flows to Close the Software Productivity Gap*. Cham: Springer, 2014. ISBN: 978-3-31900-674-1. DOI: `10.1007/978-3-319-00675-8`.

[43] Weijia Che and Karam S. Chatha. "Unrolling and retiming of stream applications onto embedded multicore processors." In: *Proceedings of the 49th Annual Design Automation Conference*. DAC '12. San Francisco, California: Association for Computing Machinery, 2012, pp. 1272–1277. ISBN: 9781450311991. DOI: `10.1145/2228360.2228598`.

[44] Aleksandra Checko, Henrik L. Christiansen, Ying Yan, Lara Scolari, Georgios Kardaras, Michael S. Berger, and Lars Dittmann. "Cloud RAN for Mobile Networks—A Technology Overview." In: *IEEE Communications Surveys & Tutorials* 17.1 (2015), pp. 405–426. DOI: `10.1109/COMST.2014.2355255`.

[45] Liang Chen, Thomas Marconi, and Tulika Mitra. "Online scheduling for multi-core shared reconfigurable fabric." In: *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2012, pp. 582–585. DOI: `10.1109/DATE.2012.6176537`.

[46] Tim Chen. *Enable Cluster Scheduling for x86 Hybrid CPUs*. Available at: `https://lore.kernel.org/lkml/cover.1688770494.git.tim.c.chen@linux.intel.com/` [Online; accessed 08-April-2025]. July 2023.

[47] Junchul Choi, Hyunok Oh, Sungchan Kim, and Soonhoi Ha. "Executing synchronous dataflow graphs on a SPM-based multicore architecture." In: *DAC Design Automation Conference 2012*. 2012, pp. 664–671.

[48] Chen-Ling Chou and Radu Marculescu. "User-Aware Dynamic Task Allocation in Networks-on-Chip." In: *2008 Design, Automation and Test in Europe*. 2008, pp. 1232–1237. DOI: `10.1109/DATE.2008.4484847`.

[49] Carlos A. Coello Coello, Gary B. Lamont, and David A. Van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Genetic and Evolutionary Computation. Springer US, 2007. ISBN: 978-0-387-33254-3. DOI: `10.1007/978-0-387-36797-2`.

[50] Carlos A. Coello Coello and Margarita Reyes Sierra. "A Study of the Parallelization of a Coevolutionary Multi-objective Evolutionary Algorithm." In: *MICAI 2004: Advances in Artificial Intelligence*. Ed. by Raúl Monroy, Gustavo Arroyo-Figueroa, Luis Enrique Sucar, and Humberto Sossa. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 688–697. ISBN: 978-3-540-24694-7. DOI: `10.1007/978-3-540-24694-7_71`.

[51] Maxime Colmant, Mascha Kurpicz, Pascal Felber, Loïc Huertas, Romain Rouvoy, and Anita Sobe. "Process-level power estimation in VM-based systems." In: *Proceedings of the Tenth European Conference on Computer Systems*. EuroSys '15. Bordeaux, France: Association for Computing Machinery, 2015. ISBN: 9781450332385. DOI: `10.1145/2741948.2741971`.

[52] Ian Cutress. *Thread Director: Windows 11 Does It Best*. Available at: `https://www.anandtech.com/show/16959/intel-innovation-alder-lake-november-4th/3` [Online; accessed 08-April-2025]. Oct. 2021.

[53] Anup Das, Bashir M. Al-Hashimi, and Geoff V. Merrett. "Adaptive and Hierarchical Runtime Manager for Energy-Aware Thermal Management of Embedded Systems." In: *ACM Transactions on Embedded Computing Systems (TECS)* 15.2 (Jan. 2016). ISSN: 1539-9087. DOI: `10.1145/2834120`.

[54] Howard David, Eugene Gorbatov, Ulf R. Hanebutte, Rahul Khanna, and Christian Le. "RAPL: Memory power estimation and capping." In: *2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*. 2010, pp. 189–194. DOI: `10.1145/1840845.1840883`.

[55] Robert H. Dennard, Fritz H. Gaensslen, Hwa-Nien Yu, V. Leo Rideout, Ernest Bassous, and Andre R. LeBlanc. "Design of ion-implanted MOSFET's with very small physical dimensions." In: *IEEE Journal of Solid-State Circuits* 9.5 (1974), pp. 256–268. DOI: `10.1109/JSSC.1974.1050511`.

[56] TensorFlow Developers. "TensorFlow." In: *Zenodo* (2022).

[57] A. E. Eiben and James E. Smith. *Introduction to Evolutionary Computing*. 2nd. Springer Publishing Company, Incorporated, 2015. ISBN: 3662448734.

[58] Cagkan Erbas, Selin Cerav-Erbas, and Andy D. Pimentel. "Multiobjective optimization and evolutionary algorithms for the application mapping problem in multiprocessor system-on-chip design." In: *IEEE Transactions on Evolutionary Computation* 10.3 (2006), pp. 358–374. DOI: `10.1109/TEVC.2005.860766`.

[59] Hadi Esmaeilzadeh, Emily Blem, Renée St. Amant, Karthikeyan Sankaralingam, and Doug Burger. "Dark silicon and the end of multicore scaling." In: *2011 38th Annual International Symposium on Computer Architecture (ISCA)*. 2011, pp. 365–376. DOI: `10.1145/2000064.2000108`.

[60] Khalil Esper, Stefan Wildermann, and Jürgen Teich. "A Comparative Evaluation of Latency-Aware Energy Optimization Approaches in Many-Core Systems." In: *Second Workshop on Next Generation Real-Time Embedded Systems (NG-RES 2021)*. Ed. by Marko Bertogna and Federico Terraneo. Vol. 87. Open Access

Series in Informatics (OASIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, 1:1–1:12. ISBN: 978-3-95977-178-8. DOI: `10.4230/OASIcs.NG-RES.2021.1`.

[61] *Evolved Universal Terrestrial Radio Access (E-UTRA); Physical channels and modulation*. Technical Specification (TS) 36.211. Version 14.2.0. 3rd Generation Partnership Project (3GPP), Apr. 2017. URL: `https://www.etsi.org/deliver/etsi_ts/136200_136299/136211/14.02.00_60/ts_136211v140200p.pdf`.

[62] Sardar M. Farhad, Yousun Ko, Bernd Burgstaller, and Bernhard Scholz. "Orchestration by Approximation: Mapping Stream Programs Onto Multicore Architectures." In: *SIGPLAN Not.* 46.3 (Mar. 2011), pp. 357–368. ISSN: 0362-1340. DOI: `10.1145/1961296.1950406`.

[63] Dror G. Feitelson and Larry Rudolph. "Toward Convergence in Job Schedulers for Parallel Supercomputers." In: *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*. IPPS '96. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 1–26. ISBN: 978-3-540-70710-3. DOI: `10.1007/BFb0022284`.

[64] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner Gardner, Marc Parizeau, and Christian Gagné. "DEAP: evolutionary algorithms made easy." In: *J. Mach. Learn. Res.* 13.1 (July 2012), pp. 2171–2175. ISSN: 1532-4435.

[65] Pascal Fradet, Alain Girault, Ruby Krishnaswamy, Xavier Nicollin, and Arash Shafiei. "RDF: A Reconfigurable Dataflow Model of Computation." In: *ACM Trans. Embed. Comput. Syst.* 22.1 (Oct. 2022). ISSN: 1539-9087. DOI: `10.1145/3544972`.

[66] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. USA: W. H. Freeman & Co., 1990. ISBN: 0716710455.

[67] David Geer. "Chip makers turn to multicore processors." In: *Computer* 38.5 (2005), pp. 11–13. DOI: `10.1109/MC.2005.160`.

[68] Andrés Goens, Robert Khasanov, Jeronimo Castrillon, Marcus Hähnel, Till Smejkal, and Hermann 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*. SCOPES '17. Sankt Goar, Germany: Association for Computing Machinery, 2017, pp. 11–20. ISBN: 9781450350396. DOI: `10.1145/3078659.3078663`.

[69] Andrés Goens, Sergio Siccha, and Jeronimo Castrillon. "Symmetry in Software Synthesis." In: *ACM Trans. Archit. Code Optim.* 14.2 (July 2017). ISSN: 1544-3566. DOI: `10.1145/3095747`.

[70] Michael I. Gordon, William Thies, and Saman Amarasinghe. "Exploiting Coarse-grained Task, Data, and Pipeline Parallelism in Stream Programs." In: *SIGOPS Oper. Syst. Rev.* 40.5 (Oct. 2006), pp. 151–162. ISSN: 0163-5980. DOI: `10.1145/1168918.1168877`.

[71] Michael I. Gordon, William Thies, Michal Karczmarek, Jasper Lin, Ali S. Meli, Andrew A. Lamb, Chris Leger, Jeremy Wong, Henry Hoffmann, David Maze, and Saman Amarasinghe. "A Stream Compiler for Communication-exposed Architectures." In: *SIGOPS Oper. Syst. Rev.* 36.5 (Oct. 2002), pp. 291–303. ISSN: 0163-5964. DOI: `10.1145/635506.605428`.

[72] Peter Greenhalgh. *Big.LITTLE Processing with ARM Cortex-A15 & Cortex-A7*. Available at: `https://www.eetimes.com/big-little-processing-with-arm-cortex-a15-cortex-a7/` [Online; accessed 08-April-2025]. 2011.

[73] Soonhoi Ha and Jürgen Teich. *Handbook of Hardware/Software Codesign*. 1st. Springer Publishing Company, Incorporated, 2017. ISBN: 9401772681.

[74] Daniel Hackenberg, Robert Schöne, Thomas Ilsche, Daniel Molka, Joseph Schuchart, and Robin Geyer. "An Energy Efficiency Feature Survey of the Intel Haswell Processor." In: *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*. 2015, pp. 896–904. DOI: `10.1109/IPDPSW.2015.70`.

[75] Marcus Hähnel, Björn Döbel, Marcus Völp, and Hermann Härtig. "Measuring energy consumption for short code paths using RAPL." In: *SIGMETRICS Perform. Eval. Rev.* 40.3 (Jan. 2012), pp. 13–17. ISSN: 0163-5999. DOI: `10.1145/2425248.2425252`.

[76] Marcus Hähnel and Hermann Härtig. "Heterogeneity by the Numbers: A Study of the ODROID XU+E big.LITTLE Platform." In: *6th Workshop on Power-Aware Computing and Systems (HotPower 14)*. Broomfield, CO: USENIX Association, Oct. 2014. URL: `https://www.usenix.org/conference/hotpower14/workshop-program/presentation/hahnel`.

[77] Marcus Hähnel and Till Smejkal. "Modular Energy Modeling Using Energy/Utility." In: *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*. ICPE '18. Berlin, Germany: Association for Computing Machinery, 2018, pp. 73–78. ISBN: 9781450356299. DOI: `10.1145/3185768.3186311`.

[78] Hardkernel Co., Ltd. *ODROID-XU4 User Manual*. Available at: `https://magazine.odroid.com/wp-content/uploads/odroid-xu4-user-manual.pdf` [Online; accessed 08-April-2025]. 2014.

[79] Christopher Harper. *A third AMD Strix Point Ryzen AI CPU has been officially confirmed— Ryzen AI 9 HX 375 debuts just above HX 370*. `https://www.tomshardware.com/pc-components/cpus/a-third-amd-strix-point-ryzen-ai-cpu-has-been-officially-confirmed-the-ryzen-ai-9-hx-375-debuts-just-above-hx-370`. [Online; accessed 08-April-2025]. 2024.

[80] Hongyu Hè, Michal Friedman, and Theodoros Rekatsinas. "EnergAt: Fine-Grained Energy Attribution for Multi-Tenancy." In: *Proceedings of the 2nd Workshop on Sustainable Computer Systems*. HotCarbon '23. Boston, MA, USA: Association for Computing Machinery, 2023. ISBN: 9798400702426. DOI: `10.1145/3604930.3605716`.

[81] Martin Hirzel, Robert Soulé, Scott Schneider, Buğra Gedik, and Robert Grimm. "A Catalog of Stream Processing Optimizations." In: *ACM Comput. Surv.* 46.4 (Mar. 2014), 46:1–46:34. ISSN: 0360-0300. DOI: `10.1145/2528412`.

[82] Alan Holt and Chi-Yu Huang. *Embedded Operating Systems: A Practical Approach*. 2nd. Springer Publishing Company, Incorporated, 2018. ISBN: 3319729764.

[83] Jia Huang, Andreas Raabe, Christian Buckl, and Alois Knoll. "A workflow for runtime adaptive task allocation on heterogeneous MPSoCs." In: *2011 Design, Automation & Test in Europe*. 2011, pp. 1–6. DOI: `10.1109/DATE.2011.5763189`.

[84] Paolo Ienne and Rainer Leupers. *Customizable Embedded Processors: Design Technologies and Applications*. English. Series in Systems on Silicon. Massachusetts: Morgan Kaufmann, July 2006. ISBN: 0-123-69526-0. DOI: `10.1016/B978-0-12-369526-0.X5000-1`.

[85] Thomas Ilsche, Daniel Hackenberg, Stefan Graul, Robert Schöne, and Joseph Schuchart. "Power measurements for compute nodes: Improving sampling rates, granularity and accuracy." In: *2015 Sixth International Green and Sustainable Computing Conference (IGSC)*. 2015, pp. 1–8. DOI: `10.1109/IGCC.2015.7393710`.

[86] Intel. *Intel Unveils 12th Gen Intel Core, Launches World's Best Gaming Processor, i9-12900K*. Available at: `https://www.intc.com/news-events/press-releases/detail/1506/intel-unveils-12th-gen-intel-core-launches-worlds-best` [Online; accessed 08-April-2025]. Oct. 2021.

[87] Intel Corporation. *What Is Intel® Thread Director?* [Online; accessed 08-April-2025]. URL: `https://www.intel.com/content/www/us/en/support/articles/000097053/processors/intel-core-processors.html`.

[88] Haris Javaid and Sri Parameswaran. "A design flow for application specific heterogeneous pipelined multiprocessor systems." In: *2009 46th ACM/IEEE Design Automation Conference*. 2009, pp. 250–253. DOI: `10.1145/1629911.1629979`.

[89] JmsDoug and Fritzchens Fritz. *Core i9-13900K labelled die shot*. `https://en.wikipedia.org/wiki/Raptor_Lake#/media/File:Intel_Core_i9-13900K_Labelled_Die_Shot.jpg`. [Online; accessed 08-April-2025]. 2023.

[90] Norman P. Jouppi et al. "In-Datacenter Performance Analysis of a Tensor Processing Unit." In: *SIGARCH Comput. Archit. News* 45.2 (June 2017), pp. 1–12. ISSN: 0163-5964. DOI: `10.1145/3140659.3080246`. URL: `https://doi.org/10.1145/3140659.3080246`.

[91] Gilles Kahn. "The semantics of a simple language for parallel programming." In: *Information processing*. Ed. by J. L. Rosenfeld. Stockholm, Sweden: North Holland, Amsterdam, Aug. 1974, pp. 471–475.

[92] Gilles Kahn and David B. MacQueen. "Coroutines and networks of parallel processes." In: *Information processing*. Ed. by B. Gilchrist. North Holland, Amsterdam, 1977, pp. 993–998.

[93] Niu Kai, Sun Jianxing, He Zhiqiang, and Kok Keong Chai. "LTE eNodeB prototype based on GPP platform." In: *2012 IEEE Globecom Workshops*. 2012, pp. 279–284. DOI: `10.1109/GLOCOMW.2012.6477583`.

[94] Shin-Haeng Kang, Hoeseok Yang, Lars Schor, Iuliana Bacivarov, Soonhoi Ha, and Lothar Thiele. "Multi-objective mapping optimization via problem decomposition for many-core systems." In: *2012 IEEE 10th Symposium on Embedded Systems for Real-time Multimedia*. 2012, pp. 28–37. DOI: `10.1109/ESTIMedia.2012.6507026`.

[95] Lina Karam, Ismail Alkamal, Alan Gatherer, Gene A. Frantz, David V. Anderson, and Brian L. Evans. "Trends in multicore DSP platforms." In: *IEEE Signal Processing Magazine* 26.6 (2009), pp. 38–49. DOI: `10.1109/MSP.2009.934113`.

[96] Manupa Karunaratne, Aditi Kulkarni Mohite, Tulika Mitra, and Li-Shiuan Peh. "HyCUBE: A CGRA with reconfigurable single-cycle multi-hop interconnect." In: *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*. 2017, pp. 1–6. DOI: `10.1145/3061639.3062262`.

[97] Robert Khasanov and Jeronimo Castrillon. "Energy-efficient Runtime Resource Management for Adaptable Multi-application Mapping." In: *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Mar. 2020, pp. 909–914. DOI: `10.23919/DATE48585.2020.9116381`.

[98] Robert Khasanov, Marc Dietrich, and Jeronimo Castrillon. “Flexible Spatio-Temporal Energy-Efficient Runtime Management.” In: *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*. Jan. 2024, pp. 777–784. DOI: `10.1109/ASP-DAC58780.2024.10473885`.

[99] Robert Khasanov, Andrés Goens, and Jeronimo Castrillon. “Implicit Data-Parallelism in Kahn Process Networks: Bridging the MacQueen Gap.” In: *Proceedings of the 9th Workshop and 7th Workshop on Parallel Programming and RunTime Management Techniques for Manycore Architectures and Design Tools and Architectures for Multicore Embedded Computing Platforms*. PARMA-DITAM ’18. Manchester, United Kingdom: Association for Computing Machinery, Jan. 2018, pp. 20–25. ISBN: 978-1-4503-6444-7. DOI: `10.1145/3183767.3183790`.

[100] Robert Khasanov, Julian Robledo, Christian Menard, Andrés Goens, and Jeronimo Castrillon. “Domain-specific Hybrid Mapping for Energy-efficient Baseband Processing in Wireless Networks.” In: *ACM Trans. Embed. Comput. Syst.* 20.5s (Oct. 2021). ISSN: 1539-9087. DOI: `10.1145/3476991`.

[101] Thomas Kissinger, Marcus Hähnel, Till Smejkal, Dirk Habich, Hermann Härtig, and Wolfgang Lehner. “Energy-Utility Function-Based Resource Control for In-Memory Database Systems LIVE.” In: *Proceedings of the 2018 International Conference on Management of Data*. SIGMOD ’18. Houston, TX, USA: Association for Computing Machinery, 2018, pp. 1717–1720. ISBN: 9781450347037. DOI: `10.1145/3183713.3193554`.

[102] Guilherme Korol, Michael Guilherme Jordan, Mateus Beck Rutzig, Jeronimo Castrillon, and Antonio Carlos Schneider Beck. “Pruning and Early-Exit Co-Optimization for CNN Acceleration on FPGAs.” In: *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2023, pp. 1–6. DOI: `10.23919/DATE56975.2023.10137244`.

[103] David Koufaty, Dheeraj Reddy, and Scott Hahn. “Bias scheduling in heterogeneous multi-core architectures.” In: *Proceedings of the 5th European Conference on Computer Systems*. EuroSys ’10. Paris, France: Association for Computing Machinery, 2010, pp. 125–138. ISBN: 9781605585772. DOI: `10.1145/1755913.1755928`.

[104] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet classification with deep convolutional neural networks.” In: *Commun. ACM* 60.6 (May 2017), pp. 84–90. ISSN: 0001-0782. DOI: `10.1145/3065386`.

[105] Manjunath Kudlur and Scott Mahlke. "Orchestrating the Execution of Stream Programs on Multicore Platforms." In: *SIGPLAN Not.* 43.6 (June 2008), pp. 114–124. ISSN: 0362-1340. DOI: `10.1145/1379022.1375596`.

[106] R. Kumar, D.M. Tullsen, P. Ranganathan, N.P. Jouppi, and K.I. Farkas. "Single-ISA heterogeneous multi-core architectures for multithreaded workload performance." In: *Proceedings. 31st Annual International Symposium on Computer Architecture, 2004.* 2004, pp. 64–75. DOI: `10.1109/ISCA.2004.1310764`.

[107] Edward A Lee and Thomas M Parks. "Dataflow process networks." In: *Proceedings of the IEEE* 83.5 (1995), pp. 773–801. DOI: `10.1109/5.381846`.

[108] Edward A. Lee. "Consistency in dataflow graphs." In: *IEEE Transactions on Parallel and Distributed Systems* 2.2 (1991), pp. 223–235. DOI: `10.1109/71.89067`.

[109] Edward A. Lee and David G. Messerschmitt. "Synchronous data flow." In: *Proceedings of the IEEE* 75.9 (Sept. 1987), pp. 1235–1245. ISSN: 0018-9219. DOI: `10.1109/PROC.1987.13876`.

[110] Edward Ashford Lee and Sanjit Arunkumar Seshia. *Introduction to Embedded Systems: A Cyber-Physical Systems Approach*. 2nd. The MIT Press, 2016. ISBN: 0262533812.

[111] Haeseung Lee, Weijia Che, and Karam Chatha. "Dynamic Scheduling of Stream Programs on Embedded Multi-core Processors." In: *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. CODES+ISSS '12. Tampere, Finland: Association for Computing Machinery, 2012, pp. 93–102. ISBN: 9781450314268. DOI: `10.1145/2380445.2380465`.

[112] Aini Li, Yan Sun, Xiaodong Xu, and Chunjing Yuan. "An energy-effective network deployment scheme for 5G Cloud Radio Access Networks." In: *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. 2016, pp. 684–689. DOI: `10.1109/INFCOMW.2016.7562164`.

[113] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. "Pruning Filters for Efficient ConvNets." In: *International Conference on Learning Representations*. 2017. URL: `https://openreview.net/forum?id=rJqFGTslg`.

[114] Kaipeng Li, Rishi R. Sharan, Yujun Chen, Tom Goldstein, Joseph R. Cavallaro, and Christoph Studer. "Decentralized Baseband Processing for Massive MU-MIMO Systems." In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 7.4 (2017), pp. 491–507. DOI: `10.1109/JETCAS.2017.2775151`.

[115] Simone Libutti, Giuseppe Massari, and William Fornaciari. "Co-scheduling tasks on multi-core heterogeneous systems: An energy-aware perspective." In: *IET Computers & Digital Techniques* 10.2 (2016), pp. 77–84. DOI: `10.1049/iet-cdt.2015.0053`.

[116] Liang-Yu Lin, Cheng-Yeh Wang, Pao-Jui Huang, Chih-Chieh Chou, and Jing-Yang Jou. "Communication-driven task binding for multiprocessor with latency insensitive network-on-chip." In: *Proceedings of the ASP-DAC 2005. Asia and South Pacific Design Automation Conference, 2005.* Vol. 1. 2005, 39–44 Vol. 1. DOI: `10.1109/ASPDAC.2005.1466126`.

[117] Linux Kernel Community. *Energy Aware Scheduling*. Available at: `https://docs.kernel.org/scheduler/sched-energy.html` [Online; accessed 08-April-2025].

[118] Zhiye Liu. *AMD Phoenix 2 Review Evaluates Zen 4, Zen 4c Performance*. Available at: `https://www.tomshardware.com/news/amd-phoenix-2-review-evaluates-zen-4-zen-4c-performance` [Online; accessed 08-April-2025]. Sept. 2023.

[119] Wictor Lund, Sudeep Kanur, Johan Ersfolk, Leonidas Tsiopoulos, Johan Lilius, Joakim Haldin, and Ulf Falk. "Execution of Dataflow Process Networks on OpenCL Platforms." In: *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. Mar. 2015, pp. 618–625. DOI: `10.1109/PDP.2015.29`.

[120] Avinash Malik, Cameron Walker, Michael O'Sullivan, and Oliver Sinnen. "Satisfiability modulo theory (SMT) formulation for optimal scheduling of task graphs with communication delay." In: *Computers & Operations Research* 89 (2018), pp. 113–126. ISSN: 0305-0548. DOI: `10.1016/j.cor.2017.08.012`.

[121] Sorin Manolache, Petru Eles, and Zebo Peng. "Task mapping and priority assignment for soft real-time applications under deadline miss ratio constraints." In: *ACM Trans. Embed. Comput. Syst.* 7.2 (Jan. 2008). ISSN: 1539-9087. DOI: `10.1145/1331331.1331343`.

[122] Giovanni Mariani, Vlad-Mihai Sima, Gianluca Palermo, Vittorio Zaccaria, Cristina Silvano, and Koen Bertels. "Using multi-objective design space exploration to enable run-time resource management for reconfigurable architectures." In: *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2012, pp. 1379–1384. DOI: `10.1109/DATE.2012.6176578`.

[123] Silvano Martello and Paolo Toth. *Knapsack Problems: Algorithms and Computer Implementations*. New York, NY, USA: John Wiley & Sons, Inc., 1990. ISBN: 0-471-92420-2.

[124] Alain J Martin. "Towards an energy complexity of computation." In: *Information Processing Letters* 77.2 (2001), pp. 181–187. ISSN: 0020-0190. DOI: `10.1016/S0020-0190(00)00214-3`.

[125] P. Marwedel. *Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems, and the Internet of Things*. Embedded Systems. Springer International Publishing, 2021. ISBN: 9783030609108. URL: `https://books.google.de/books?id=qCoXEAAAQBAJ`.

[126] Giuseppe Massari, Edoardo Paone, Patrick Bellasi, Gianluca Palermo, Vittorio Zaccaria, William Fornaciari, and Cristina Silvano. "Combining application adaptivity and system-wide Resource Management on multi-core platforms." In: *2014 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV)*. 2014, pp. 26–33. DOI: `10.1109/SAMOS.2014.6893191`.

[127] David McGrew, Michael Curcio, and Scott Fluhrer. *Leighton-Micali Hash-Based Signatures*. RFC 8554. Apr. 2019. DOI: `10.17487/RFC8554`.

[128] Armin Mehran, Ahmad Khadem-Zadeh, and Samira Saeidi. "DSM: A Heuristic Dynamic Spiral Mapping algorithm for network on chip." In: *IEICE Electronics Express* 5.13 (July 2008), pp. 464–471. DOI: `10.1587/elex.5.464`.

[129] Christian Menard, Andrés Goens, and Jeronimo Castrillon. "High-level NoC model for MPSoC compilers." In: *2016 IEEE Nordic Circuits and Systems Conference (NORCAS)*. 2016, pp. 1–6. DOI: `10.1109/NORCHIP.2016.7792876`.

[130] Christian Menard, Andrés Goens, Gerald Hempel, Robert Khasanov, Julian Robledo, Felix Teweleitt, and Jeronimo Castrillon. "Mocasin——Rapid Prototyping of Rapid Prototyping Tools: A Framework for Exploring New Approaches in Mapping Software to Heterogeneous Multi-cores." In: *Proceedings of the 2021 Drone Systems Engineering and Rapid Simulation and Performance Evaluation: Methods and Tools Proceedings*. DroneSE and RAPIDO '21. Budapest, Hungary: Association for Computing Machinery, Jan. 2021, pp. 66–73. ISBN: 9781450389525. DOI: `10.1145/3444950.3447285`.

[131] Tulika Mitra. "Heterogeneous Multi-core Architectures." In: *IPSJ Transactions on System and LSI Design Methodology* 8 (2015), pp. 51–62. DOI: `10.2197/ipsjtsldm.8.51`.

[132] Sparsh Mittal. "A Survey of Techniques for Approximate Computing." In: *ACM Comput. Surv.* 48.4 (Mar. 2016). ISSN: 0360-0300. DOI: `10.1145/2893356`.

[133] Jeffrey C. Mogul, Jayaram Mudigonda, Nathan Binkert, Parthasarathy Ranganathan, and Vanish Talwar. "Using Asymmetric Single-ISA CMPs to Save Energy on Operating Systems." In: *IEEE Micro* 28.3 (2008), pp. 26–41. DOI: `10.1109/MM.2008.47`.

[134] Gordon E. Moore. "Cramming more components onto integrated circuits." In: *Electronics* 38.8 (Apr. 1965).

[135] Orlando Moreira, Jacob Jan-David Mol, and Marco Bekooij. "Online resource management in a multiprocessor with a network-on-chip." In: *Proceedings of the 2007 ACM Symposium on Applied Computing*. SAC '07. Seoul, Korea: Association for Computing Machinery, 2007, pp. 1557–1564. ISBN: 1595934804. DOI: `10.1145/1244002.1244335`.

[136] Tridib Mukherjee, Ayan Banerjee, Georgios Varsamopoulos, Sandeep K. S. Gupta, and Sanjay Rungta. "Spatio-temporal thermal-aware job scheduling to minimize energy consumption in virtualized heterogeneous data centers." In: *Computer Networks* 53.17 (Dec. 2009). Virtualized Data Centers, pp. 2888–2904. ISSN: 1389-1286. DOI: `10.1016/j.comnet.2009.06.008`.

[137] Ricardo Neri. *Introduce classes of tasks for load balance*. Available at: `https://lore.kernel.org/lkml/20230613042422.5344-1-ricardo.neri-calderon@linux.intel.com/` [Online; accessed 08-April-2025]. June 2023.

[138] Mina Niknafs, Ivan Ukhov, Petru Eles, and Zebo Peng. "Runtime Resource Management with Workload Prediction." In: *Proceedings of the 56th Annual Design Automation Conference 2019*. DAC '19. Las Vegas, NV, USA: Association for Computing Machinery, 2019. ISBN: 9781450367257. DOI: `10.1145/3316781.3317902`.

[139] Vincent Nollet, Prabhat Avasare, Hendrik Eeckhaut, Diederik Verkest, and Henk Corporaal. "Run-Time Management of a MPSoC Containing FPGA Fabric Tiles." In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 16.1 (2008), pp. 24–33. DOI: `10.1109/TVLSI.2007.912097`.

[140] Gereon Onnebrink, Ahmed Hallawa, Rainer Leupers, Gerd Ascheid, and Awaid-Ud-Din Shaheen. "A heuristic for multi objective software application mappings on heterogeneous MPSoCs." In: *Proceedings of the 24th Asia and South Pacific Design Automation Conference*. ASPDAC '19. Tokyo, Japan: Association for Computing Machinery, 2019, pp. 609–614. ISBN: 9781450360074. DOI: `10.1145/3287624.3287651`.

[141] Heikki Orsila, Tero Kangas, Erno Salminen, Timo D. Hämäläinen, and Marko Hännikäinen. "Automated memory-aware application distribution for Multi-processor System-on-Chips."

In: *J. Syst. Archit.* 53.11 (Nov. 2007), pp. 795–815. ISSN: 1383-7621. DOI: `10.1016/j.sysarc.2007.01.013`.

[142] Chandandeep Singh Pabla. "Completely fair scheduler." In: *Linux J.* 2009.184 (Aug. 2009). ISSN: 1075-3583.

[143] Thomas Martyn Parks. "Bounded scheduling of process networks." UMI Order No. GAX96-21312. PhD thesis. USA, 1996.

[144] Paul I Pénzes and Alain J. Martin. "Energy-delay efficiency of VLSI computations." In: *Proceedings of the 12th ACM Great Lakes Symposium on VLSI*. GLSVLSI '02. New York, New York, USA: Association for Computing Machinery, 2002, pp. 104–111. ISBN: 1581134622. DOI: `10.1145/505306.505330`.

[145] Quentin Perret. *Energy Aware Scheduling (EAS) in Linux 5.0*. Available at: `https://community.arm.com/arm-community-blogs/b/architectures-and-processors-blog/posts/energy-aware-scheduling-in-linux` [Online; accessed 08-April-2025]. Feb. 2019.

[146] Chuck Pheatt. "Intel® threading building blocks." In: *Journal of Computing Sciences in Colleges* 23.4 (2008), pp. 298–298. URL: `https://api.semanticscholar.org/CorpusID:59747733`.

[147] Andy D. Pimentel, Cagkan Erbas, and Simon Polstra. "A systematic approach to exploring embedded system architectures at multiple abstraction levels." In: *IEEE Transactions on Computers* 55.2 (2006), pp. 99–112. DOI: `10.1109/TC.2006.16`.

[148] Behnaz Pourmohseni, Michael Glaß, Jörg Henkel, Heba Khdr, Martin Rapp, Valentina Richthammer, Tobias Schwarzer, Fedor Smirnov, Jan Spieck, Jürgen Teich, Andreas Weichslgartner, and Stefan Wildermann. "Hybrid Application Mapping for Composable Many-Core Systems: Overview and Future Perspective." In: *Journal of Low Power Electronics and Applications* 10.4 (2020). ISSN: 2079-9268. DOI: `10.3390/jlpea10040038`.

[149] Behnaz Pourmohseni, Michael Glaß, and Jürgen Teich. "Automatic operating point distillation for hybrid mapping methodologies." In: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. 2017, pp. 1135–1140. DOI: `10.23919/DATE.2017.7927160`.

[150] Claudius Ptolemaeus, ed. *System Design, Modeling, and Simulation using Ptolemy II*. Ptolemy.org, 2014. URL: `http://ptolemy.org/books/Systems`.

[151] Jakob Puchinger, Günther R. Raidl, and Ulrich Pferschy. "The Multidimensional Knapsack Problem: Structure and Algorithms." In: *INFORMS J. on Computing* 22.2 (Apr. 2010), pp. 250–265. ISSN: 1526-5528. DOI: `10.1287/ijoc.1090.0344`.

[152] Wei Quan and Andy D. Pimentel. “Towards Exploring Vast MPSoC Mapping Design Spaces Using a Bias-Elitist Evolutionary Approach.” In: *2014 17th Euromicro Conference on Digital System Design*. 2014, pp. 655–658. DOI: `10.1109/DSD.2014.46`.

[153] Wei Quan and Andy D. Pimentel. “A Hybrid Task Mapping Algorithm for Heterogeneous MPSoCs.” In: *ACM Trans. Embed. Comput. Syst.* 14.1 (Jan. 2015). ISSN: 1539-9087. DOI: `10.1145/2680542`.

[154] Francesco Ratto, Tiziana Fanni, Luigi Raffo, and Carlo Sau. “Mutual Impact between Clock Gating and High Level Synthesis in Reconfigurable Hardware Accelerators.” In: *Electronics* 10.1 (2021). ISSN: 2079-9292. DOI: `10.3390/electronics10010073`.

[155] João Gabriel Reis and Antônio Augusto Fröhlich. “Mutant Components: Efficiently Managing Multiple Implementations.” In: *2016 VI Brazilian Symposium on Computing Systems Engineering (SBESC)*. IEEE. 2016, pp. 101–108. DOI: `10.1109/SBESC.2016.023`.

[156] João Gabriel Reis and Antônio Augusto Fröhlich. “OS Support for Adaptive Components in Self-aware Systems.” In: *SIGOPS Oper. Syst. Rev.* 51.1 (Sept. 2017), pp. 101–112. ISSN: 0163-5980. DOI: `10.1145/3139645.3139663`.

[157] Chae-Eun Rhee, Han-You Jeong, and Soonhoi Ha. “Many-to-many core-switch mapping in 2-D mesh NoC architectures.” In: *IEEE International Conference on Computer Design: VLSI in Computers and Processors, 2004. ICCD 2004. Proceedings.* 2004, pp. 438–443. DOI: `10.1109/ICCD.2004.1347959`.

[158] Martino Ruggiero, Alessio Guerri, Davide Bertozzi, Francesco Poletti, and Michela Milano. “Communication-aware allocation and scheduling framework for stream-oriented multi-processor systems-on-chip.” In: *Proceedings of the Design Automation & Test in Europe Conference*. DATE ’06. Munich, Germany: European Design and Automation Association, 2006, pp. 3–8. ISBN: 3981080106. DOI: `10.1109/DATE.2006.243950`.

[159] Walid Saad, Mehdi Bennis, and Mingzhe Chen. “A Vision of 6G Wireless Systems: Applications, Trends, Technologies, and Open Research Problems.” In: *IEEE Network* 34.3 (2020), pp. 134–142. DOI: `10.1109/MNET.001.1900287`.

[160] Juan Carlos Saez, Fernando Castro, and Manuel Prieto-Matias. “Enabling performance portability of data-parallel OpenMP applications on asymmetric multicore processors.” In: *Proceedings of the 49th International Conference on Parallel Processing*. ICPP ’20. Edmonton, AB, Canada: Association for Computing Machinery, 2020. ISBN: 9781450388160. DOI: `10.1145/3404397.3404441`.

[161] Juan Carlos Saez, Alexandra Fedorova, Manuel Prieto, and Hugo Vegas. "Operating system support for mitigating software scalability bottlenecks on asymmetric multicore processors." In: *Proceedings of the 7th ACM International Conference on Computing Frontiers*. CF '10. Bertinoro, Italy: Association for Computing Machinery, 2010, pp. 31–40. ISBN: 9781450300445. DOI: `10.1145/1787275.1787281`.

[162] Juan Carlos Saez and Manuel Prieto-Matias. "Evaluation of the Intel thread director technology on an Alder Lake processor." In: *Proceedings of the 13th ACM SIGOPS Asia-Pacific Workshop on Systems*. APSys '22. Virtual Event, Singapore: Association for Computing Machinery, 2022, pp. 61–67. ISBN: 9781450394413. DOI: `10.1145/3546591.3547532`.

[163] Juan Carlos Saez, Daniel Shelepov, Alexandra Fedorova, and Manuel Prieto. "Leveraging workload diversity through OS scheduling to maximize performance on single-ISA heterogeneous multicore systems." In: *Journal of Parallel and Distributed Computing* 71.1 (2011), pp. 114–131. ISSN: 0743-7315. DOI: `10.1016/j.jpdc.2010.08.020`.

[164] Robert Schöne, Thomas Ilsche, Mario Bielert, Andreas Gocht, and Daniel Hackenberg. "Energy Efficiency Features of the Intel Skylake-SP Processor and Their Impact on Performance." In: *2019 International Conference on High Performance Computing & Simulation (HPCS)*. 2019, pp. 399–406. DOI: `10.1109/HPCS48598.2019.9188239`.

[165] Lars Schor, Iuliana Bacivarov, Devendra Rai, Hoeseok Yang, Shin-Haeng Kang, and Lothar Thiele. "Scenario-Based Design Flow for Mapping Streaming Applications onto on-Chip Many-Core Systems." In: *Proceedings of the 2012 International Conference on Compilers, Architectures and Synthesis for Embedded Systems*. CASES '12. Tampere, Finland: Association for Computing Machinery, 2012, pp. 71–80. ISBN: 9781450314244. DOI: `10.1145/2380403.2380422`.

[166] Lars Schor, Iuliana Bacivarov, Hoeseok Yang, and Lothar Thiele. "AdaPNet: Adapting process networks in response to resource variations." In: *2014 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*. Oct. 2014, pp. 1–10. DOI: `10.1145/2656106.2656112`.

[167] Andreas Schranzhofer, Jian-Jia Chen, and Lothar Thiele. "Power-Aware Mapping of Probabilistic Applications onto Heterogeneous MPSoC Platforms." In: *2009 15th IEEE Real-Time and Embedded Technology and Applications Symposium*. 2009, pp. 151–160. DOI: `10.1109/RTAS.2009.24`.

[168] Andreas Schranzhofer, Jian-Jian Chen, and Lothar Thiele. "Dynamic Power-Aware Mapping of Applications onto Heterogeneous MPSoC Platforms." In: *IEEE Transactions on Industrial Informatics* 6.4 (2010), pp. 692–707. DOI: `10.1109/TII.2010.2062192`.

[169] Shahriar Shahabuddin, Aarne Mämmelä, Markku Juntti, and Olli Silvén. "ASIP for 5G and Beyond: Opportunities and Vision." In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 68.3 (2021), pp. 851–857. DOI: `10.1109/TCSII.2021.3050785`.

[170] Chung-Ching Shen, William Plishker, Hsiang-Huang Wu, and Shuvra S Bhattacharyya. "A lightweight dataflow approach for design and implementation of SDR systems." In: *Proceedings of the Wireless Innovation Conference and Product Exposition*. Dec. 2010, pp. 640–645.

[171] Hamid Shojaei, Twan Basten, Marc Geilen, and Azadeh Davoodi. "A fast and scalable multidimensional multiple-choice knapsack heuristic." In: *ACM Trans. Des. Autom. Electron. Syst.* 18.4 (Oct. 2013). ISSN: 1084-4309. DOI: `10.1145/2541012.2541014`.

[172] Abraham Silberschatz, Greg Gagne, and Peter Baer Galvin. *Operating System Concepts*. 6th ed. Wiley, 2002. ISBN: 0471250600.

[173] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: `1409.1556 [cs.CV]`.

[174] Amit Kumar Singh, Akash Kumar, and Thambipillai Srikanthan. "Accelerating throughput-aware runtime mapping for heterogeneous MPSoCs." In: *ACM Trans. Des. Autom. Electron. Syst.* 18.1 (Jan. 2013). ISSN: 1084-4309. DOI: `10.1145/2390191.2390200`.

[175] Amit Kumar Singh, Muhammad Shafique, Akash Kumar, and Jörg Henkel. "Mapping on multi/many-core systems: survey of current and emerging trends." In: *Proceedings of the 50th Annual Design Automation Conference*. DAC '13. Austin, Texas: Association for Computing Machinery, 2013. ISBN: 9781450320719. DOI: `10.1145/2463209.2488734`.

[176] Amit Kumar Singh, Muhammad Shafique, Akash Kumar, and Jörg Henkel. "Resource and Throughput Aware Execution Trace Analysis for Efficient Run-Time Mapping on MPSoCs." In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35.1 (2016), pp. 72–85. DOI: `10.1109/TCAD.2015.2446938`.

[177] Amit Kumar Singh, Thambipillai Srikanthan, Akash Kumar, and Wu Jigang. "Communication-aware heuristics for run-time task mapping on NoC-based MPSoC platforms." In: *Journal of Systems Architecture* 56.7 (2010). Special Issue on HW/SW Co-Design: Systems and Networks on Chip, pp. 242–255. ISSN: 1383-7621. DOI: `10.1016/j.sysarc.2010.04.007`.

[178] Magnus Själander, Sally McKee, Peter Brauer, David Engdal, and Andras Vajda. "An LTE Uplink Receiver PHY Benchmark and Subframe-based Power Management." In: *2012 IEEE International Symposium on Performance Analysis of Systems Software*. 2012, pp. 25–34. DOI: `10.1109/ISPASS.2012.6189203`.

[179] Till Smejkal, Marcus Hähnel, Thomas Ilsche, Michael Roitzsch, Wolfgang E. Nagel, and Hermann Härtig. "E-Team: Practical Energy Accounting for Multi-Core Systems." In: *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. Santa Clara, CA: USENIX Association, July 2017, pp. 589–601. ISBN: 978-1-931971-38-6. URL: `https://www.usenix.org/conference/atc17/technical-sessions/presentation/smejkal`.

[180] Till Smejkal, Robert Khasanov, Jeronimo Castrillon, and Hermann Härtig. *E-Mapper: Energy-Efficient Resource Allocation for Traditional Operating Systems on Heterogeneous Processors*. June 2024. arXiv: `2406.18980 [cs.OS]`.

[181] Lodewijk T. Smit, Gerard J.M. Smit, Johann L. Hurink, Hajo Broersma, Daniel Paulusma, and Pascal T. Wolkotte. "Run-time mapping of applications to a heterogeneous reconfigurable tiled system on chip architecture." In: *Proceedings. 2004 IEEE International Conference on Field- Programmable Technology (IEEE Cat. No.04EX921)*. 2004, pp. 421–424. DOI: `10.1109/FPT.2004.1393315`.

[182] Jelena Spasic, Di Liu, and Todor Stefanov. "Exploiting resource-constrained parallelism in hard real-time streaming applications." In: *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Mar. 2016, pp. 954–959. DOI: `10.3850/9783981537079_0201`.

[183] Jan Spieck, Stefan Wildermann, and Jürgen Teich. "A Learning-Based Methodology for Scenario-Aware Mapping of Soft Real-Time Applications onto Heterogeneous MPSoCs." In: *ACM Trans. Des. Autom. Electron. Syst.* 28.1 (Dec. 2022). ISSN: 1084-4309. DOI: `10.1145/3529230`.

[184] Jan Spieck, Stefan Wildermann, and Jürgen Teich. "A Scenario-Based DVFS-Aware Hybrid Application Mapping Methodology for MPSoCs." In: *ACM Trans. Des. Autom. Electron. Syst.* 29.4 (June 2024). ISSN: 1084-4309. DOI: `10.1145/3660633`.

[185] Manikantan Srinivasan, C Siva Ram Murthy, and Anusuya Balasubramanian. “Modular performance analysis of Multicore SoC-based small cell LTE base station.” In: *2015 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*. 2015, pp. 37–42. DOI: `10.1109/VLSI-SoC.2015.7314388`.

[186] Sundararajan Sriram and Shuvra S. Bhattacharyya. *Embedded Multiprocessors: Scheduling and Synchronization*. 1st. USA: Marcel Dekker, Inc., 2000. ISBN: 0824793188.

[187] Sander Stuijk, Marc Geilen, and Twan Basten. “A Predictable Multiprocessor Design Flow for Streaming Applications with Dynamic Behaviour.” In: *2010 13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools*. 2010, pp. 548–555. DOI: `10.1109/DSD.2010.31`.

[188] Anastasia Stulova, Rainer Leupers, and Gerd Ascheid. “Throughput driven transformations of Synchronous Data Flows for mapping to heterogeneous MPSoCs.” In: *2012 International Conference on Embedded Computer Systems (SAMOS)*. July 2012, pp. 144–151. DOI: `10.1109/SAMOS.2012.6404168`.

[189] Xiaofeng Tao, Yanzhao Hou, Haiyang He, Kaidong Wang, and Yingyue Xu. “GPP-based soft base station designing and optimization (invited paper).” In: *7th International Conference on Communications and Networking in China*. 2012, pp. 49–53. DOI: `10.1109/ChinaCom.2012.6417446`.

[190] Faisal Tariq, Muhammad R. A. Khandaker, Kai-Kit Wong, Muhammad A. Imran, Mehdi Bennis, and Merouane Debbah. “A Speculative Study on 6G.” In: *IEEE Wireless Communications* 27.4 (2020), pp. 118–125. DOI: `10.1109/MWC.001.1900488`.

[191] Surat Teerapittayanon, Bradley McDanel, and H.T. Kung. “BranchyNet: Fast inference via early exiting from deep neural networks.” In: *2016 23rd International Conference on Pattern Recognition (ICPR)*. 2016, pp. 2464–2469. DOI: `10.1109/ICPR.2016.7900006`.

[192] *TensorFlow Lite*. `https://www.tensorflow.org/lite`. [Online; accessed 08-April-2025; TensorFlow Lite has been renamed to LiteRT]. 2023.

[193] Lothar Thiele, Iuliana Bacivarov, Wolfgang Haid, and Kai Huang. “Mapping Applications to Tiled Multiprocessor Embedded Systems.” In: *Seventh International Conference on Application of Concurrency to System Design (ACSD 2007)*. 2007, pp. 29–40. DOI: `10.1109/ACSD.2007.53`.

[194] Vivek Tiwari, Sharad Malik, and Andrew Wolfe. “Power analysis of embedded software: a first step towards software power minimization.” In: *IEEE Transactions on Very Large Scale Integra-*

tion (VLSI) Systems 2.4 (1994), pp. 437–445. DOI: `10.1109/92.335012`.

[195] Alexey Tumanov, Timothy Zhu, Jun Woo Park, Michael A. Kozuch, Mor Harchol-Balter, and Gregory R. Ganger. "TetriSched: Global Rescheduling with Adaptive Plan-Ahead in Dynamic Heterogeneous Clusters." In: *Proceedings of the Eleventh European Conference on Computer Systems*. EuroSys '16. London, United Kingdom: Association for Computing Machinery, 2016. ISBN: 9781450342407. DOI: `10.1145/2901318.2901355`.

[196] Stavros Tzilis, Pedro Trancoso, and Ioannis Sourdis. "Energy-Efficient Runtime Management of Heterogeneous Multicores using Online Projection." In: *ACM Transactions on Architecture and Code Optimization (TACO)* 15.4 (Jan. 2019). ISSN: 1544-3566. DOI: `10.1145/3293446`.

[197] Tore Ulversoy. "Software Defined Radio: Challenges and Opportunities." In: *IEEE Communications Surveys & Tutorials* 12.4 (2010), pp. 531–550. DOI: `10.1109/SURV.2010.032910.00019`.

[198] Vanchinathan Venkataramani, Bruno Bodin, Aditi Kulkarni, Tulika Mitra, and Li-Shiuan Peh. "Time-Predictable Software-Defined Architecture with Sdf-Based Compiler Flow for 5g Baseband Processing." In: *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2020, pp. 1553–1557. DOI: `10.1109/ICASSP40776.2020.9054285`.

[199] Vanchinathan Venkataramani, Aditi Kulkarni, Tulika Mitra, and Li-Shiuan Peh. "SPECTRUM: A Software-defined Predictable Many-core Architecture for LTE/5G Baseband Processing." In: *ACM Trans. Embed. Comput. Syst.* 19.5 (Sept. 2020). ISSN: 1539-9087. DOI: `10.1145/3400032`.

[200] Vanchinathan Venkataramani, Anuj Pathania, and Tulika Mitra. "Scalable Optimal Greedy Scheduler for Asymmetric Multi-/Many-Core Processors." In: *Embedded Computer Systems: Architectures, Modeling, and Simulation*. Cham: Springer International Publishing, Aug. 2019, pp. 127–141. ISBN: 978-3-030-27562-4. DOI: `10.1007/978-3-030-27562-4_9`.

[201] Gregory K. Wallace. "The JPEG still picture compression standard." In: *IEEE Transactions on Consumer Electronics* 38.1 (1992), pp. xviii–xxxiv. DOI: `10.1109/30.125072`.

[202] Xinbo Wang, Cicek Cavdar, Lin Wang, Massimo Tornatore, Hwan Seok Chung, Han Hyub Lee, Soo Myung Park, and Biswanath Mukherjee. "Virtualized Cloud Radio Access Network for 5G Transport." In: *IEEE Communications Magazine* 55.9 (2017), pp. 202–209. DOI: `10.1109/MCOM.2017.1600866`.

[203] Govind Wathan. *Arm DynamIQ: Technology for the next era of compute*. Available at: `https://community.arm.com/arm-community-blogs/b/architectures-and-processors-blog/posts/arm-dynamiq-technology-for-the-next-era-of-compute` [Online; accessed 08-April-2025]. 2017.

[204] Andreas Weichslgartner, Deepak Gangadharan, Stefan Wildermann, Michael Glaß, and Jürgen Teich. "DAARM: Design-time application analysis and run-time mapping for predictable execution in many-core systems." In: *2014 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. 2014, pp. 1–10. DOI: `10.1145/2656075.2656083`.

[205] Andreas Weichslgartner, Stefan Wildermann, Deepak Gangadharan, Michael Glaß, and Jürgen Teich. "A Design-Time/Run-Time Application Mapping Methodology for Predictable Execution Time in MPSoCs." In: *ACM Trans. Embed. Comput. Syst.* 17.5 (Nov. 2018). ISSN: 1539-9087. DOI: `10.1145/3274665`.

[206] Stefan Wildermann, Michael Glaß, and Jürgen Teich. "Multi-objective distributed run-time resource management for many-cores." In: *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Dresden, Germany, 2014, pp. 1–6. ISBN: 978-3-9815370-2-4. DOI: `10.7873/DATE.2014.234`.

[207] Stefan Wildermann, Andreas Weichslgartner, and Jürgen Teich. "Design Methodology and Run-Time Management for Predictable Many-Core Systems." In: *2015 IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*. Apr. 2015, pp. 103–110. DOI: `10.1109/ISORCW.2015.48`.

[208] Robert Wittig, Andrés Goens, Christian Menard, Emil Matus, Gerhard P. Fettweis, and Jeronimo Castrillon. "Modem Design in the Era of 5G and Beyond: The Need for a Formal Approach." In: *2020 27th International Conference on Telecommunications (ICT)*. Oct. 2020, pp. 1–5. DOI: `10.1109/ICT49546.2020.9239539`.

[209] Dong Wu, B.M. Al-Hashimi, and P. Eles. "Scheduling and mapping of conditional task graphs for the synthesis of low power embedded systems." In: *2003 Design, Automation and Test in Europe Conference and Exhibition*. 2003, pp. 90–95. DOI: `10.1109/DATE.2003.1253592`.

[210] Peng Yang, Paul Marchal, Chun Wong, Stefaan Himpe, Francky Catthoor, Patrick David, Johan Vounckx, and Rudy Lauwereins. "Managing dynamic concurrent tasks in embedded real-time multimedia systems." In: *Proceedings of the 15th International Symposium on System Synthesis*. ISSS '02. Kyoto, Japan: Association for Computing Machinery, 2002, pp. 112–119. ISBN: 1581135769. DOI: `10.1145/581199.581226`.

[211] Simei Yang, Sébastien lez Nours, Maria mendez Real, and Sébastien Pillement. “Mapping and Frequency Joint Optimization for Energy Efficient Execution of Multiple Applications on Multicore Systems.” In: *2019 Conference on Design and Architectures for Signal and Image Processing (DASIP)*. 2019, pp. 29–34. DOI: `10.1109/DASIP48288.2019.9049177`.

[212] Chantal Ykman-Couvreur, Vincent Nollet, Francky Catthoor, and Henk Corporaal. “Fast multidimension multichoice knapsack heuristic for MP-SoC runtime management.” In: *ACM Transactions on Embedded Computing Systems (TECS)* 10.3 (May 2011). ISSN: 1539-9087. DOI: `10.1145/1952522.1952528`.

[213] Jae-Sung Yoon, Jeong-Hyun Kim, Hyo-Eun Kim, Won-Young Lee, Seok-Hoon Kim, Kyusik Chung, Jun-Seok Park, and Lee-Sup Kim. “A Unified Graphics and Vision Processor With a 0.89 /spl mu/W/fps Pose Estimation Engine for Augmented Reality.” In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 21.2 (2013), pp. 206–216. DOI: `10.1109/TVLSI.2012.2186157`.

[214] Heechul Yun, Renato Mancuso, Zheng-Pei Wu, and Rodolfo Pellizzoni. “PALLOC: DRAM bank-aware memory allocator for performance isolation on multicore platforms.” In: *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 2014, pp. 155–166. DOI: `10.1109/RTAS.2014.6925999`.

[215] Heechul Yun, Gang Yao, Rodolfo Pellizzoni, Marco Caccamo, and Lui Sha. “Memguard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms.” In: *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE. 2013, pp. 55–64. DOI: `10.1109/RTAS.2013.6531079`.

[216] Ali Zaidi, Fredrik Athley, Jonas Medbo, Ulf Gustavsson, Giuseppe Durisi, and Xiaoming Chen. *5G Physical Layer: Principles, Models and Technology Components*. 1st. USA: Academic Press, Inc., 2018. ISBN: 9780128145784.

[217] Gang Zeng, Tetsuo Yokoyama, Hiroyuki Tomiyama, and Hiroaki Takada. “Practical Energy-Aware Scheduling for Real-Time Multiprocessor Systems.” In: *2009 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*. 2009, pp. 383–392. DOI: `10.1109/RTCSA.2009.47`.

[218] ZES ZIMMER Electronic Systems. *4-Channel Power Meter LMG450*. Available at: `https://www.zes.com/en/content/download/286/2473/file/Brochure_LMG450_Rev1.0_web_e.pdf` [Online; accessed 08-April-2025].

[219] Jiali Teddy Zhai, Mohamed A. Bamakhrama, and Todor Stefanov. "Exploiting just-enough parallelism when mapping streaming applications in hard real-time systems." In: *Proceedings of the 50th Annual Design Automation Conference*. DAC '13. Austin, Texas: Association for Computing Machinery, 2013. DOI: `10.1145/2463209.2488944`.

[220] Sergey Zhuravlev, Juan Carlos Saez, Sergey Blagodurov, Alexandra Fedorova, and Manuel Prieto. "Survey of Scheduling Techniques for Addressing Shared Resources in Multicore Processors." In: *ACM Comput. Surv.* 45.1 (Dec. 2012). ISSN: 0360-0300. DOI: `10.1145/2379776.2379780`.

COLOPHON
This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede and Ivo Pletikosić. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*".

COPYRIGHT
© 2025, Robert Khasanov