UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

PEDRO HENRIQUE EXENBERGER BECKER

Selectively supporting ISA-extensions to enhance heterogeneous MPSoC designs under power and area constraints

Thesis presented in partial fulfillment
of the requirements for the degree of
Master of Computer Science

Advisor: Prof. Dr. Antonio Carlos Schneider
Beck

Porto Alegre
November 2019

CIP — CATALOGING-IN-PUBLICATION

Becker, Pedro Henrique Exenberger

Selectively supporting ISA-extensions to enhance heterogeneous MPSoC designs under power and area constraints / Pedro Henrique Exenberger Becker. – Porto Alegre: PPGC da UFRGS, 2019.

90 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2019. Advisor: Antonio Carlos Schneider Beck.

1. MPSoC Design. 2. Partial-ISA. 3. Heterogeneous Systems. I. Beck, Antonio Carlos Schneider. II. Título.

“Sonhos não envelhecem.”

— MILTON NASCIMENTO

AGRADECIMENTOS

Gostaria de agradecer a todos que me contribuiram durante o período de mestrado. Em especial, agradeço aos meus pais Raquel e Denis, por seu apoio incondicional, conselhos, e incentivo. Também agradeço à Marina por garantir minha alegria mesmo nos piores dias, pelo suporte, e pelo companheirismo.

Aos colegas de pesquisa também deixo meu agradecimento. Primeiramente ao professor e orientador Antonio, que foi incansável em me ensinar e guiar desde os primeiros dias de pesquisa, ainda na inciciação científica, há mais de quatro anos. Também agradeço a todos os colegas com os quais dividi o laboratório 213 neste período. Particularmente, agradeço ao colega Jeckson pelas revisões e opiniões sobre o presente trabalho.

Por fim, agradeço também ao instituto de informática da UFRGS, corpo docente, secretaria, e todos os demais funcionários que dão seu melhor todos os dias para tornar o INF um local de excelência. Notavelmente, agradeço aos professores que tive na pós-graduação, meu orientador Antonio e professor Luigi, pelas aulas, discussões e ideias.

ABSTRACT

Heterogeneous MPSoCs are crucial to meeting energy efficiency and performance, given their combination of general-purpose cores and accelerators. However, their design yields several restrictions regarding the total power (e.g., for battery-powered devices) and area (e.g., for tiny wearable devices) of the final project. This makes it very challenging for the designers to improve next-generation MPSoCs while respecting such constraints. Towards overcoming this challenge, this work presents a novel technique for MPSoCs design, increasing their specialization and task-parallelism while respecting the former area and power budget. By removing the microarchitectural support of costly ISA extensions (e.g., FP, SIMD, crypto) from a few cores (transforming them into Partial-ISA Cores), we simplify their datapaths, creating slack in the system's area and power budget. Thereby, we make room to add extra in-order cores and hardware accelerators. Given that, applications execute at lower power consumption during their ISA-extension-free phases, since partial cores have much simpler datapaths compared to their full-ISA counterparts. When extension support is necessary, otherwise, applications are migrated to full-ISA cores, maintaining binary compatibility. On top of it, the additional cores and accelerators increase task-level parallelism and make the MPSoC more suitable for application-specific scenarios. We evaluate the effectiveness of our approach by composing different MPSoCs in distinct execution scenarios, using the FP instructions and RISC-V ISA processors as a case study. To coordinate the execution of workloads in this novel design, we also propose two scheduling policies, performance- and energy-oriented. For the former policy, we achieve 2.81x speedup for a neural network road sign detection, 2.58x speedup for a video-streaming app, 1.76x speedup for an edge-computing load, and 1.2x speedup for a task-parallel scenario, consuming 68%, 91%, 56%, and 33% less energy, respectively. For the energy-oriented policy, partial-ISA reduces geomean energy consumption by 61% over a highly efficient baseline, also with increased performance. Overall, we demonstrate that partial-ISA adoption leads to better designs regarding the EDP metric for almost every scenario we investigate.

Keywords: MPSoC Design. Partial-ISA. Heterogeneous Systems.

Suportando extensões de ISA seletivamente para melhorar MPSoCs hetorogêneos sob limitações de potência e área

RESUMO

MPSoCs heterogêneos são fundamentais para garantir eficiência energética e desempenho, dada a sua combinação de núcleos de propósito geral e aceleradores. Entretanto, estes dispositivos carregam diversas restrições quanto à sua potência total (e.g., para poupar bateria) e área (e.g., para caber em dispositivos pequenos) no projeto final. Por este motivo, melhorar a próxima geração de MPSoCs enquanto respeitam-se as restrições de projeto é um grande desafio. Neste trabalho, propomos uma nova técnica para compor MPSoCs, aumentando sua vazão de tarefas e especialização, observando as limitações de área e potência do projeto. Para tanto, removemos o suporte micro arquitetural de extensões de ISA (e.g., Ponto-Flutuante) de alguns núcleos (transformando-os em núcleos parciais), tornando-os mais simples e criando saldo em área e potência. Portanto, criamos espaço para adicionar núcleos pequenos e também aceleradores. Assim, as aplicações podem executar com potência reduzida quando extensões de ISA não são necessarias, já que os núcleos parciais são menos complexos. Quando é necessário executar extensões de ISA, as tarefas são migradas para núcleos não-parciais, mantendo a compatibilidade binária. Além disso, a existência de mais núcleos e de aceleradores dedicados aumenta a vazão de tarefas e torna o MPSoC mais adequado para cenários específicos. Validamos a proposta compondo vários MPSoCs com núcleos parciais executando diferentes cenários, e considerando instruções de ponto-flutuante da ISA RISC-V como estudo de caso. Para gerenciar o sistema, propomos políticas de escalonamento orientada a performance e orientada a energia, para que coordenem a execução de aplicações no MPSoC. Na a primeira política, aumentamos a performance em 2.81x no uso de redes neurais para a detecção de sinalização de trânsito, 2.58x para decodificação de vídeo, 1.76x para um conjunto de aplicações edge, e 1.2x para um cenário multi-tarefa, reduzindo o consumo de energia em 68%, 91%, 56%, and 33%, respectivamente. Para a política orientada a energia, o suporte parcial de ISA reduz o consumo de energia em 61% sobre um sistema original já altamente eficiente, além de aumentar o seu desempenho. Não obstante, demonstramos que o suporte parcial de ISA resulta em melhores projetos de MPSoC na métrica de EDP.

Palavras-chave: Projeto de MPSoCs. ISA parcial. Sistemas Heterogeneos.

LIST OF ABBREVIATIONS AND ACRONYMS

AI Artificial Intelligence.

ASIC Application Specific Integrated Circuit.

ASIP Application-Specific Instruction Set Processor.

CAD Computer-Aided Design.

CMP Chip Multi-Processor.

CNN Convolutional Neural Network.

DCT Discrete Cosine Transform.

DMA Direct Memory Access.

DSE Design Space Exploration.

DSP Digital Signal Processing.

DVFS Dynamic Voltage and Frequency Scaling.

EDP Energy-Delay Product.

FP Floating-Point.

FPU Floating-Point Unit.

FSM Finite State Machine.

GPP General Purpose Processor.

GPU Graphics Processing Unit.

HLS High-Level Synthesis.

ILP Instruction-Level Parallelism.

IO Input/Output.

IoT Internet of Things.

IP Intellectual Property.

IPC Instructions per Second.

ISA Instruction-Set Architecture.

MAC Multiply-Accumulate.

MPSoC Multiprocessor Systems on Chip.

OoO Out-of-Order.

OS Operating System.

ROB Re-order Buffer.

SIMD Single Instruction, Multiple Data.

VLE Variable Length Encoder.

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

Computer systems' usage has dramatically changed in the past decade. They are now spread around the world, executing apps in the smartphones of billions of people, in their smartwatches monitoring activities, and handling countless requests and incoming data in many data-centers and on-the-edge nodes. Actually, not only computing platforms have ramified into many, but also what is to be computed. Nowadays, systems need to cope with the simultaneous execution of a variety of workloads, such as Artificial Intelligence (AI) solutions, video streaming, image processing, health monitoring, and encrypted messages exchange. These are frequent applications in the trendy interaction between embedded and cloud systems such as mobile, edge, and servers.

To manage these varied requirements, Heterogeneous Multiprocessor Systems on Chip (MPSoCs) are a handful solution. They combine performance-driven and energy-efficient General Purpose Processors (GPPs) cores to cooperatively work with dedicated hardware accelerators, also known as Application Specific Integrated Circuits (ASICs). With the GPP cores, MPSoCs can equate performance, area, power, and energy constraints of general-purpose workloads while accelerators can handle specific time-costly jobs such as cryptography, graphics, and AI (SAMSUNG, 2018; APPLE, 2018). Not surprisingly, it is mandatory that next-generation Heterogeneous MPSoCs be even more powerful and efficient, enhancing the execution of current and future computing necessities.

However, delivering such ameliorated designs is not an easy task. Performance improvements, as we detail further, must come while respecting power and area constraints, so we have, for example, small devices that do not overheat. Power consumption, in turn, needs to be reduced, considering its impact on performance. In fact, these design optimization trade-offs are often inter-dependent and require smart insights from designers, so they deliver the final solution in a spot nearer to the optimal. To aggravate, old-fashion solutions such as increasing the frequency and reducing the transistor feature size are becoming unsustainable, as we detail next, and designers may not count on them in the near future. As a consequence, there is a need for novel approaches in systems design so that we can deliver improved MPSoCs to cope with multitasking and specialized workloads.

In this work, we propose a solution for this struggle, observing the burden in area and power of homogeneous Instruction-Set Architecture (ISA) support, .i.e. when all GPPs support all instructions defined by the ISA. Notably, we investigate how the support

of complex ISA-extensions has a significant impact on area and power. We then demonstrate how we can ignore the ISA support of ISA-extensions in *some* cores to alleviate the pressure on power and area constraints, and increase the design space for improved MPSoCs. In the next sections, we go deeper into this discussion for a more precise understanding of the proposal and importance of the present dissertation.

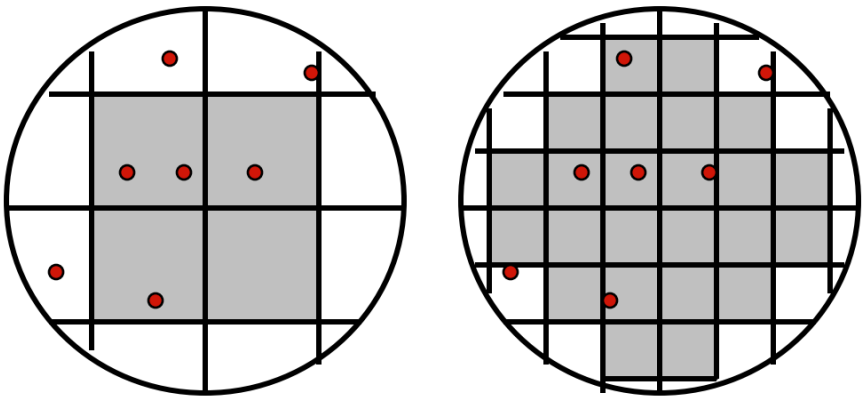
1.1 Constraints of processors' design

The ubiquity of computer systems brings a variety of restrictions for processors design. The power budget of a processor, for example, depends on multiple factors. First, power impacts directly on energy consumption, which is a major concern for battery-powered platforms such as smartphones and wearables, where batteries shall last as long as possible. At the same time, reducing power (and energy) is critical to reducing the costs of keeping data-centers on.

Notwithstanding, power also impacts directly on the chip temperature. Again, this is problematic for mobile because of usability, and also for servers because of the cooling expenditures. Facebook, for example, has placed data-centers near to the arctic circle, in Sweden, because of cheap electricity and low temperatures (KARAGIANNOPOULOS, 2018). It is also noticeable how temperature triggered the dark silicon discussion for many-core architectures (ESMAEILZADEH et al., 2011; SHAFIQUE et al., 2014), where power consumption and associated heat makes the usage of all cores all the time prohibitive.

The area of the chip also restricts processors' design. Similarly to power constraints, reasons are manifold. First, the physical size is a problem, especially for smartphones and wearables, where the device has volume constraints itself. Another reason comes from production costs. The processor's size impacts directly on the number of produced chips per wafer. At the same time, there are intrinsic failures per cm^2 of the silicon wafer. The randomly placed faults introduced by the manufacturing process will affect some of the processors in the die, which will not work properly after deploy. However, if processors in the wafer are small, the number of affected chips will also be small compared to the total number of successfully produced chips (die yield). Figure 1.1 illustrates this discussion. Rabaey et al. (RABAEY; CHANDRAKASAN; NIKOLIC, 2002) concluded that die costs are proportional to the fourth power of the area, based on typical parameters at that time. In a more recent work, authors claim good die yield combined

Figure 1.1: The impact of the die area on the production yield.



Source: (RABAEY; CHANDRAKASAN; NIKOLIC, 2002)

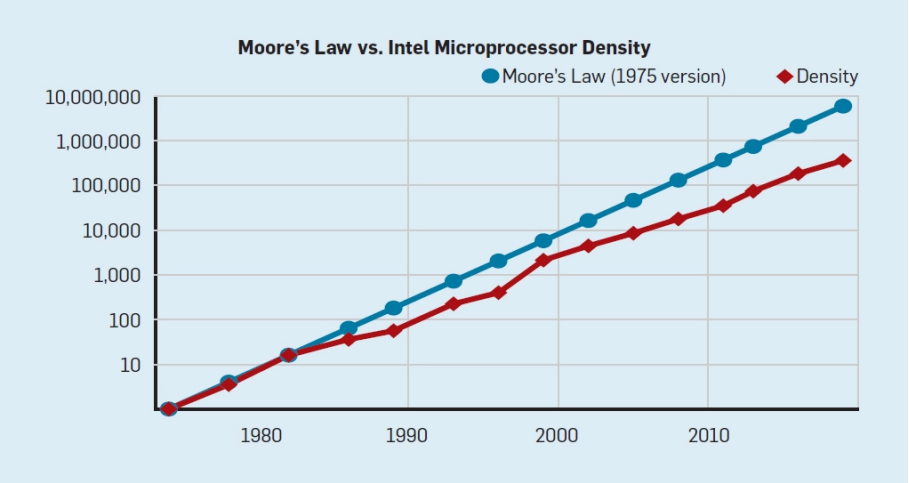
with die per wafer makes die cost proportional to the square of the area in practice (LI et al., 2013). In either case, the power function relation between area and die cost emphasizes the importance of respecting area constraints.

By last, we highlight requirements for performance, which frequently holds the most noteworthy expectations from a computer system. Users want their smartphones to be responsive, for example, watching a real-time video, Internet of Things (IoT) actuators and sensors need short-latency response from on-the-edge nodes, while companies want their websites with the highest service throughput. However, performance is generally conflicting with area and power constraints. Modern heterogeneous MPSoCs may add up multiple cores and specialized hardware to cope with the performance requirements, which mounts area and power demands.

1.2 Challenges for improved MPSoCs

The constraints from processors' design we presented above summarize the expectations for next-generation systems: smaller, more efficient, and more performing. While these requirements are each day more restrictive, the manners to achieve such improvements are also challenging. Following, we discuss two challenges for continually delivering improved complex systems such as MPSoCs: the weakening of technology scaling effectiveness, and the impact of the rising complexity of ISA support. Both challenges support our approach of simplifying the microarchitecture of the processors with partial-ISA, motivating our proposal defined in the next section.

Figure 1.2: Transistors per chip of Intel microprocessors vs. Moore's Law.



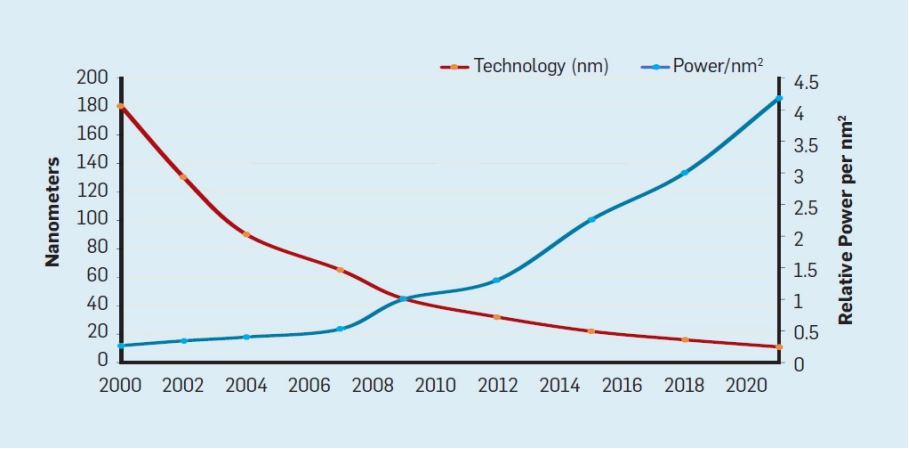
Source: (HENNESSY; PATTERSON, 2019)

1.2.1 Challenges from technology scaling

Continually adding new resources and features to improve heterogeneous MPSoCs, a frequent option for improving systems performance, is becoming unfeasible. Figure 1.2 depicts how the transistor density of Intel microprocessors is disconnecting from Moore's Law predictions. Indeed, some works already claim evidence that Moore's law is coming to an end (WALDROP, 2016; THEIS; Philip Wong, 2017; HENNESSY; PATTERSON, 2019). Advancing to the next technology node is not only difficult, but also very expensive, and need very high-profit expectations to justify the investments (WESTE; HARRIS, 2013). Because transistors size is no longer shrinking at the same pace as before, new techniques are required to deliver more computation density for MPSoCs (i.e., more processors and accelerators) within the same chip die.

The problem is reinforced by the breakdown of Dennard Scaling (HENNESSY; PATTERSON, 2019). By *Dennard scaling*, power per mm^2 would be constant as technology scales. This perception has changed since transistors' technology got below 65nm, as reported in Figure 1.3. For those technology nodes, voltage levels do not scale with feature size (because of threshold voltage), and the static power increases exponentially (because of leakage current), reducing the share of dynamic power in the total power budget, which holds back frequency boost. Therefore, scaling down voltage even further (to reduce power) as also scaling up frequency (to increase performance) is a threatened option. Otherwise, what became necessary is the *wise usage of the available transistors*.

Figure 1.3: Transistors per chip and power per mm^2.



Source: (HENNESSY; PATTERSON, 2019)

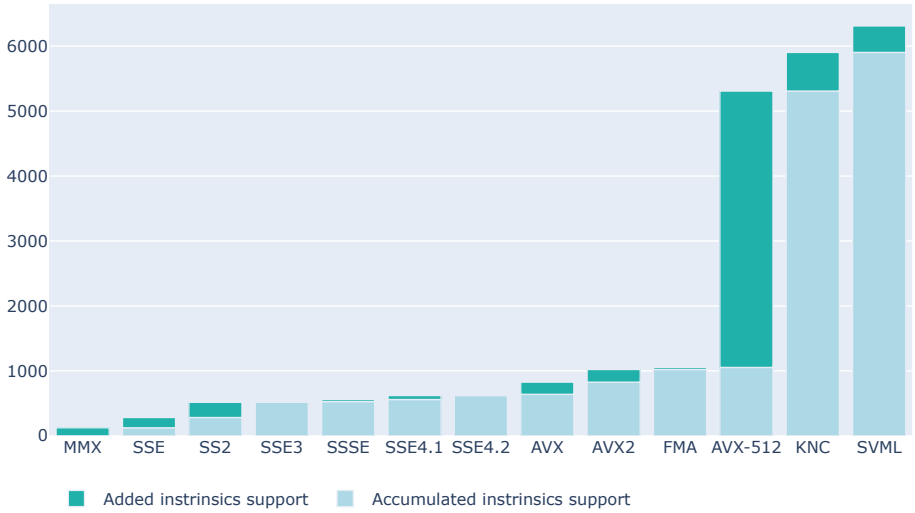
1.2.2 Challenges from ISA support

At the same time, we observe that heterogeneous multi-processors with homogeneous ISAs may suffer from the increased number of ISA extensions (e.g., Intel Intrinsics, ARM NEON) such as Floating-Point (FP), Single Instruction, Multiple Data (SIMD), and cryptography that emerged to deal with trendy applications niches. For example, Figure 1.4 depicts the number of different *Intel Intrinsics* operations, which is a software library that facilitates the use of ISA-extension in Intel processors by software developers. Programmers can leverage this set of Intel ISA-extensions to exploit, e.g., SIMD computation, among many other purposes. As one can see, a processor that supports all of these extensions must be compatible with more than 6,000 operations (in addition to the regular ones).

Although *intrinsics* instructions will be transformed into a combination of a reduced number of micro-ops, the datapath for supporting such a high amount of specialized operations is still considerable. Remarkably, homogeneous ISA multi-core systems have to replicate full ISA support to keep scheduling transparency (so the Operating System (OS) can assign any workload to any available core), raising the systems' area and power requisites.

To illustrate the impact of ISA-extensions, we synthesized a complex Out-of-Order (OoO) processor[1] and present its area and power data in Table 1.1. Particularly, we detail the impact of supporting FP instructions, which we will exploit in this work. As one can see, FP support demands expressive 37% of the processors' area, while consuming more than half of its total power. On the other hand, these instructions may not

Figure 1.4: Available *Intel Intrinsics* operations.



Source: The author, with data from (Intel Corporation, 2018).

be leveraged depending on the workload. Table 1.2 presents three applications to motivate our statement. For instance, FP support is unnecessary for graph traversal and barely leveraged for image processing. Thus, we may be *wasting area budget* implementing expensive Floating-Point Units (FPUs) which could be exchanged for an increased number of computing nodes in the system. Notwithstanding, real-life processors like ARM's A15 (used in big.LITTLE configurations (GREENHALGH, 2011)) apply clock gating to save power consumption of such costly FPUs (ARM Limited, 2012). However, although it avoids dynamic power consumption, it does not prevent static (leakage) power, an increasingly important issue with smaller transistor sizes (HENNESSY; PATTERSON, 2017). In this case, we are also *wasting power budget*.

[1] A 4-issue Berkeley Out-of-Order Machine (ASANOVIC et al., 2015) based on 15nm technology (MARTINS et al., 2015; SHAFAEI et al., 2014), including L1 caches (instruction and data). More details in Chapter 5.

Table 1.1: The impact of FP support on an OoO processor[1]

Processor	**Area (mm^2)**	**Power (W)**
Out-of-order processor	0.34257	0.767
Out-of-order floating-point datapath	0.12453	0.409
% Floating-point support impact	∼37%	∼53%

Table 1.2: Distribution of FP committed instructions (in %) for different application scenarios. Binaries are RISC-V based, the ISA used in our case study.

Scenarios	**FP Usage (%)**
Image processing	2.5%
Graph	0%
Linear Algebra	15%

1.3 Work proposal

Based on the aforementioned observations, we introduce our main idea. We claim that supporting specific ISA-extensions in *some* (instead of *all*) processors in a heterogeneous MPSoC can be a suitable alternative to design performance-, area-, and power-wise systems. For such, we propose to remove the support for a given ISA-extension, allowing us to *trim-out* its related datapath. In this manner, some cores in the system may ignore a subset of the ISA, implementing a *partial-ISA*. Thus, cores' size is diminished, making room for extra hardware modules. We leverage this increased design space to add either simple processors or application-specific hardware, creating suitable systems for task-parallelism or specialized tasks, two important demands of nowadays systems. Therefore, the present work responds to the constrained scenario for increasing MPSoCs capabilities by reallocating former area and power budgets into additional computing nodes.

Naturally, some of the cores in the multi-processor system should still support all of the instructions (*full-ISA*), so we keep the system's compatibility with *full-ISA* dependent binaries. To coordinate the execution of workloads in a partial-ISA system we also propose two mapping strategies: performance-oriented and energy-oriented. These strategies serve as a scheduling policy to guide the task assignment of different workloads in the partial-ISA system.

As case study, we remove the support for FP instructions in some cores in the MPSoC, thereby removing their FPUs. As we demonstrate in our Results chapter, we leverage the created slack in area and power budget to compose different MPSoC configurations, which we use to experiment under different scenarios. We also asses the impact of both mapping strategies mentioned above. In our experimentation, we add extra in-order cores for task parallel scenarios, and ASIC for latency-critical applications, such as video decoding and neural network inference.

With this approach, we demonstrate that MPSoC enhanced through partial-ISA adoption can provide significant performance gains. The use of accelerators reach up to

3.19× speedup over the baseline for a latency-critical scenario, while the in-order cores achieves 1.2× speedup for the task parallel scenarios. Energy gains are even more aggressive. In some scenarios, the partial-ISA MPSoC consumes only $\frac{1}{10}$ of the baseline's energy. In the overall, we show that partial-ISA MPSoCs consistently lead to better Energy-Delay Product (EDP) in the evaluated configurations when compared to traditional MPSoC designs.

The remaining of this dissertation is organized as follows. Fundamental background concepts and related work are presented in Chapter 2. After, in Chapter 3, we present the architecture of MPSoCs aided with partial-ISA. We proceed the simulation environment we developed for this work in Chapter 4. Subsequently, in Chapter 5, we explain the methodology for our experimentation, explore and discuss the achieved results. Finally, Chapter 6 summarizes the conclusions of the work and states future work.

2 BACKGROUND AND RELATED WORK

In this chapter, we discuss background concepts, detailing subjects that will be frequently mentioned along with the present work. We also examine previous work on such topics. We start defining heterogeneous MPSoCs and analyzing previous proposals for delivering better designs of such systems. After, we narrow our discussion to ISA-extensions and the exploitation of partial-ISA (the focus of this work) as an enabler for enhanced MPSoCs configurations. Finally, we compare our proposal with previous works.

2.1 Heterogeneous MPSoC Designs

2.1.1 Background

MPSoCs combine processors and dedicated hardware accelerators into a single chip, being versatile to handle a wide range of applications. Not surprisingly, they were first proposed to achieve effective computation, delivering performance and power efficiency under strict requirements constraints (WOLF; JERRAYA; MARTIN, 2008).

Besides the heterogeneity from GPP and accelerators, MPSoC can also present heterogeneity within its general-purpose cores. Table 2.1 presents four possible MPSoC arrangements regarding the ISA and microarchitecture of its GPPs. When there are cores with different microarchitectures in the MPSoC, it is called a heterogeneous MPSoC. Take for example the heterogeneous MPSoC platform in Figure 2.1. The *Processing System* part of the figure depicts the heterogeneous MPSoC aside with a *Programmable Logic* (not important in our case) shipped with the Xilinx Zynq board. The MPSoC has four ARM A53 and two ARM Cortex R5, and several other hardware modules such as memory controllers, security, and a graphics processing unit. Because A53 and R5 cores have different microarchitectures, the Figures depicts a heterogeneous MPSoC. Even further, the cores have ISA heterogeneity since the A53 cores support the ARM's A64 ISA, which the R5 cores do not support.

Because these modules can be used to host different kinds of applications, MPSoCs can be employed in several circumstances. They are exceptionally well suited for energy-efficient scenarios, because of their specific modules, and when applications are highly heterogeneous. For example, MPSoCs are suitable for smartphones with their nu-

Table 2.1: Possible arrangements of GPPs in a MPSoC regarding the ISA and the microarchitecture.

	ISA	Microarchitecture
Homogeneous	The supported ISA is the same among all cores in the system	The implemented microarchitecture is the same among all cores in the system
Heterogeneous	The supported ISA varies among cores in the system	The implemented microarchitecture varies amog cores in the system

merous peripherals requiring responsiveness, while also having a high load of applications to execute in their GPPs.

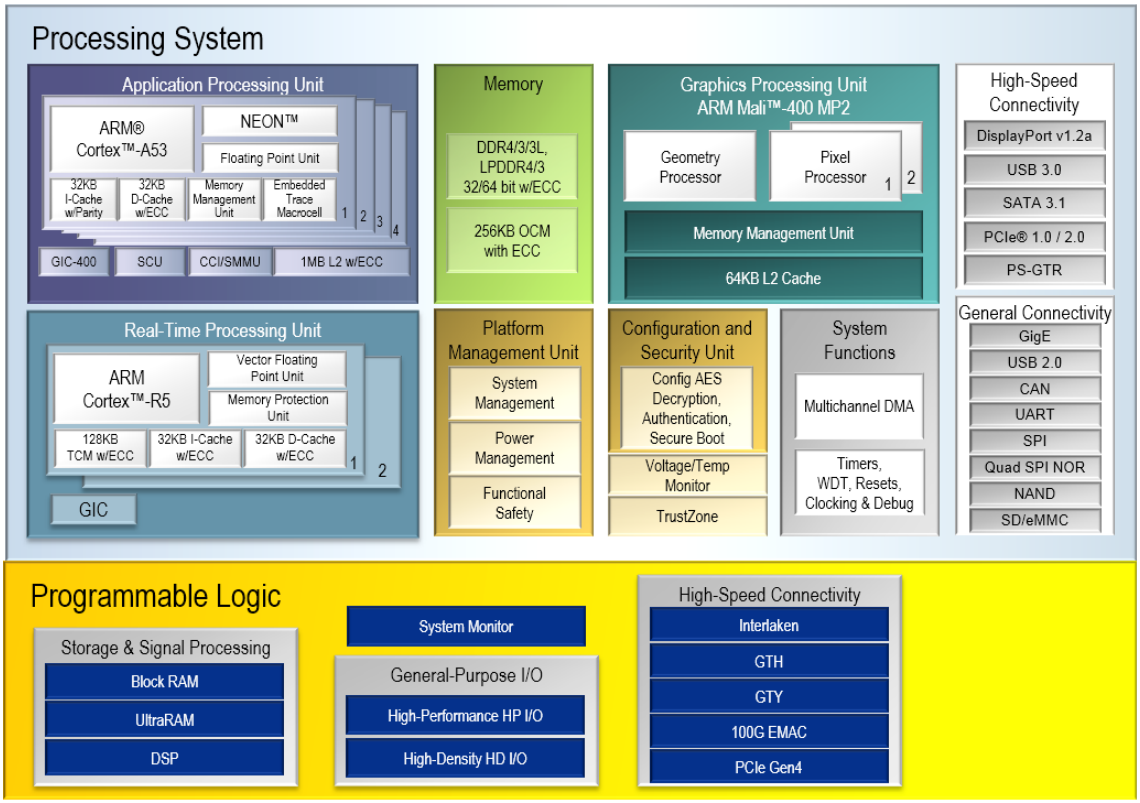
2.1.2 Related Work

Given that MPSoCs are highly adopted in modern systems, a variety of previous research focused on amortizing area and power impacts in MPSoCs design to increase their capabilities under constrained scenarios. Since defining which and how many processors and accelerators to put in the system has a significant impact on the aforementioned aspects of these systems, many works focus on providing tools for fast and valuable Design Space Exploration (DSE) of MPSoC platforms. In these works, the goal is to search for an improved combination of computing nodes in MPSoCs to optimize a set of metrics.

In (ANGIOLINI et al., 2006), authors combine an Application-Specific Instruction Set Processors (ASIPs) Computer-Aided Design (CAD) tool-chain with a virtual platform that enables designers to sweep axes of the configuration space. Through this, it is possible to explore processing elements, memory hierarchy, and chip interconnect fabrics to be adopted in the target heterogeneous MPSoC. The platform is extensible, and allows additional Intellectual Property (IP) cores to be submitted for evaluation. In the results, they present data from a cycle-accurate simulation of cores competing for shared resources across different points in the exploration space (allowed by their tool).

In (Van Stralen; PIMENTEL, 2010), authors introduce the use of different workload scenarios while exploring the MPSoC's design space, for a more adjusted final configuration. For this, authors use a co-evolutionary genetic algorithm to dynamically explore the design space of possible MPSoC arrangements at the same time it searches for representative workload scenarios. They use this approach to select the best heteroge-

Figure 2.1: A Xilinx Zynq MPSoC + reconfigurable logic.



Source: (XILINX, 2019)

neous MPSoC configuration with up to eight elements, among MIPS processors, ARM processors and dedicated Discrete Cosine Transform (DCT) and Variable Length Encoder (VLE) hardware blocks. With this methodology, they demonstrate superior configurations regarding execution time and cost.

Another DSE framework that models both processors and hardware accelerators combined in MPSoCs is detailed in (ROSVALL; SANDER, 2014) and extended in (ROSVALL; SANDER, 2017). It is based on user-defined constraints, the set of applications, and execution schedule for evaluating the design and shows that adopting accelerators in the MPSoC can lead to configurations nearer to the optimal. Singh et al. (SINGH et al., 2013) present an extensive survey of works for mapping methodologies for multi/many-core systems, which also aim to optimize either performance, power consumption, temperature distribution, or other axes of the system. However, although these works on DSE are important to place the system as near to the optimal configuration as possible, they do not improve the possible optimal itself. Notwithstanding, while our solution can be adopted by these DSE tools, it allows them to search in a larger design space, possibly with a superior optimal configuration.

2.2 ISA-Extensions

2.2.1 Background

The ISA of a processor defines the set of instructions that one can leverage to interface with the host processor. The ISA specifies how instructions are assembled, their operation codes, among other definitions for the processor to fetch instructions and operate its bitstream. For example, there are industrial ISAs, such as the x86 (Intel), AMD64 (AMD), and A64 (ARM), as well as research and free-licensed ISAs such as the RISC-V (ASANOVIĆ; PATTERSON, 2014).

However, although these ISAs have a base version, they are continually evolving to increase the semantics for programmers (and compilers) to express computational tasks, as also to expose novel capabilities from the processors' microarchitecture. Take Intel as an example; every x86 processor in production is still capable of executing binary code compiled to execute in an Intel 8086 processor from the '80s (with a few limitations such as prohibiting self-modifying code when the OS set the processor to run in protected mode (Intel Corporation, 2016)). At the same time, Intel have adopted a variety of extensions over the former ISA to cope with novel requirements.

For example, the MMX instructions (PAVER; KHAN; ALDRICH, 2004) were released by Intel as ISA-extensions to improve the execution of multimedia applications. When fetching such instructions, x86-MMX compatible processors execute SIMD operations in chunks of 2, 4 or 8 elements at a time (hence targeting multimedia applications such as gaming, where vector operations are often necessary). Subsequently, many other ISA-extensions were released over the base x86 ISA, such as SSE and AVX. Programmers can leverage these instructions through Intel Intrinsics (Intel Corporation, 2018), which is an interface library in the C language for using a variety of Intel extensions. Similar characteristics occur with other ISAs. ARM, for example, has a base ISA (integer-only), and extensions for FP and SIMD (called NEON), and Digital Signal Processing (DSP) operations, for example.

It is essential to highlight that ISA-extensions are called *extensions* because they are not part of the base ISA. Hence, it is possible to have ISA compatible processors, even though they do not support some of the extensions. The RISC-V ISA, for example, contains a base ISA (integer-only, as ARM's ISA), and several extensions that may or may not be supported by a RISC-V processor. For example, it has extensions for

atomic instructions (for parallel processing), FP instructions, compact instructions (for reduced memory footprint, similar to ARM's thumb), among others. Supporting such ISA-extensions increases the available functionalities for programs execution, but also increases the hardware costs (in area and power) given the need for a datapath that can execute such instructions.

2.2.2 Related Work

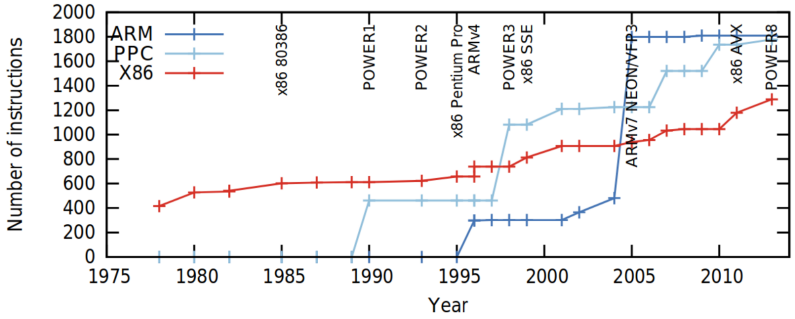
As we discussed in the previous subsection, the ISA plays a vital role in interfacing the applications with the processor's capabilities, which in turn increases each day given the varied set o applications processors must handle. For this reason, we highlight important works that evaluate the impact of the ISA on the microarchitecture of processors. After, we narrow our discussion to the exploitation of partial-ISA, the focus of this work, as an enabler for enhanced MPSoCs configurations.

The work by Venkat and Tullsen (VENKAT; TULLSEN, 2014) exploits the advantages of different ISAs to build an ISA-rich system, which combines cores with the x86-64, ARM thumb, and Alpha ISAs to exploit their different semantic characteristics. They analyze how the different ISAs behave on different aspects such as code density, dynamic instruction count, register pressure, and SIMD and FP support. They claim that having heterogeneous ISA support is beneficial for binding applications along with their execution phases.

The same authors extend their studies in a recent work (VENKAT et al., 2019), proposing a single super-set ISA from which one can derive different subset ISAs. With this proposal, they exploit the different aspects from different ISAs (the same x86, ARM thumb, and Alpha ISAs). The difference in this work is that they consider an extended x86 ISA, which re-implements ARM thumb and Alpha capabilities altogether into a new ISA. They claim this approach overcome their previous work difficulties such as having multiple proprietary ISAs jointly in a chip, and binary translation to execute in each possible ISA. With this setup, authors perform a large DSE, varying both ISAs capabilities and microarchitectural parameters to find the best 4-core heterogeneous ISA Chip Multi-Processor (CMP). In their results, they show that having heterogeneous design with heterogeneous ISA surpasses in 17% heterogeneous designs with a single-ISA on average.

Blem et al. (BLEM et al., 2013) and (BLEM et al., 2015) discuss how the RISC

Figure 2.2: The growing number of instructions in different ISAs.



Source: (LOPES et al., 2015)

and CISC models affect modern architectures. Authors perform extensive experimentation of applications running in different ISAs and microarchitectures in either simulated and real-life environments. By the combination of (VENKAT; TULLSEN, 2014) and (BLEM et al., 2015), authors conclude that choosing between RISC and CISC models is irrelevant to the power and performance of a modern system, and what affects these metrics is the presence or absence of specialized ISA-extensions (such as vector, FP, and crypto instructions). Therefore, Venkat et al. and Blem et al. conclusions are somewhat contradictory. The former defend different ISAs have different impacts, while the later support ISA-extensions are more decisive, regardless the base ISA.

In (LOPES et al., 2015), an extensive analysis of ISA aging and the cost of the decoder for keeping old operations in the architecture set is performed. They analyze how new instructions are frequently added (see 2.2) regardless of the ISA, and claim that new instructions have a higher impact on the decoder because they need extra fields to differentiate them from old ones. The authors propose a technique to remove and recycle instructions that are not used by compilers anymore. Removed instructions that are eventually fetched for execution must be emulated for backward compatibility. By removing the old instructions and re-encoding instructions using their technique, it is possible to reduce the critical path, area, and power consumption of the decoder. On the software side, it can reduce code size, and instruction cache misses.

As we discussed so far, there are plenty of works showing that a diverse and renewed ISA is a viable path for enhancing next-generation processors. From now on, we narrow the discussion to works covering ISA-extensions. Especially, to balance instruction extensions among cores, overlapping-ISAs extensions processors have been proposed. These are processors in which the cores individually implement different extensions, but all share a base ISA. Following, we detail these works.

Li et al. (LI et al., 2010) discuss the challenges of implementing the OS support for

such overlapping ISAs, implementing a technique called *fault-and-migrate* in the Linux kernel 2.6.24. In this technique, when a core fetches an unsupported instruction, the application migrates to a core that supports it. They emulate the existence of asymmetric hardware (i.e., cores with different functionalities, by disabling ISA extension support on real-life Intel processors) to test their proposed fair scheduling algorithm, which extends the time-sharing round-robin model for asymmetric processors. They demonstrate how a real-life OS can be modified to support heterogeneous-ISA with minimal overheads caused by forced migrations when unsupported instructions are fetched.

Reddy et al. (REDDY et al., 2011) claims that in heterogeneous CMPs (like in ARM's big.LITTLE (GREENHALGH, 2011)), the big and the LITTLE cores may have unbalanced support on ISA-extensions implementation, raising software compatibility issues. The work somewhat complements the work by Li et al. (LI et al., 2010), studying how to define what is to be handled by the software (which burdens the programmer) and what is to be handled by the OS in such overlapping-ISA systems. Also, they present design techniques in the OS for bridging software to this possible functionally-asymmetric cores and discuss the pros and cons of each method.

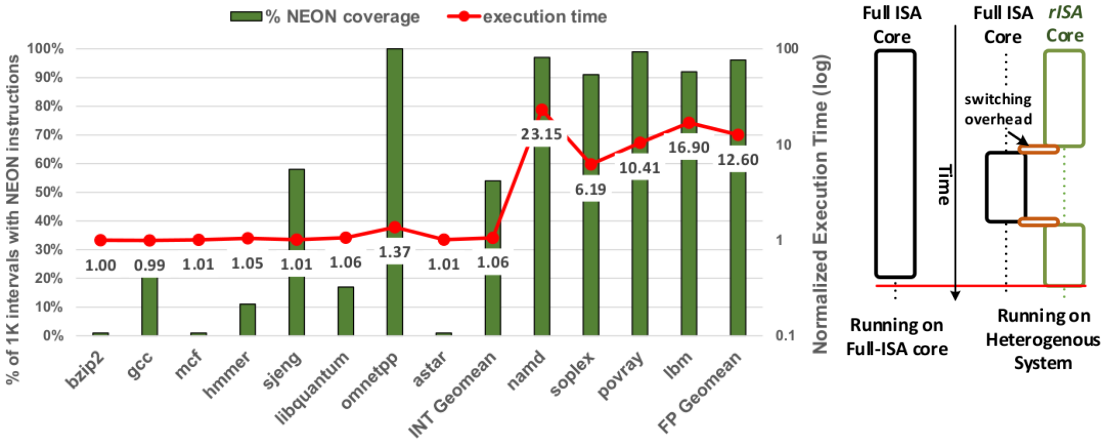
Lee et al. (LEE et al., 2017) observed the hardware power impact from supporting ISA-extensions and proposed a microarchitectural approach to overcome this burden. In Figure 2.3-a, authors present different SPEC2006 benchmarks, the percent of ARM NEON instructions every thousand instructions, and the performance loss (normalized execution time) if removing NEON support. Although sometimes the performance impact is minimal, resulting in energy savings, other situations may present orders of slowdown, especially if NEON instructions are highly used. Similar investigation is done with other ISA-extensions such as *load and store multiple* instructions, *predicated* instructions, and *DSP-like* instructions.

To take advantage of energy efficiency and avoid the performance overhead from removing ISA-extensions support, authors propose to add a *reduced* ISA core altogether with the former *full* ISA core. This proposal appears in Figure 2.3-b. In this way, they use the reduced ISA core whenever possible, and migrate to the full ISA core when an extension instruction is fetched. Through this approach, they reduce performance loss while reducing power consumption considerably. The experiments are conducted with single-task execution on two ARM A15 (one full ISA, one reduced ISA), hence homogeneous microarchitectures. Although they increase the total area of the design, they power-gate the unused processor to have power savings. Since ARM processors (such as A15 ((ARM

Limited, 2012)) and A75 (ARM Limited, 2016)) allow power-gating in the grain of cores, authors cannot turn off the FPU or SIMD datapath separately.

Figure 2.3: Previous work study on ARM NEON instructions impact for different SPEC2006 benchmarks and the proposed approach to overcome performance loss.

(a) NEON Impact (b) Proposed solution



Source: (LEE et al., 2017)

2.3 Work novelty

As we discussed above, different works focused on improving MPSoC designs, and also exploit ISA tuning for enhanced systems w.r.t. related work. In summary, the present work has the following contributions: We,

- *adopt partial-ISA* to save area and power (static and dynamic) on MPSoC designs, different from (LI et al., 2010; REDDY et al., 2011), where they focus on the software/OS support. We also differ from (LEE et al., 2017), which considers partial-ISA in the microarchitecture but causes increased area. Moreover, partial-ISA in an MPSoC context is not studied by any of those works;

- *use the area an power budget* to design richer MPSoC configurations, with more in-order cores and also application-specific accelerators, *respecting* former baseline constraints. Differing from (LEE et al., 2017) where neither in-order cores or accelerators are adopted, nor former constraints are respected. Hereby, while previous work generally leverages partial-ISA to reduce power/energy consumption, our methodology is *able to improve performance* also;

- consider partial-ISA in *simultaneous multi-task* scenarios and *heterogeneous com-*

puting nodes, differently from the single-task in (LEE et al., 2017) migrating among two A15 cores (one reduced and one full ISA); instead our proposal introduces the partial-ISA concept into a complex arrangement, with heterogeneous architectures;

- detail *two mapping strategies* to benefit from the novel design, coping with heterogeneity in both microarchitecture and ISA at the same time, also not covered by previous work;

3 PROPOSED PARTIAL-ISA HETEROGENEOUS MPSOCS

In this chapter, we describe the proposed partial-ISA heterogeneous MPSoCs. We highlight the necessary steps to add more computing nodes, as extra in-order cores or hardware accelerators, and the manners to leverage them for reduced power consumption and improved performance under baseline constraints. We discuss the methodologies for design changes (such as trimming the ISA extensions support), and for mapping workloads (such as task migration mechanisms and the proposed scheduling policies).

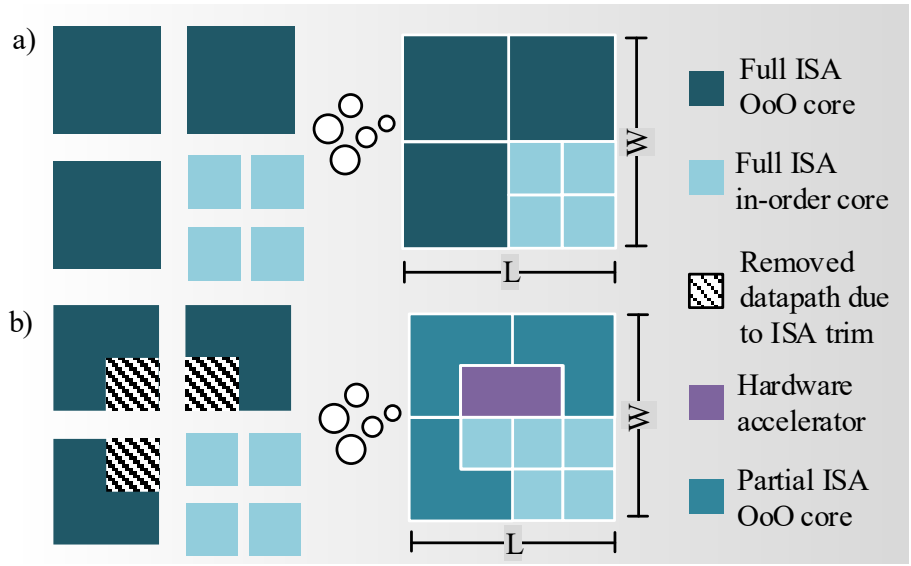
3.1 Architecture of the partial-ISA

Figure 3.1 presents the main concept proposed by this work. It illustrates a scenario where a chip die of length (L) and width (W) constraints the project design. From this, the traditional approach is to accommodate several processors that support the whole instruction set (full-ISA) to utilize the entire area and power budget available, which appears in Figure 3.1-a. We propose to reduce the number of cores that support the entire ISA (a given ISA extension is taken off from some of them), adopting what we call partial-ISA cores. We depict this in Figure 3.1-b. Employing partial-ISA in some cores permits the designer to adopt more and possibly different computing nodes to the system, increasing its heterogeneity.

In this work, we exclude the *FP instructions from the RISC-V architecture* as a use case, which is a high source of logic overhead (ASANOVIC et al., 2015), as we appointed in Table 1.1 (Chapter 1). This simplifies the processor by manifold reasons: the entire FP pipeline is removed from the execution unit of the processor; the decoder stage and the issue queue are trimmed by removing support for these instructions; the register renaming table is simplified, and the FP register file is also removed. We highlight these simplifications in OoO processors in Figure 3.2, adapting the original figure from the work by Smith and Sohi (SMITH; SOHI, 1995). Other secondary hardware components are also indirectly influenced, such as the routing logic, write-back, and forwarding structures.

Nonetheless, in-order cores can also implement Partial-ISA. However, the overall savings in area and power are modest compared to OoO cores. Generally, OoO cores have much more aggressive implementations for ISA-extensions, since they need to be conforming with former complex logic, while also delivering high-performance. For instance, supporting FP in a big (OoO) core occupies 7$\times$ the area of supporting FP in a

Figure 3.1: The system's composition possibilities due to partial-ISA adoption. a) The traditional arrangement. b) Partially trimming the ISA of OoO processors allow adding more computing nodes (in-order processors and accelerators).



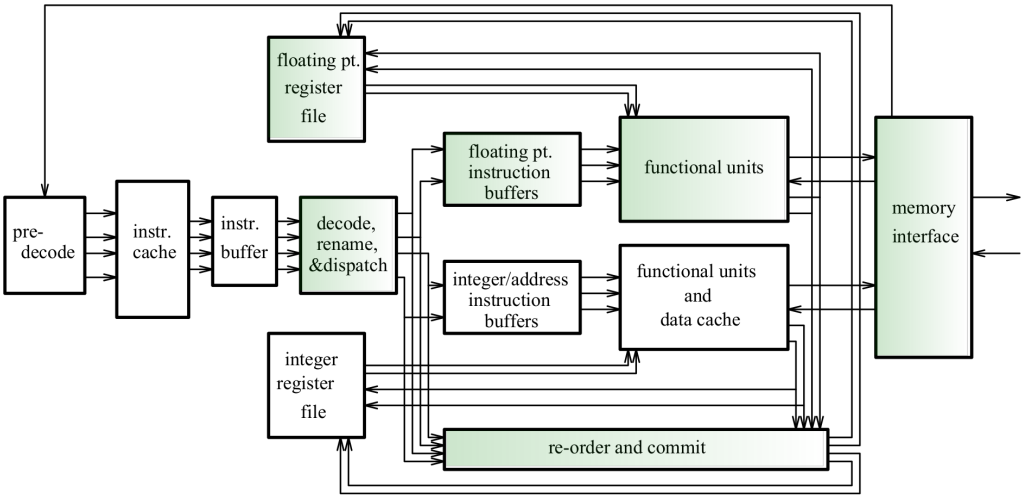
Source: The author.

little (in-order) core, as we discuss in our results (chapter 5). Given that, big cores with partial-ISA creates sufficient design space to improve the system with extra full-ISA little cores or accelerators, while partial-ISA little cores enable the addition of small ASICs only.

Hence, our proposed partial-ISA MPSoC may present ISA heterogeneity (by partially or fully implementing ISA extensions), microarchitectural heterogeneity (in-order or OoO cores), and coupled application-specific accelerators. Importantly, there must always be one or more full-ISA cores in the system, so the MPSoC is capable of executing every instruction from the ISA (by migrating tasks to the full-ISA cores whenever necessary, as we detail further).

This new partial-ISA MPSoC methodology offers several benefits. First of all, it enlarges the design space, so the designer can explore diversified modules to compose the final chip, without exceeding area and power budget. Since partial-ISA cores decrease area and power consumption, the system can leverage an increased number of computing components for task-parallelism and specialization. Additionally, partial-ISA cores can execute extension-free phases from applications with diminished power consumption. In this environment, applications can be assigned to a partial or a full-ISA (depending on the need for ISA extensions or not), in either big cores for performance or little cores (in-order) for energy efficiency, and hardware accelerator when there is one to that specific task, as illustrated in Figure 3.3. Thereby, the system's heterogeneity increases, extending

Figure 3.2: A typical OoO processor datapath. In green, the parts of the datapath that can be trimmed or simplified when removing the floating-point support from the processor.



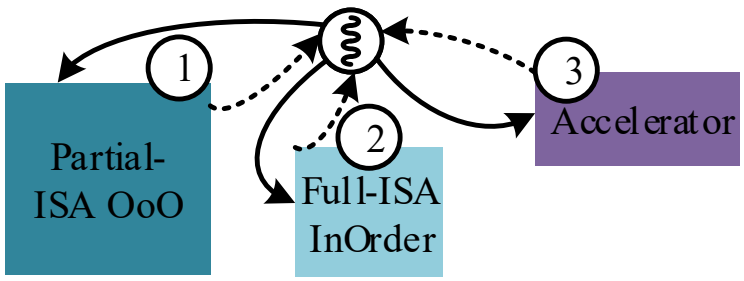
Source: Adapted from (SMITH; SOHI, 1995) by the author.

the binding possibilities among applications and host cores. We can take advantage of this arrangement to either improve energy-efficiency (e.g., using partial cores to execute *integer* applications with the same performance full-cores would) as also performance (e.g., leveraging accelerators or extra cores).

3.2 Task mapping and execution flow

Since partial-ISA MPSoC compositions enable novel task mapping possibilities, an adequate mechanism to coordinate task assignment also becomes necessary. Particularly, we use task migration to exploit available computing nodes. Migration is used to guarantee full-ISA support for applications during FP operation phases, and to attempt

Figure 3.3: Workloads leverage partial-ISA enabled heterogeneity to dynamically execute on a favorable host.



Source: The author.

accelerator usage during accelerated phases of applications (to leverage faster and energy-efficient execution, if an accelerator is available). Such task mapping solutions, we recall, are generally within the system's OS scheduler. For our purposes, however, we let the OS specificity aside, and define a more high-level mechanism, which we call a *System Manager*. As we present in the next chapter, having this definition allows us to investigate the tasks distribution in a partial-ISA system - building a simulated *System Manager* - without the burden of modifying an actual OS.

The *System Manager* coordinates the execution flow of workloads among the computing nodes of the system, through a *Scheduler* and auxiliary mechanisms. The *Scheduler* holds a queue of workloads and the list of computing nodes available in the system. It is aware of which cores are OoO and which are in-order; their supported ISA (full or partial); and the set of available hardware accelerators.

The *Scheduler* takes place periodically and verifies whether there are workloads to dispatch and preempt. The *System Manager* preempts workloads from GPPs after executing for around 160K cycles[1], in a round-robin fashion, avoiding starvation and allowing other workloads to execute on the core. The preempted workloads are pushed into the end of the queue. When a partial-ISA core fetches an ISA-extension instruction (which it does not support) it throws a trap (like the *X86_TRAP_UD* for the x86 ISA in Linux), telling the *System Manager* the workload needs a full-ISA core, marking it as full-ISA dependent. When the Scheduler further selects this workload to dispatch to a core, it is transparently migrated to a full-ISA core. Accelerators, as we explain further, execute their tasks without interruptions.

In the beginning, all cores are idle, so the *Scheduler* picks workloads from the queue, assigning them to one of the available GPP cores. We define two possible policies for mapping workloads to cores: the *performance-oriented* policy and the *energy-oriented* policy. The scheduling policy defines how the *Scheduler* assigns a task when more than one computing node is available, as formally presented in Algorithm 1 and Algorithm 2, and also explained below.

In the performance policy, the *Scheduler* prioritizes using OoO cores instead of the in-order cores. In the energy policy, otherwise, in-order cores are preferred rather than OoO ones. Regardless of the policy, partial-ISA cores are always preferred over their full-ISA counterparts. This is justified since partial-ISA cores are more energy-efficient and often as performing as full-ISA cores. For example, integer phases of applications

[1] A period that introduces minimal impact on performance as suggested by previous studies (CONSTANTINOU et al., 2005).

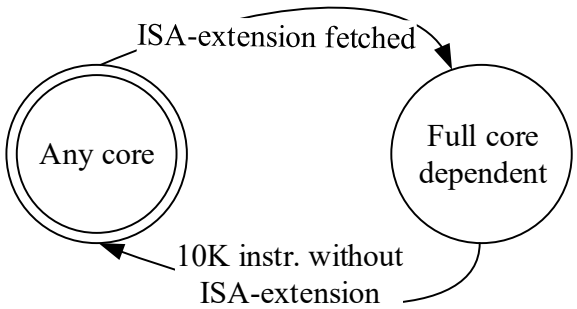
execute in the same way in a partial or a full-ISA core. Because the *Scheduler* does not know when an instruction extension (like a FP operation) will be fetched, it prefers partial-ISA cores until a full core is necessary. Moreover, we relief the use of full-cores, so they are more likely to be available when FP instructions need to be executed. Once an instruction extension is fetched, possibly causing a task migration (as we detail soon), the *System Manager* marks the application as full-ISA dependent. In this way the *Scheduler* avoids misplacing the workload, and maps it to full cores until the ISA-extension is not used for a minimum period[2]. Figure 3.4 illustrates this *full-ISA dependency mechanism*.

Accelerators never start the execution of an application. We notice that application kernels are generally preceded by data preparation, like loading data from disk to main memory and other *system calls* tasks which ASICs may not implement. However, applications that have accelerated target regions (which are marked by special function calls or code annotations) trigger a migration if an appropriate accelerator is available.

If the program reaches an accelerated target region, the application is preempted and marked to execute in the accelerator (similarly to the full-ISA dependency), holding also which accelerator type it requires. If there is a dedicated ASIC for that task available, the *Scheduler* will then prioritize its usage since the execution will be faster and more efficient. Even though the workload is pushed to the end of the queue, in our model, the *Scheduler* traverses the whole queue looking for candidate workloads as long as there are idle nodes in the system, so the accelerated task can be quickly allocated if the accelerator is idle. The accelerator dependency is removed after executing the target region, or as soon as the *System Manager* asserts that there is no accelerator available. As previously stated, accelerators execute the whole target region without preemption. Their execution depends

[2]When an application does not execute an ISA-extension instruction for more than 10K committed instructions, the *System Manager* removes its full-ISA dependency mark, so the application can be allocated in any core. Whenever an ISA-extension instruction is fetched, the mark is set back.

Figure 3.4: On-the-fly management of workloads dependency on full-cores.



Source: The author.

on data feed through specific memory regions (e.g., Direct Memory Access (DMA) (Cota et al., 2015)), and they are not crafted to create a context from their internal architectures to seemly migrate to a GPP in the middle of their computation.

Algorithm 1: Choosing the target core with a performance policy scheduling

```
Data: workload
Result: target core for the workload
if workload.region.canExecuteInAccelerator then
    idleAccelerators = getIdleAccelerators(workload.region.acceleratorType);
    if idleAccelerators is Not Empty then
        targetCore = idleAccelerators.head();
        return targetCore;
    end
end
if workload.canExecuteInAnyCore then
    idlePartialBigCores = getIdlePartialBigCores();
    if idlePartialBigCores is Not Empty then
        targetCore = idlePartialBigCores.head();
        return targetCore;
    end
end
idleFullBigCores = getIdleFullBigCores();
if idleFullBigCores is Not Empty then
    targetCore = idleFullBigCores.head();
    return targetCore;
end
if workload.canExecuteInAnyCore then
    idlePartialLittleCores = getIdlePartialLittleCores();
    if idlePartialLittleCores is Not Empty then
        targetCore = idlePartialLittleCores.head();
        return targetCore;
    end
end
idleFullLittleCores = getIdleFullLittleCores();
if idleFullLittleCores is Not Empty then
    targetCore = idleFullLittleCores.head();
    return targetCore;
end
return Empty;
```

Finally, Figure 3.5 exemplifies the scheduling flow with a simple scenario with three workloads. A system composed of one accelerator and two cores, which diverge by their ISA support for FP instructions. For simplicity, we consider generic cores and do not distinguish them by OoO or in-order. The figure presents ten snapshots of the workloads execution, enumerated in temporally ordered. In the figure, three different

Algorithm 2: Choosing the target core with an energy policy scheduling

```
Data: workload
Result: target core for the workload
if workload.region.canExecuteInAccelerator then
    idleAccelerators = getIdleAccelerators(workload.region.acceleratorType);
    if idleAccelerators is Not Empty then
        targetCore = idleAccelerators.head();
        return targetCore;
    end
end
if workload.canExecuteInAnyCore then
    idlePartialLittleCores = getIdlePartialLittleCores();
    if idlePartialLittleCores is Not Empty then
        targetCore = idlePartialLittleCores.head();
        return targetCore;
    end
end
idleFullLittleCores = getIdleFullLittleCores();
if idleFullLittleCores is Not Empty then
    targetCore = idleFullLittleCores.head();
    return targetCore;
end
if workload.canExecuteInAnyCore then
    idlePartialBigCores = getIdlePartialBigCores();
    if idlePartialBigCores is Not Empty then
        targetCore = idlePartialBigCores.head();
        return targetCore;
    end
end
idleFullBigCores = getIdleFullBigCores();
if idleFullBigCores is Not Empty then
    targetCore = idleFullBigCores.head();
    return targetCore;
end
return Empty;
```

symbols represent the three workloads.

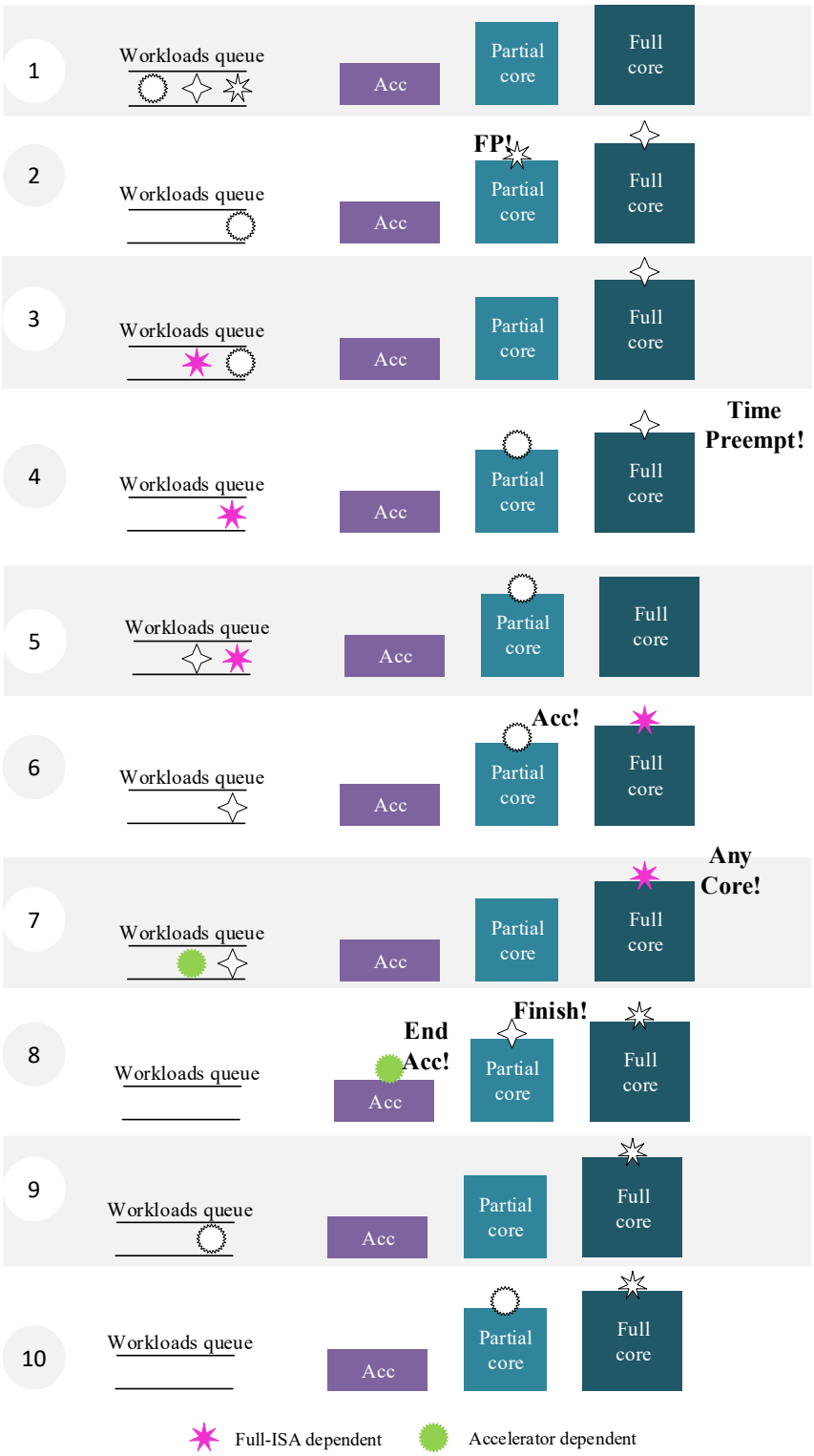
In the beginning (1), all workloads are in the *Workloads queue*. In the next moment (2), both GPP cores start the workloads execution. Note that the workload in the head of the *Workloads queue* was assigned to the partial-core, since these cores are prioritized regardless of the policy. Recall since we do not distinguish the core's type, we consider them to be of the same kind (either in-order or OoO).

At the end of the second snapshot, the workload in the partial-core fetches a FP instruction. This causes it to be marked as *Full-ISA dependent* (pink color as in the legend) and to be pushed to the end of the queue (3). Since the partial-core is free, it can receive the subsequent task (the circle, in snapshot 4). At the same time, we depict a time preemption in the full-core (which occurs after 160K cycles of execution). This forces the workload to be pushed to the end of the queue (5).

Since the full-core is now available, the *Full-ISA dependent* workload, now head of the queue, can be assigned to it (6). Meanwhile, the application in the partial-core (the circle) finds a mark for an accelerated region (6). The workload is assigned as *Accelerator dependent*, and pushed to the end of the queue (7). In snapshot (7), the *Full-ISA dependent* workload notifies the *Scheduler* that it can be assigned to any core again, removing the *Full-ISA dependency* mark, after executing for 10K cycles without any FP instruction.

Subsequently, the head of the queue (four dot star in snapshot 7) is pushed to the available partial-core (8). At the same time (8), the *Scheduler* assigns the *Accelerator dependent* workload to the propitious accelerator (the accelerator type is also suppressed for simplicity). Also, the workload in the partial-core finishes its execution, while the *Accelerator dependent* reaches the end of the accelerated region, being pushed to the queue (9). The *Scheduler* reassigns it to a core (10). This flow continues from this point on until the all workloads finish their execution.

Figure 3.5: An example of the scheduling flow in a system with two cores (differing by partial-ISA adoption) and one accelerator.



Source: The author.

3.3 Design effort and challenges

Partial-ISA intrinsically delivers sub-set microarchitectures, and thus increases the design effort (compared to a homogeneous system). However, in this work, we exemplify how to remove ISA extensions from a processor's datapath as also to adjust other microarchitectural structures (e.g., cache size, issue-width) using core generators. Particularly, we synthesize real-life RISC-V processors with BOOM (OoO) (ASANOVIC et al., 2015) and Rocket (in-order) (ASANOVIC et al., 2016) core generators to analyze power and area. These parametric tool-chains process high-level hardware description languages (namely Chisel, similar to Scala), translating the description to Verilog, a standard and lower-level hardware description language (compared to Chisel). This favors the designer to adjust the processor accordingly to its needs. For example, Listing 3.1 depicts how we define a class of processors without a FPU in the Chisel language, used by the aforementioned tools. Hence, we can use these core generators to create cores without FP support, having partial-ISA cores.

Furthermore, design efforts are natural in the design flow of a chip. Indeed, the industry offers microarchitecture customization; for instance, some ARM processors can be delivered with or without a NEON unit (although never within a partial-ISA context as in our proposal, where the ISA is heterogeneous). Also, Sun's UltraSPARC T1 had a single loosely coupled FPU shared among cores (the high latency for the loosely coupled accesses proven too costly later, however).

```
1 class WithoutBoomFPU extends Config((site, here, up) => {
2    case BoomTilesKey => up(BoomTilesKey, site) map { r => r.copy(core =
      r.core.copy(
3       usingFPU = false))
4    }
5 })
```

Listing 3.1: Defining a class of RISC-V BOOM cores without Floating-Point support.

Nevertheless, partial-ISA requires specific support from the hardware. For instance, if partial-ISA cores fetch an unsupported instruction, they need to generate a trap for proper handling. Fortunately, modern processors already have a trap generation scheme used when unknown instructions are fetched. We leverage this mechanism to notify the *System Manager* when partial-ISA cores do not support an extension. We also consider a simple hardware-counter module to accumulate the timespan without executing ISA-extensions, to inform the *scheduling* when the ISA-dependency can be removed.

4 SIMULATION TOOL-CHAIN

Since partial-ISA requires modifications in both the hardware and the task management (e.g., OS) and we do not have such a real-life system, we perform our analysis in a simulated environment. In this chapter, we overview how the simulation environment is set up and how we combine data from different tools to evaluate our proposal.

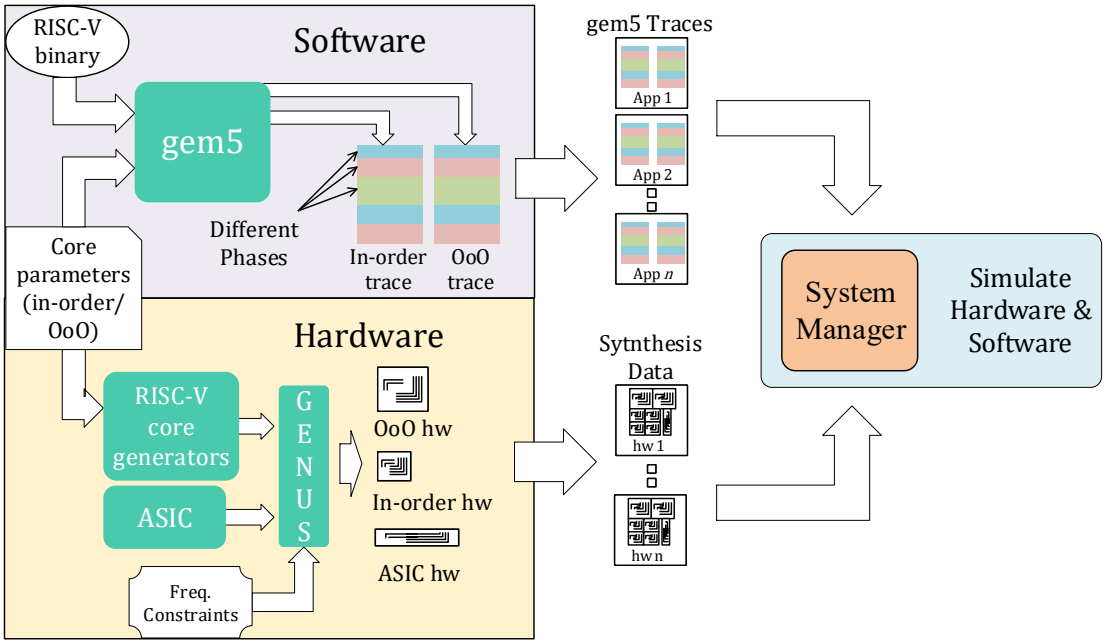
Figure 4.1 presents a high-level diagram of the simulation tool-chain we describe in this chapter. In the following sections, we will refer to this diagram, explaining all the necessary steps to simulate the execution of multiple applications on a partial-ISA MPSoC. Namely, we detail how we acquire area and power data, how we collect workloads execution behavior, how we combine these data to build our partial-ISA MPSoC simulator, and how we set up important configurations for experimentation.

4.1 Acquiring area and power data

Partial-ISA adoption aims to facilitate the composition of enhanced MPSoCs respecting a given power and area budget. Because we want to experiment with different MPSoC configurations, containing a variety of computing nodes, it is necessary to know how much power and area each of these nodes consumes. To accurately gather this information, we use the Genus Synthesis Solution from Cadence, a newer version of the well-known RTL Compiler, from the same company. In this tool, we use a contemporary 15nm standard cell library (MARTINS et al., 2015) for the synthesis. With this setup, we provide the hardware description we want Genus to synthesize (i.e., a digital system described in Verilog or VHDL), together with constraints as the target clock frequency. This appears at the bottom of Figure 4.1. The tool processes the description, transforming it into a gate netlist, optimizing the design to meet the target frequency (values are detailed next). When it finishes, the tool reports post-synthesis metrics on mean power and area usage.

However, the methodology above depends on the existence (and availability) of a hardware description for each node we want to synthesize. This is not a problem for the GPP RISC-V cores, since we use open-source hardware generators to get their description, as we detailed in section 3.3. Sometimes, however, we had to consider data from previously published works whose hardware description is not open-sourced, such as recently proposed accelerators that present state-of-the-art solutions in trendy fields. This

Figure 4.1: The simulation tool-chain used in this work.



Source: The author.

is the case for a Convolutional Neural Network (CNN) accelerator we use in our experimentation (chapter 5). Because a synthesis is not possible, we scale data reported by the authors to the 15nm technology node (considered for every hardware in this work), for a fair analysis. Conversions are based on the recent work from Stillmaker (STILLMAKER; BAAS, 2017), which provides scaling data ranging from 180nm to 7nm technologies.

4.2 Profiling and tracing workloads execution phases

As we discussed in Chapter 3, the partial-ISA MPSoC execution flow depends on a few hardware triggers for proper functioning. For example, the cores must notify the *Scheduler* when there are workload dependencies for an ISA-extension datapath (as a FPU), and also when an application reaches an accelerated region. Thereby, it is mandatory to have those triggers information to simulate task migrations in an accurate manner. Also, because the MPSoC has heterogeneous cores, and workloads can execute on any of them, our proposed simulator must address the different performance that applications have in the different cores available. For example, a workload can execute quickly on an OoO but slowly on an in-order core. Therefore, these previous observations suggest for us to build our simulated environment using dynamic instrumentation.

To collect the behavior of the workloads under different host cores, we use the

gem5 cycle-accurate microarchitecture simulator (BINKERT et al., 2011). The gem5 is capable of executing binary code from different ISAs, including the RISC-V used in this work. It can be used as a tool for measuring stats of applications (like the execution time, the number of committed instructions, the L1 cache miss-rate) in different organizations. More importantly, its open-source code models cycle-accurate in-order and out-of-order cores and allows internal modifications for detailed on-the-fly execution profiling.

For our purposes, we have modified the gem5 simulator so it traces the execution of RISC-V applications while executing workloads on both OoO (big) and in-order (little) cores. This appears on the superior part of Figure 4.1. As it appears in the Figure, the traces created during gem5 profiling hold information regarding different phases of the workload's execution. Particularly, these phases will be consulted by our simulated *System Manager*, so it knows: (i) which portions of the program require extensions support (to mark applications as ISA-extension dependent during simulation); (ii) which portions of the program are accelerated regions (to mark applications as accelerator dependent during simulation). For this, we modify the applications with regions of interest for accelerators with annotation. When executing the applications, gem5 dynamically retrieves these annotations to generate the traces accordingly; (iii) whenever the program had executed for longer than 10K cycles without extension instructions (to remove the ISA-extension dependency during simulation, recall Figure 3.4). With the aforementioned instrumentation, gem5 traces contain dynamic information of the hardware triggers we expect from cores in our partial-ISA MPSoC proposal.

To understand how the profiling traces are internally created and how we leverage their content information, Figure 4.2 depicts two traces for a given workload: one for its execution in a big core (OoO), and another for its execution in a little core (in-order). Each trace is composed of a set of blocks, representing the intervals of the application's execution. As the legend of the Figure describes, these blocks represent the intervals of the application with integer-only instructions, with ISA-extensions instructions (FP in this work), and also the accelerated regions. Importantly, the blocks hold the number of cycles and instructions these execution intervals took to perform, depending on the host core.

To generate the traces, gem5 counts the number of committed instructions from the beginning to the end of an execution interval, on both OoO and in-order simulations. The blocks usually represent up to 10K instructions[1], and are successively reported covering

[1]The size of the blocks must be small for fine-grain representation of the execution phases, but not too small causing traces to be composed of many of them, turning the traces into big files (which also overheads the traces parsing in our simulator, as we explain in section 4.3).

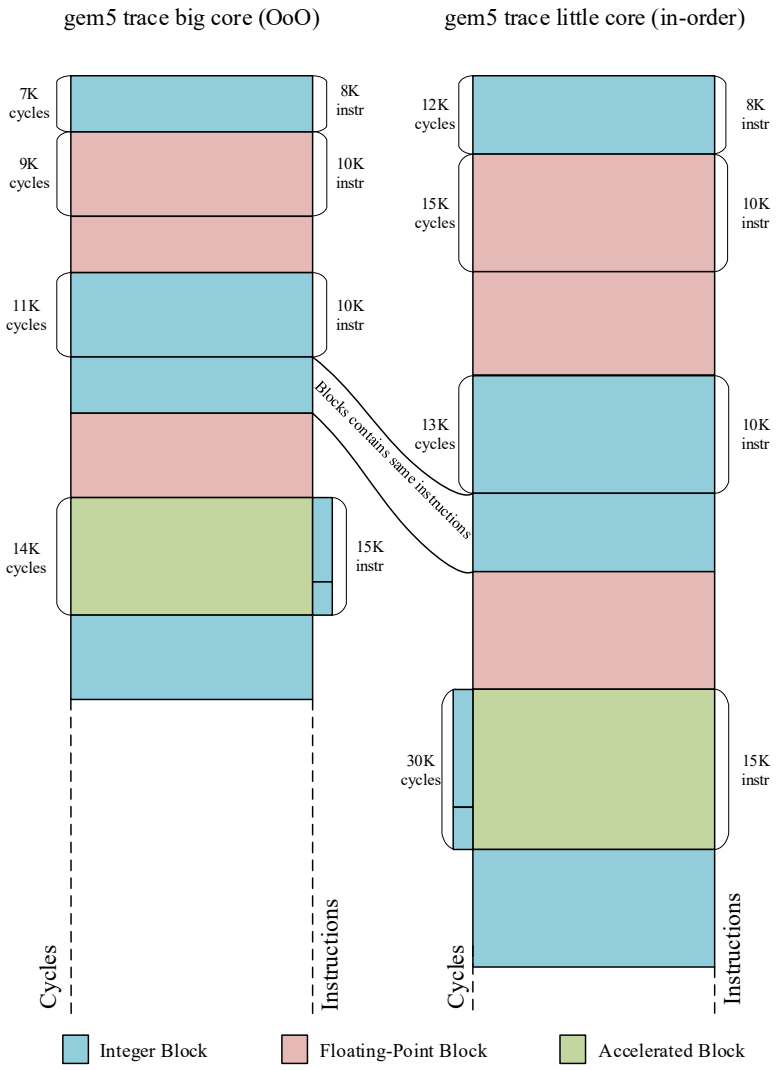
the whole execution of the application. Blocks can be shortened, however, when an ISA-extension is fetched, which closes an integer block and immediately starts a FP block. This is depicted at the end of the first block of both traces in Figure 4.2. When gem5 counts more than 10K non-extension instructions, it closes the FP block and starts an integer block, emulating how a core would trigger the removal of the ISA-dependency from an application in our Partial-ISA proposal.

The accelerated regions also lead to a special case for the block's instruction count. Because accelerators execute the whole accelerated region (recall the Partial-ISA scheduling policies in section 3.2), gem5 does not limit the size of these blocks. This appears in Figure 4.2, in the green blocks representing accelerated regions. However, since we may not have an accelerator for the accelerated region, we keep tracing the execution of the region in the gem5 cores *inside* the accelerated block, so we can also use this information to simulate the execution of the accelerated region in a GPP, if necessary[2]. This is also depicted in the Figure, where the accelerated block wraps non-accelerated blocks (detailed in the border of the accelerated blocks). In time, we recall gem5 gathers the execution time of the accelerated block executing in general-purpose cores only. Our simulator considers the nominal speedup of the hardware accelerator over the big core, to obtain the execution time of the accelerated block on an ASIC, as we specify later in this chapter.

Importantly, we guarantee the blocks represent the same portion of a program execution regardless of the host core, i.e., the number of total blocks and the instructions they represent are the same in both OoO and in-order traces. This appears in Figure 4.2, observing the block's amount of instructions, depicted on the right side of the blocks in the traces. Since both microarchitectures commit instructions in order, the Nth committed instruction of a given application is the *same* for both cores. Since our modified gem5 counts instructions, its internal counters will be incremented symmetrically in both in-order and OoO simulations, for a given workload. Hence, the Kth block in the big core trace is equivalent to the Kth block in the little core trace. This is important because it assures that, at the end of each block, the workload is at the same point of execution in both traces. Thereby, our proposed simulator can read the big core trace up to a point, and continue reading from that point ahead in the little core trace, which allows us to simulate migrations coherently, as we detail further (section 4.3).

[2]We highlight that executing in accelerators and GPP requires different binaries. We consider this to be available since both GPP and accelerators are already in use today, with plenty of tools for binary generation. For instance, different binaries optimized for different architectures can even be within the same binary with function-multi-versioning (PARK et al., 2019; JIMBOREAN; LOECHNER; CLAUSS, 2011).

Figure 4.2: Representation of the execution traces of a workload in both a big and a little core, generated with the gem5 simulator. The current host core of the workload determines which of the traces will be consumed for a given slice of the execution.



Source: The author.

What can (and generally will) vary is the amount of time it takes to execute a block depending on the host core. For example, big cores can achieve higher Instruction-Level Parallelism (ILP) exploitation to commit the instructions of a block faster than the little core. We also illustrate this in Figure 4.2, presenting the cycles for executing the blocks (varying with the core). This is used by our simulator to extract the performance difference among the different cores.

Finally, with gem5, we have also assessed the required time to flush a cache and refill it with new data, which we considered for the migration costs (e.g., when dispatching workloads, as we explain in Section 4.3). We base this assumption based on a previous

work by Li et al. (LI et al., 2007), which claims cache overheads are dominant for task migration.

4.3 Modeling the *System Manager* for a multi-task simulation

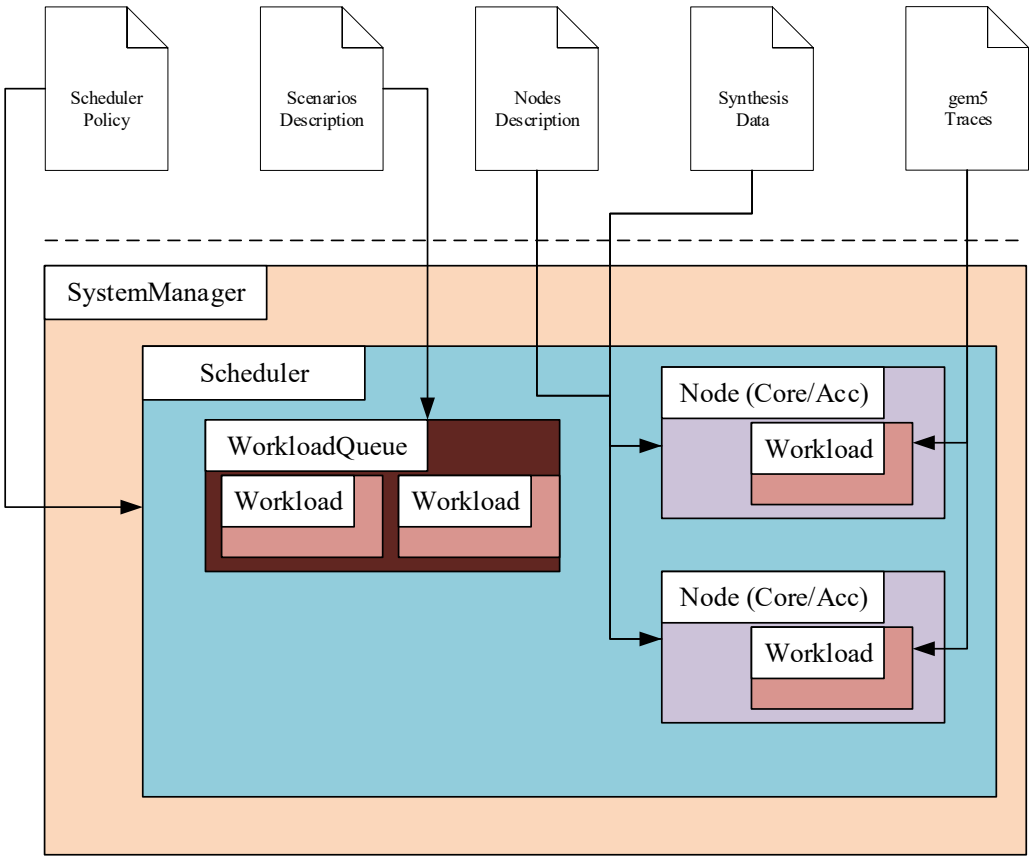
Although we execute every workload in gem5 to generate its execution traces, it is also necessary to consider its execution in a multi-task environment, where multiple workloads share the MPSoC resources. For that end, we developed an in-house simulator to model a *System Manager* (as described in Chapter 3.2). Figure 4.1 presents a comprehensive overview of the tools and simulation flow. Through the steps described in Section 4.1, we provide hardware-related data from synthesis to the *System Manager*, so it has information regarding the area and power of the computing nodes it is simulating. At the same time, through the steps described in Section 4.2, we provide software-related data from gem5 to the *System Manager*, so it has the executions traces of the applications to properly simulate the execution of the workloads under the available hardware. Internally, the *System Manager* implements the *Scheduler*, the *Cores*, and all auxiliary modules to perform accurate simulation, as we describe next. With this, we can verify the impact of having partial-ISA cores, and the extra computing nodes (as in-order cores and accelerators) in heterogeneous MPSoC designs.

Figure 4.3 presents an overview of the simulator components and inputs. As in the Figure, the *System Manager* module wraps the *Scheduler*. The *Scheduler*, in turn, has a list of workloads and the reference for available nodes in the system. Note that workloads can be in the queue, or executing in a computing node. The mapping strategy to assign a workload from the queue to an idle core is implemented accordingly with the scheduling policy, detailed in Chapter 3.2.

The diagram illustrated in Figure 4.3 also presents (on the top) the necessary inputs for accurate experimentation. They are used in the following manner:

- **Scheduler Policy.** The policy is defined at the initialization of the simulator. With this input, it is possible to choose between the performance-oriented or the energy-oriented mapping strategies. Also, the existence of this input allows the simulator to extend the number of policies while keeping an easy interface. We implement the scheduling policies inside our simulator, based on Algorithms 1 and 2. We detail how the scheduler interacts with the remaining modules of our simulator further in

Figure 4.3: A high-level view of the partial-ISA MPSoC simulator. On the top, the set of inputs necessary for execution. Above, the different modules of the simulator and their interaction with each other and with the inputs.



Source: The author.

this section.

- **Scenarios Descriptions.** This input contains the list of workloads we want to execute in the simulator, allowing us to create multi-tasking execution environments. The list of workloads is used to fill the *WorkloadQueue* when the simulator starts the execution. This input is simply a *json* file which holds the workloads names, and the reference for the trace files from gem5 (another input of the simulator as we detail further, and as also depicted in Figure 4.1). With this approach, we can quickly create different scenarios for experimentation.

- **Computing Nodes Description.** This input is a *json* file containing each computing node in the system and its microarchitecture since it is important to inform the simulator of the computing nodes available. Especially, the *Scheduler* may leverage this information when applying its current policy. In the *json* file, each element can

be a GPP or an accelerator. If it is a GPP, it can be defined as an OoO or an in-order core, as also the ISA-extensions it supports (or does not support, if partial-ISA). If an accelerator, it contains the accelerator type, so the *Scheduler* knows which accelerated regions can be assigned to the accelerator, and the accelerator speedup compared to a big (OoO) core, used as a reference for its performance because gem5 does not generate traces for the accelerators (this will be detailed further in this chapter). Configuring the elements in the *json* file is all it takes for adjusting the composition of the MPSoC computing nodes.

- **Synthesis Data.** We use the synthesis outcome to feed our simulator (as also appears in Figure 4.1) with area and power information. With this, the simulator can compute the total area of the MPSoC, and also have the mean power of each computing node at hand. We sum up the energy consumed by each computing node along with the execution of the workloads using data from the *System Manager*, which knows whether or not the cores are executing.

- **gem5 Traces.** As previously explained, gem5 traces are used as a trustful representation of a workload executing in a core. In our environment, a workload can execute in either big or little cores (OoO or in-order), or accelerators. Because of this, for each workload we want to simulate, we need (and have) two gem5 traces. One contains the execution trace for the workload under an OoO core, and the other has the execution trace of the workload in an in-order core. Depending on the workloads' host core (decided on-the-fly by the *Scheduler* during our simulation), the appropriate trace will be consulted to advance the workloads' execution accordingly. When the *Scheduler* assigns a workload to an accelerator, we use the big core trace and the accelerator nominal speedup to simulate the execution.

The execution of workloads in our simulation proceeds after we have these inputs at hand. With the *Workloads Queue* filled with workloads, the *Scheduler* assigns tasks for the available computing nodes, accordingly with the chosen scheduling policy, following the specifications in Chapter 3. For such, we carefully assure all the scheduler restrictions defined in Section 3.2 are respected in our implementation (e.g., never start the execution of a workload assigning it to an accelerator). When a simulated *Core* has been assigned with a workload, it can execute it. For this, the host *Core* module verifies its type (big or little). Based on that, the *Core* looks up in the corresponding trace (big core trace or little core trace, depending on its type) and gathers a block, checking the execution interval of

it. This is used to append execution time in the total time of the workload's execution, and to add up the energy consumption for the block execution on that core. Also, it is used by the *Scheduler* to know what cores are occupied or idle at a given time. At the end of the block, the *Scheduler* verifies for how many cycles the workload has been executing on the core. If it surpassed the threshold of 160K (see Chapter 3), the workload is preempted from the core and pushed to the *Workloads Queue* module.

If the workload is not preempted, the simulated *Core* looks up for the next block. If the next block is different from the previous block (e.g., it is marked as *ISA-extension dependent*), the core check if it is capable of executing it. If it is, the same process as above is repeated; if it is not capable of executing the block, the workload is preempted (removed from the core and pushed to the *Workloads Queue*), to be later assigned to a capable host. When the *Scheduler* assigns an accelerated block to execute in the appropriate hardware accelerator, we use the nominal speedup of the accelerator compared to the big core to get the number of cycles it takes to execute the accelerated block, as stated by Equation 4.1. The specific values of the ASIC speedups are discussed along with the methodology for our results, presented in the upcoming section.

$$accelerator\ block\ execution\ time = \frac{block\ execution\ time\ in\ big\ core}{accelerator\ nominal\ speedup\ over\ big\ core} \tag{4.1}$$

4.4 Results Methodology

Below, we describe how we have set up important knobs used in our simulations to examine partial-ISA impacts in MPSoCs. These adjustments hold for all experiments, which we introduce in the next chapter.

4.4.1 Partial-ISA Cores

To evaluate our proposed MPSoCs with partial-ISA processors, we have used the Rocket (for in-order cores) (ASANOVIC et al., 2016) and BOOM (for OoO cores) (ASANOVIC et al., 2015) RISC-V core generators. Both generators provide a tool-chain for a parametric generation of synthesizable Verilog code for RISC-V based processors.

We have parametrized our cores to be alike ARM's big.LITTLE cores (OoO as A15 and in-order as A7) in these RISC-V tools, based on previously published data (GREENHALGH, 2011; ENDO et al., 2015). For the partial-ISA cores, the FPU (as well as all the related hardware) was removed, through the configuration files as we detailed in Listing 3.1 (Chapter 3). The detailed list of the configured parameters for each core can be found in Table 4.1.

As detailed in section 4.1, we provide the core generator's resultant Verilog to the Cadence Genus synthesis tool with the Verilog using a 15nm cell library (MARTINS et al., 2015). Additionally, for estimating the area and power of L1 caches, we use FinCACTI (SHAFAEI et al., 2014) (under the same technology).

As previously explained (section 4.2), we have used the gem5 simulator (BINKERT et al., 2011) for performance analysis, running RISC-V binaries to generate execution traces. The cores in gem5 were modeled with the same parameters used on Rocket (for little cores) and BOOM (for big cores) tool-chains (which appears in Table 4.1), and applications were compiled with *riscv-gcc* using the -O3 flag.

4.4.2 Application-Specific Hardware

Additionally to GPP cores, we also consider the use of hardware-accelerators (ASIC) as an option for increased heterogeneity that can be leveraged in MPSoCs through partial-ISA. However, neither core generators (to model area and power) nor gem5 (to model performance) provides straightforward support for hardware-accelerators as it does for GPP cores. For this reason, we have a different approach to model ASICs.

We consider three hardware-accelerators in this work - the reasons are discussed in the Results Chapter: a decoder for the H.264/AVC digital video compression standard (named Nova); an accelerator for convolutional neural networks (named Origami); and an accelerator for the AES cryptography standard (named Avalon). Both the H264 decoder (XU; CHOY, 2008) and the AES accelerator (RUSCHIVAL, 2017) were obtained at the *opencores.org* website, an online repository for open-sourced hardware. Consequently, we have access to the hardware description of both accelerators and could submit them to the Cadence Genus to get their area and mean power (refer to Section 4.1). However, the CNN accelerator that we consider in this work comes from the work in (CAVIGELLI et al., 2015). The published results contain data on area, power, and performance, but do not provide a hardware description. Because the work presents data in 65nm technol-

Table 4.1: Modeling parameters for the big (OoO) and little (in-order) cores.

Parameter		**Big core (BOOM)**	**Little core (Rocket-chip)**
Frequency		2GHz	1.4GHz
L1 - I	Size	32kB	32kB
	Associativity	2	2
	MSHRs	2	2
	TLB Entries	32	32
L1 - D	Size	32kB	32kB
	Associativity	2	4
	MSHRs	6	4
	TLB Entries	32	32
BTB Entries		2048	128
Fetch Width		4	1
Decode Width		4	1
INT/FP Pipeline Depth		10/14	5/8
INT/FP Physical registers		90/256	32/32
FP Units		2	1
Issue Queue Size	Mem	20	–
	ALU	20	
	FP	20	
ROB Entries		64	–

ogy, we had to scale their results to the 15nm technology, which is used in all remaining hardware of this work (either accelerators or cores). We perform the scaling based on (STILLMAKER; BAAS, 2017), as aforementioned in Section 4.1.

Given that our simulator uses gem5 traces for performance evaluation, and traces in gem5 are generated by core models, we cannot generate accelerated traces from the accelerators in gem5. Thus, we define a mathematical method to stipulate the accelerator's performance, using the nominal speedup against the big core, as in Equation 4.1. The execution time in the big core, used by the equation, is provided by gem5, but the accelerator nominal speedup over the big core has not been yet defined. Following, we detail how we calculate the nominal speedup for the aforementioned accelerators.

H264 Decoder. We use latency and frequency from the synthesized H264 decoder to calculate the time to execute three frames of video in the QCIF video resolution, defined by the H264 standard (RICHARDSON, 2010). Each of these frames contains 11 $\times$

9 *mega blocks* of video (each mega block contains 16 pixels $\times$ 16 pixels), summing up to a total of 297 mega blocks. The datasheet of the H264 accelerator states that each mega block takes 204 cycles to be decoded. By considering the synthesized frequency (200 MHz, reproducing orignal authors decision), a total of 0.3ms are necessary to decode the three frames. When performing the decoding kernel (excluding file loading and initialization) for three frames of a video in equivalent quality in the big core, gem5 reported 22.7ms execution time - a difference of about 75$\times$.

AES Cryptography. To define the nominal speedup of the AES accelerator, we perform a similar approach. We execute encryption and decryption of a 40kB message in gem5, which results in 3.22ms for the big core. We consider the same message to be encrypted or decrypted in the hardware-accelerator. Referring to Avalon's documentation, we check it takes 10 cycles to process each 128 bits and also runs at 200MHz (as original authors defined). This leads to 0.125ms to encrypt or decrypt the same amount of data (the Finite State Machine (FSM) for the encrypt and decrypt take the same amount of steps accordingly to the accelerator manual). The performance gap, in this case, is around 25$\times$ between the ASIC and the software approach executing in our big core.

Convolution Neural Network. Because we do not have access to the hardware description of the CNN accelerator, we must refer to the data provided by the authors in the original paper where the accelerator is described (CAVIGELLI et al., 2015). Ideally, the speedup of the accelerator over the big core is given by the equation:

$$Speedup_{cnn\ acc\ over\ big\ core} = \frac{time_{big\ core}}{time_{cnn\ acc}} \tag{4.2}$$

Unfortunately, authors do not provide the execution time of the accelerator to complete a task, so we cannot reproduce its execution in the big core to get a direct comparison. What is given, however, is the throughput of the accelerator in terms of operations per second (ops/s), where operations refer to Multiply-Accumulate (MAC) in the convolutional layer of a neural network. To have a comparison metric, we run a CNN benchmark in the big core (detailed in the scenarios, later) and obtain the throughput in terms of Instructions per Second (IPC). Considering each instruction as an operation, and knowing the cycle period (in seconds), we obtain the throughput of the core (in instr/s), derived from its IPC. In this point we have the following:

$$Throughput_{big\ core} = \frac{instr_{big\ core}}{time_{big\ core}} \tag{4.3}$$

$$Throughput_{cnn\ acc} = \frac{ops_{cnn\ acc}}{time_{cnn\ acc}} \tag{4.4}$$

To proceed, we make a pessimistic simplification (considering the core better than it is and therefor reducing the speedup gains of the accelerator). We consider most of the instructions committed by the big core to be useful operations (MAC operations of the convolutional kernel). Hence:

$$Throughput_{big\ core} = \frac{instr_{big\ core}}{time_{big\ core}} \approx \frac{ops_{big\ core}}{time_{big\ core}} \tag{4.5}$$

At this point, we observe that the number of MAC operations of a convolution is determined by the neural network topology and by the input size, and does not depend on the target architecture that calculates it. This leads to the following:

$$ops_{big\ core} = ops_{cnn\ acc} = ops \tag{4.6}$$

Applying Equation 4.6 in Equations 4.5 and 4.4:

$$ops \approx Throughput_{cnn\ acc} \times time_{cnn\ acc} \approx Throughput_{big\ core} \times time_{big\ core} \tag{4.7}$$

Which finally leads to:

$$\frac{Throughput_{cnn\ acc}}{Throughput_{big\ core}} \approx \frac{time_{big\ core}}{time_{cnn\ acc}} \tag{4.8}$$

Therefore having the speedup relation required by Equation 4.2, expressed in terms of throughput, which we have at hand. With data from the accelerator (203Gops/s throughput) and the big core via gem5 (2.08Ginstr/s $\approx$ 2.08Gops/s throughput), we have a 100$\times$ speedup relation.

Even though accelerators' performance definition contains simplifications, we reach plausible values of speedup, which could appear in real-life implementations. Accelerators are designed for delivering orders of magnitude higher performance and lower energy against GPP solutions since they are specifically tailored for a single end. Hence, we consider that our approach is a plausible way of assessing the benefits in performance when using hardware-accelerators. In time, we plan to simulate hardware accelerators executing the kernels in our future work, improving the experimentation accuracy.

5 RESULTS

In this chapter, we present and discuss the results of our proposal. We start by defining the *execution-scenarios* and *hw-configurations*, which are combined to evaluate the behavior of partial-ISA MPSoCs under different circumstances. After, we begin the analysis of the results evaluating the scheduling mechanism, presenting how workloads are mapped through the execution of our scenarios. Next, we discuss the partial-ISA impacts in performance and energy, using the observations from the scheduling analysis and from the computing nodes adopted. We close the Chapter presenting data on EDP of the partial-ISA configurations. As we conclude, our proposal consistently delivers enhanced configurations regarding the EDP metric.

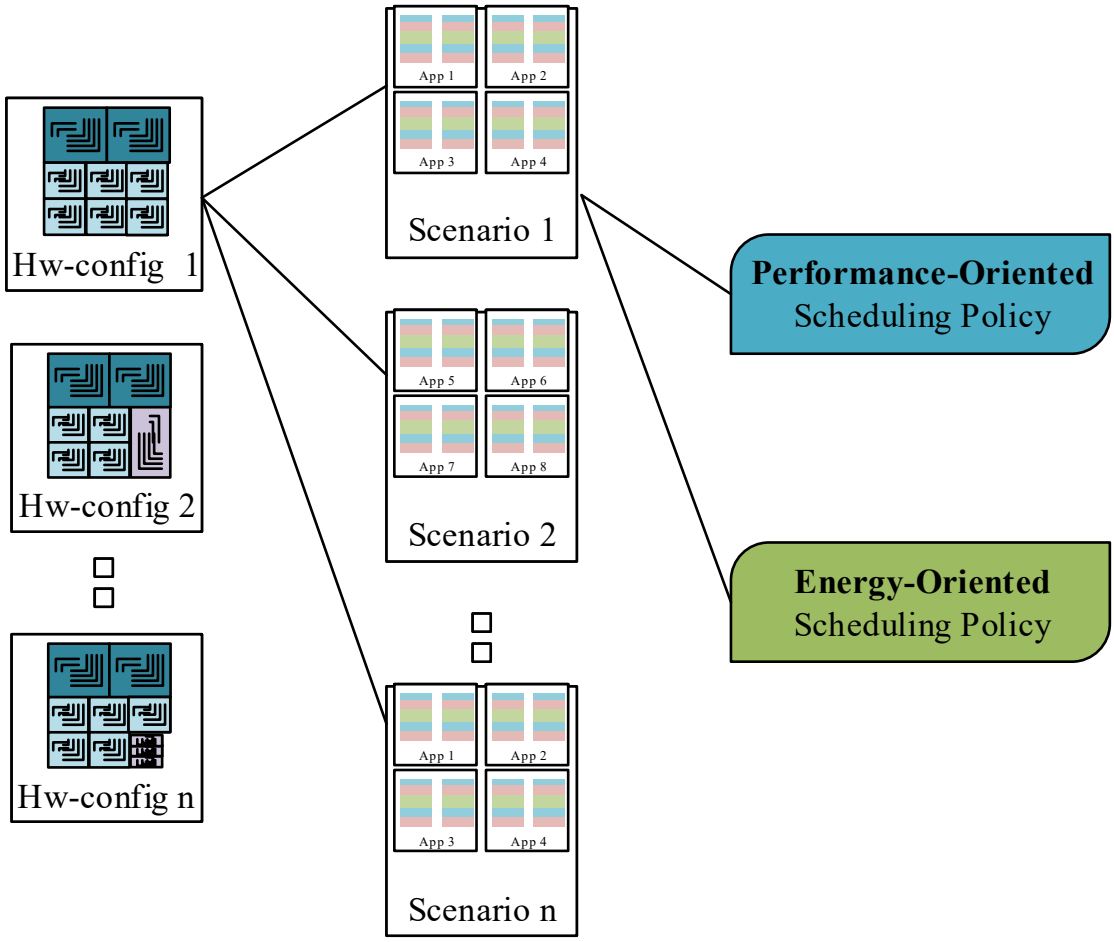
5.1 Scenarios and Configurations

To evaluate Partial-ISA adoption for enhanced heterogeneous MPSoC designs, we investigate several possibilities. For instance, depending on the number of cores we turn into Partial-ISA, different design spaces arise. Since we can compose different designs, they may perform differently depending on the applications' requirements. Sometimes it is better to leverage partial-ISA to add more GPP in-order cores, increasing the MPSoC's task-parallelism. In other cases, adding an application-specific accelerator can be more suitable. Naturally, removing the ISA-extension support from a few cores can also impact negatively on the overall performance, if those extensions are frequent in the applications being executed. For this reason, we must create different application pools, and verify how different MPSoC designs perform on each of them.

Figure 5.1 depicts our methodology for this experimentation. We call each MPSoC design a *hw-configuration*. Similarly, we have the *executing-scenarios*, which are a pool of applications representing some real-life situation. As we depict in the figure, each *hw-configuration* runs each *executing-scenario*, under both scheduling policies, such that we have a broad investigation regarding Partial-ISA usage.

Following, we enumerate the chosen Scenarios and justify the applications we select. Also, we present the Baseline MPSoC design and the Partial-ISA MPSoCs derived from it, which respect the Baseline's area and power budget.

Figure 5.1: A representation on how we extensively combine hardware configurations, software scenarios, and scheduling policies throughout the experiments.



Source: The author.

5.1.1 Scenarios

We use a variety of applications from different benchmarks suites to evaluate our partial-ISA MPSoC proposal. With the intent to mimic real-life situations, we group workloads into different sets, which we call *Executing-Scenarios*, to perform under our simulator. Table 5.1 presents six scenarios, their set of workloads, and the manner we consider applications execute (at the same time as in a pipeline, or at the same time as independent and simultaneous tasks).

Next, we explain the motivations for each scenario:

- **Smartphone app.** This scenario contains a set of daily use tasks, such as text-to-speech, image processing, and string manipulation. This scenario aims to represent, as the name states, the execution of apps in a smartphone. We consider tasks execute independently from each other, hence the *simultaneous tasks* execution mode.

Table 5.1: The applications scenarios evaluated in this work.

Scenario	Tasks	Execution Mode
Smartphone app	rsynth qsort jpeg-d susan-c susan-e susan-s stringsearch	Simultaneous Tasks
Multitask	Mix of more than 30 fp and int instances of MiBench benchmarks	Simultaneous Tasks
Edge computing	2dconv astar svm histogram ecg_health dwt_compression dtw_pattern matching aes_encrypt aes_decrypt	Simultaneous Tasks & Pipeline
Road sign detection	aes crc32 cnn_inference	Pipeline
Video streamming	aes crc32 h264_video_dec	Pipeline
FP-driven	correlation gemm 2mm 3mm cholesky ludcmp gramschmidt jacobi-1d	Simultaneous Tasks

Benchmarks are from the MiBench suite (GUTHAUS et al., 2001).

- **Multitasking.** This scenario contains more than 30 instances of MiBench benchmarks, with either integer and FP applications. The goal of this scenario is to pressure the host system regarding task parallelism. This can be the case in cloud-servers, for example. With this scenario, we may demonstrate the benefits of increasing the computing nodes of an MPSoC through partial-ISA.

- **Edge computing.** This scenario contains a variety of *on-the-edge* jobs (which process data from embedded systems before forwarding to data-centers, for example). It contains kernels for image processing, navigation, health monitoring, pattern matching, and data compression. To simulate the inputs and outputs send-receive of each kernel, we surround the kernels with encryption and decryption steps. While decryption-processing-encryption can be considered to execute in a pipeline manner, there can be several executions of similar decryption-processing-encryption occurring simultaneously. We explore this scenario to analyze the impact of adding in-order cores or AES hardware accelerators into partial-ISA MPSoCs. All benchmarks come from the IoT-LOCUS suite (TAN et al., 2017).

- **Road sign detection.** This scenario contains a CNN inference kernel, which detects and reads the data from road signs (as maximum speed). We consider the kernel to execute in a pipeline with aes and crc32 - simulating data acquisition and checksum. With this scenario, we expect to demonstrate the benefits of adding accelerators to the chip after adopting partial-ISA. The CNN kernel comes from the work by

(PEEMEN; MESMAN; CORPORAAL, 2011), while aes and crc32 come from the MiBench suite.

- **Video Streaming.** This scenario contains a h264 video decoding application, running aside with aes and crc32 benchmarks for data acquisition and checksum. Likewise, *Road sign detection*, this scenario can be enhanced through the adoption of hardware accelerators, enabled by our partial-ISA proposal. The video application comes from the work by (SUEHRING, 2010), while aes and crc32 come from MiBench.

- **FP-Driven.** This scenario contains FP-only benchmarks to stress the host system FPUs. Since our partial-ISA proposal removes FP support from some cores in the MPSoC, we use this scenario to assess the impact of such an approach when FP usage rises. The benchmarks are mainly mathematical and come from the polybench suite (POUCHET; YUKI, 2015).

5.1.2 Configurations

The adoption of partial-ISA cores in an MPSoC results in slack in area and power, which the designer can exploit to adopt extra computing nodes to the system. Following, we present data regarding the area and power savings for the considered GPPs, when we trim FP support to transform them into partial-ISA GPPs. Additionally, we present the data for hardware-accelerators. Using this data, we compose different *hw-configurations* targeting the scenarios aforementioned in Table 5.1. We demonstrate that partial-ISA configurations allow increased heterogeneity while respecting the area and power constraints of the Baseline.

Table 5.2 presents area and mean power for every computing node we consider in this work. As one can see, we present data for the GPP cores (see the configurations in Table 4.1) with and without FP support (Full or Partial, in the table). Also, we present data for three accelerators, already introduced in Section 4.4. The AES Avalon, the simplest node evaluated, is application-specific hardware that implements AES encryption and decryption; the Nova is a decoder for the h264 digital video standard; the Origami hardware is an accelerator for the convolutional layers of CNNs. We have chosen accelerators that are suitable for at least one of the *executing-scenarios* (Table 5.1). For example, the Origami accelerator can be assigned to execute the convolutional layers of the *Road*

Table 5.2: The area and power of the computing nodes considered in our work. Including Level 1 instruction and data caches.

	Area (mm^2)	Mean Power (W)
Big core - Full	0.343	0.767
Big core - Partial	0.218	0.358
Little core - Full	0.111	0.086
Little core - Partial	0.093	0.032
Avalon AES Encrypt/Decrypt Accelerator	0.007	0.004
Nova H264 Decoder	0.071	0.029
Origami Convolutional Accelerator	0.355	0.124

sign detection scenario, the Nova can be employed in the *Video Streaming* scenario, and the Avalon can handle aes kernels (which appear in more than one scenario).

We highlight a few observations from Table 5.2. First, trimming the ISA support for FP instructions in the big cores has more impact than in little cores. The absolute area saved with partial big cores is around 7× the absolute area saved with partial little cores. Power savings follow a similar trend. This occurs because the FPU in wide OoO are more complex (deeper pipeline and larger FP register file, for example), and also must cope with the out-of-order execution, interacting with complex modules (e.g., the Re-order Buffer (ROB)). Also, our big core contains *two* FPU (like in the ARM Cortex A15, which we model). Second, removing the FP support from an OoO core create sufficient slack in area and power to add an extra in-order core, or accelerators (except Origami). Third, it is possible to add accelerators (Avalon AES) even when we trim the ISA for little cores.

To apply our partial-ISA methodology, **we consider a Baseline hw-configuration composed of a 4 big (OoO) core + 4 little (in-order) system.** This Baseline is based on ARM's heterogeneous setup with the DynamIQ technology (ARM, 2017). DynamIQ extends the former big.LITTLE technology by increasing the number of cores, and relaxing the ratio between big and little cores (in big.LITTLE, big and little cores always come in pairs).

We use the data of each individual hardware (cores or accelerators), presented in Table 5.2, to get the total area and power of the Baseline. We then consider the Baseline's area and power as the limit for any other MPSoC hw-configuration. Upon this, we simplify some cores from the Baseline (turning them partial-ISA cores) and leverage the savings in area and power to compose different hw-configurations, always under the baseline constraints. Table 5.3 presents the different hw-configurations we built with this

Table 5.3: The configurations evaluated in this work.

Configuration	Description	Target scenario
Baseline	4 OoO 4 In-Order	FP-Driven
Task Parallel	2 OoO 2 OoO (Partial-ISA) 6 In-Order	Multitasking Smartphone App Edge Computing
AES Accelerated	4 OoO 2 In-Order 2 In-Order (Partial-ISA) 4 AES Accelerator (Avalon)	Edge Computing Video Streaming Road Sign Detection
Video Accelerated	2 OoO 2 OoO (Partial-ISA) 5 In-Order 1 H264 Accelerator (Nova)	Video Streaming
CNN Accelerated	1 OoO 3 OoO (Partial-ISA) 4 In-Order 1 CNN Accelerator (Origami)	Road Sign Detection
Accelerator Rich	4 OoO (Partial-ISA) 4 In-Order 4 AES Accelerator (Avalon) 1 H264 Accelerator (Nova) 1 CNN Accelerator (Origami)	Edge Computing Video Streaming Road Sign Detection

Table 5.4: The area and power of evaluated configurations.

Configuration	Total Area (mm^2)	Total Mean Power (W)
Baseline	1.81	3.41
Task Parallel	1.79	2.76
AES Accelerated	1.80	3.32
Video Accelerated	1.75	2.71
CNN Accelerated	1.79	2.31
Accelerator Rich	1.77	1.94

methodology, their computing nodes, and the scenarios they target to improve. At the same time, Table 5.4 presents the total area and power of each of these hw-configurations, and demonstrate that we never surpass the area and power budget from the former Baseline.

The proposed *hw-configurations* intend to demonstrate how partial-ISA MPSoCs can be composed to tackle different *executing-scenarios*. As said, in Table 5.3 we highlight target scenarios of each hw-configuration, which depend on the set of computing nodes that compose a hw-configuration (which appears in the *Description* column). For example, the *Task Parallel* has an increased number of cores, which can be helpful when the number of tasks rises (scenarios such as *Multitasking*, *Smartphone App*, and *Edge*

Computing). The AES Accelerated hw-configuration contains accelerators for cryptography, which can be valuable when cryptography is required (scenarios such as *Edge Computing*, *Video Streaming*, and *Road Sign Detection*). Likewise, *Video Accelerated* and *CNN Accelerated* hw-configurations contain application-specific hardware that can execute the respective kernels with lower energy and higher performance. The *Accelerator Rich* hw-configuration contains every evaluated accelerator, but can suffer under *FP-Driven* because it removes the FPU from all OoO cores. The *Baseline*, in turn, has full cores only and should not struggle with FP loads.

5.2 Scheduling analysis

After defining the *executing-scenarios* and the *hw-configurations* we now turn our focus to the analysis of the proposed technique. We start observing how the *Scheduler* can leverage the configurations in both *performance-oriented* (where big OoO cores are preferred) and *energy-oriented* (where little in-order cores are preferred). As we stated previously (see Figure 5.1), we execute every *scenario* under every *hw-configuration*, in both scheduling policies, which sums up to a combination of $6 scenarios \times 6 configurations \times 2\ policies = 72\ executions$. To simplify the explanation, we elaborate the scheduling analysis discussion over a selection of insightful situations to clarify the partial-ISA functioning, benefits, and threats. The analysis presented here supports later discussion on overall results for performance and energy. Following, we observe how each *hw-configuration* behaves in at least one *executing-scenario*, comparing to the Baseline design.

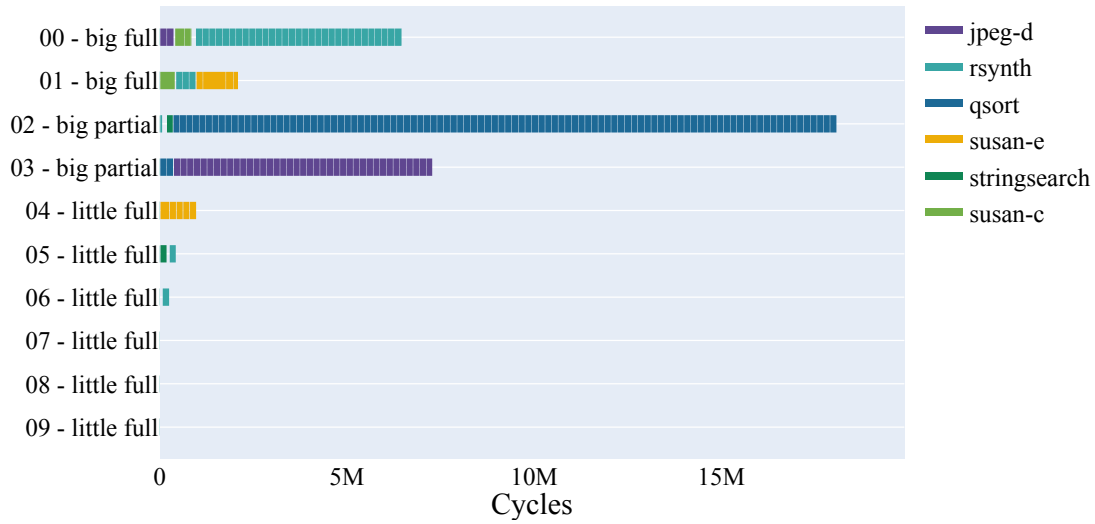
5.2.1 Task Parallel Hw-Configuration

Figure 5.2 presents the performance-oriented task scheduling for the *Smartphone App* scenario on two designs. It compares the execution under the *Task Parallel* hw-configuration (a), and *Baseline* hw-configuration (b). The chart depicts the applications (listed in the legend) scheduling in each host core in a timeline manner. The x-axis presents the execution cycles, using the frequency of the big cores as the step (since it has the higher frequency and hence finer grain) and normalizing the cycle duration of the other nodes to the big core cycle, so we have a common step in the x-axis. At the same

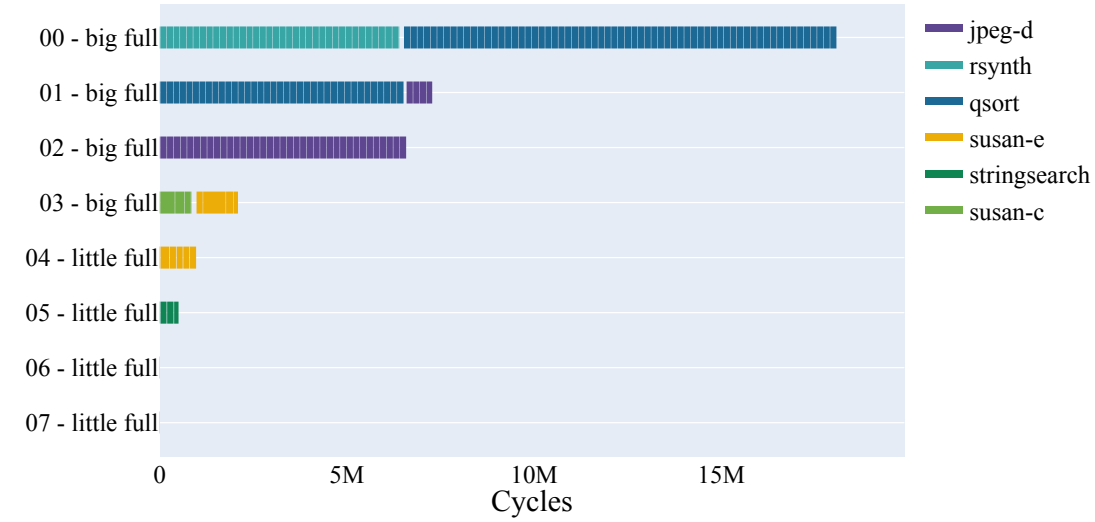
time, the y-axis presents the computing nodes of the *hw-configuration*, accordingly with their *description* in Table 5.3. For example, in Figure 5.2 the *Task Parallel* (a) has 10 cores, where cores 2 and 3 are partial. We highlight that this chart pattern holds for the subsequent figures in this section.

Figure 5.2: Execution of the *Smartphone App* scenario in the *performance-oriented* policy.

(a) Task Parallel Hw-Configuration



(b) Baseline Hw-Configuration



Source: The author.

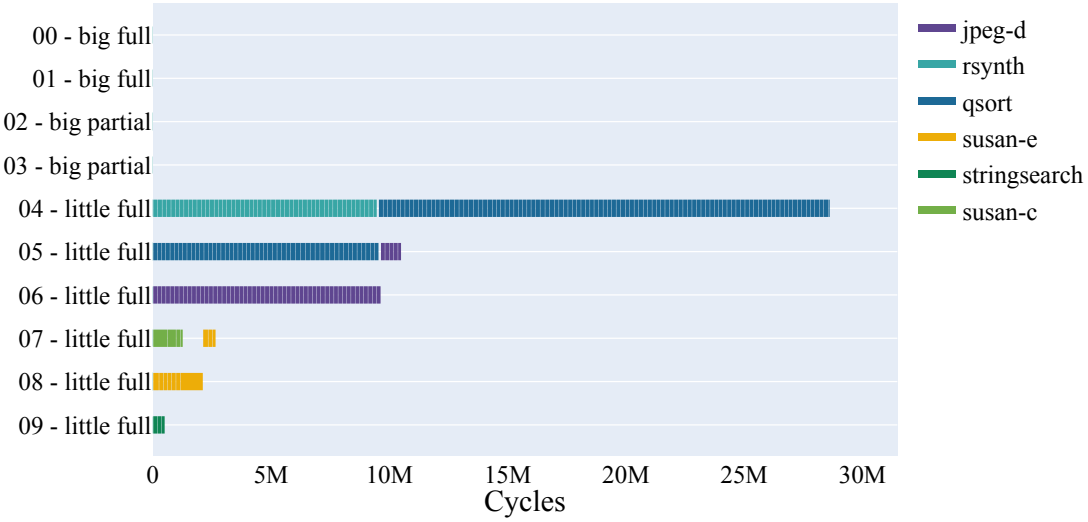
Although the extra cores are not used in this scenario (because there are fewer applications than cores), big partial-ISA cores are preferred over other GPP cores, as expected for the performance-oriented scheduling policy (recall Algorithm 1). With this, the integer apps *qsort* and *jpeg-d* applications can execute with reduced power (partial-ISA

cores are simpler), while keeping the performance (still running under an OoO microarchitecture). For the remaining applications, they migrate to the cores accordingly to their needs (e.g., FP support necessary for *susan-c*) and respecting the scheduling policy (preferring OoO over in-order cores).

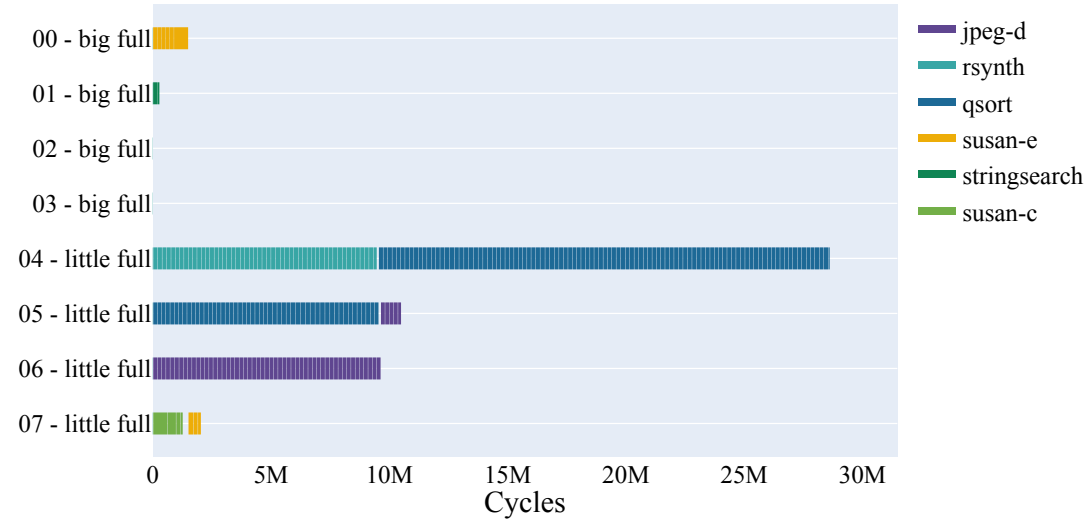
In comparison, Figure 5.3 presents the same scenario and hw-configurations, but in the *energy-oriented* scheduling policy. In this case, the execution flow in the *Task Parallel* (Figure 5.3-a) and *Baseline* (Figure 5.3-b) is very similar, since we do not have partial in-order cores in both hw-configurations. The difference, however, is that the increased number of in-order cores in the *Task Parallel* hw-configuration avoids the use of big cores. This saves energy while it does not introduce any overall latency penalty, hence being as performing as the *Baseline*.

Figure 5.3: Execution of the *Smartphone App* scenario in the *energy-oriented* policy.

(a) Task Parallel Hw-Configuration



(b) Baseline Hw-Configuration

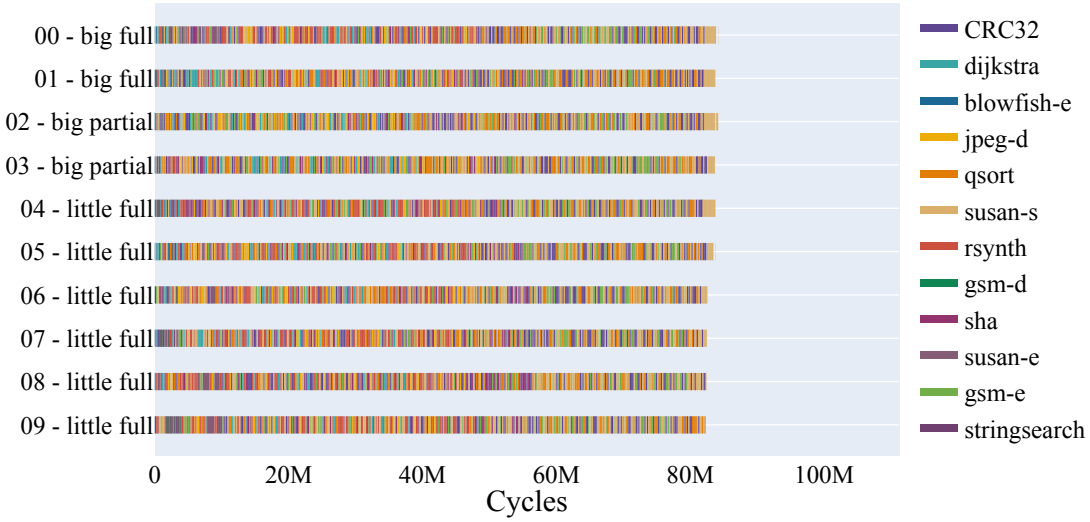


Source: The author.

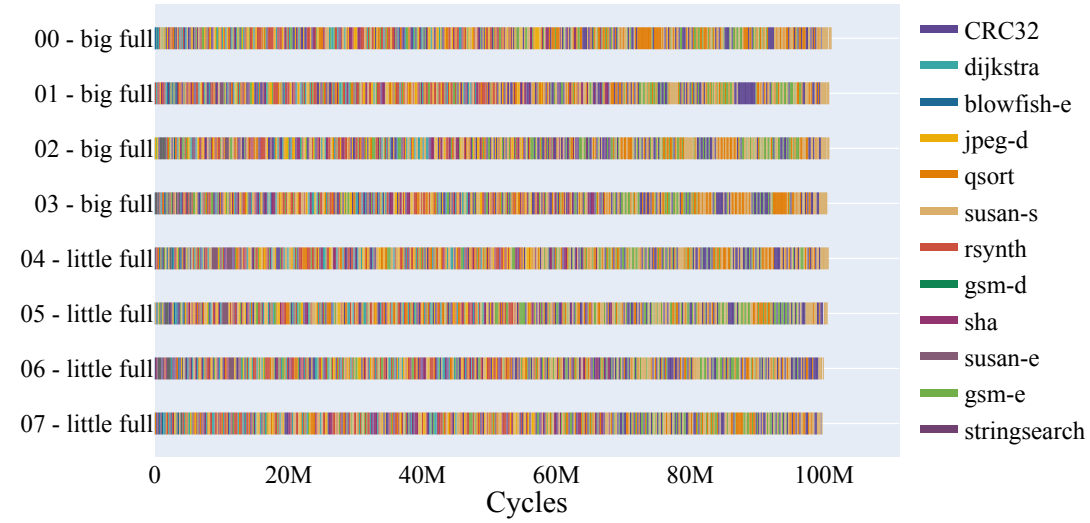
We continue the analysis of the *Task Parallel* hw-configuration, but now presenting the scheduling flow for the *Multitask* executing-scenario. Figure 5.4 depicts the comparison of this design with the *Baseline*, in the *performance-oriented* scheduling policy. This scenario contains more than 30 instances of MiBench applications (we execute three instances of each of the benchmarks in the legend).

Figure 5.4: Execution of the *Multitask* scenario in the *performance-oriented* policy.

(a) Task Parallel Hw-Configuration



(b) Baseline Hw-Configuration



Source: The author.

Since the *Task Parallel* hw-configuration has two extra full-ISA little cores, this design has higher throughput compared to the Baseline. Notably, this setup is achieved while respecting the Baseline's former area and power budget (Table 5.4). As one can see, the set of applications finish their execution in about 80% of the *Baseline's* cycle count. Thus, improving performance. Additionally, since the *Task Parallel* hw-configuration executes fewer applications in the full-ISA OoO cores, by taking advantage of the simpler partial-ISA OoO cores and additional little cores, energy reductions will also appear.

Results for the *energy-oriented* scheduling policy are similar to the *performance-oriented* scheduling policy in this case, and we suppress the corresponding chart for

brevity. This similarity occurs because, as detailed in Section 3.2, the *Scheduler* verifies the available cores to decide which of them to assign a workload. However, in this scenario, cores are occupied most of the time, and the *Scheduler* essentially assigns workloads to the first core that preempts a task and becomes available.

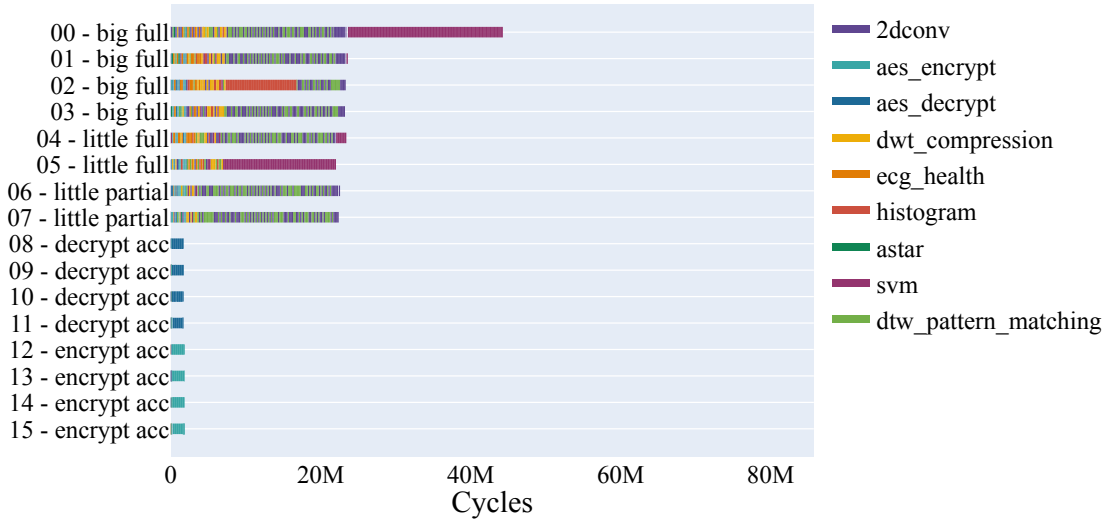
5.2.2 AES Accelerated Hw-Configuration

To analyze the task scheduling on the AES Accelerated hw-configuration, Figure 5.5 presents the *Edge Computing* executing-scenario performing under the *performance-oriented* scheduling policy on both *AES Accelerated* (a) and *Baseline* (b) hw-configurations. Notably, encryption and decryption kernels heavily occupy GPP cores in the Baseline hw-configuration (Figure 5.5-b). This occurs because on-the-edge tasks have a high load of encrypted communication since they are used to exchange (and prepare) data generated by leaf nodes (e.g., sensors) to the cloud.

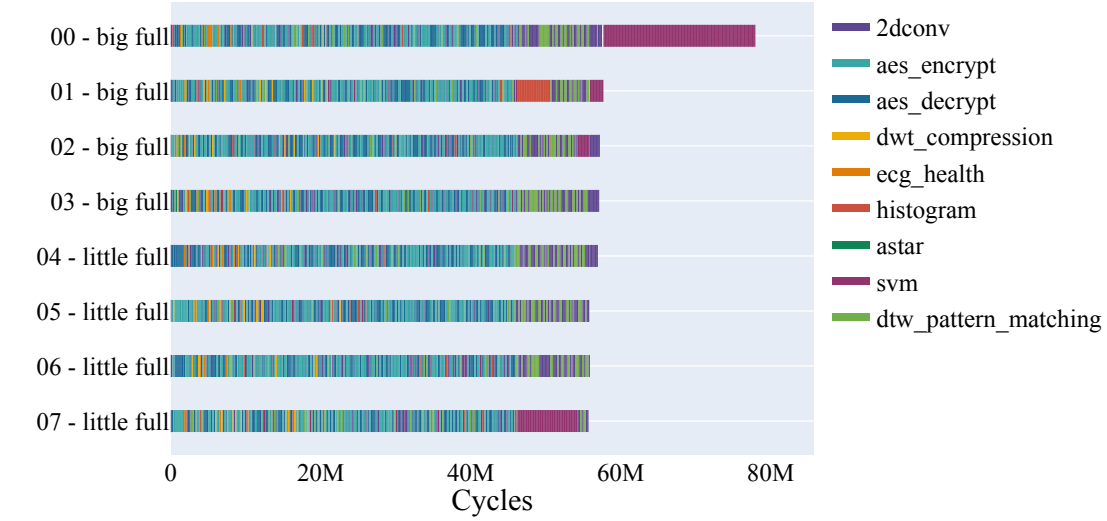
As defined in Table 5.3, the *AES Accelerated* hw-configuration has *four* AES Avalon accelerators, which are possible since we transform two in-order cores from the *Baseline* into partial-ISA cores to define this design while respecting former constraints. These new nodes appears in Figure 5.5-a. Also, these accelerators are capable to decrypt and encrypt in the AES standard in a much quicker (see Section 4.4) and power-efficiently (see Table 5.2) way compared to GPP cores. Hereby, we can offload the cryptography tasks to the accelerators, while exploiting the GPP cores to execute the remaining general-purpose tasks.

Figure 5.5: Execution of the *Edge Computing* scenario in the *performance-oriented* policy.

(a) AES Accelerated Hw-Configuration



(b) Baseline Hw-Configuration

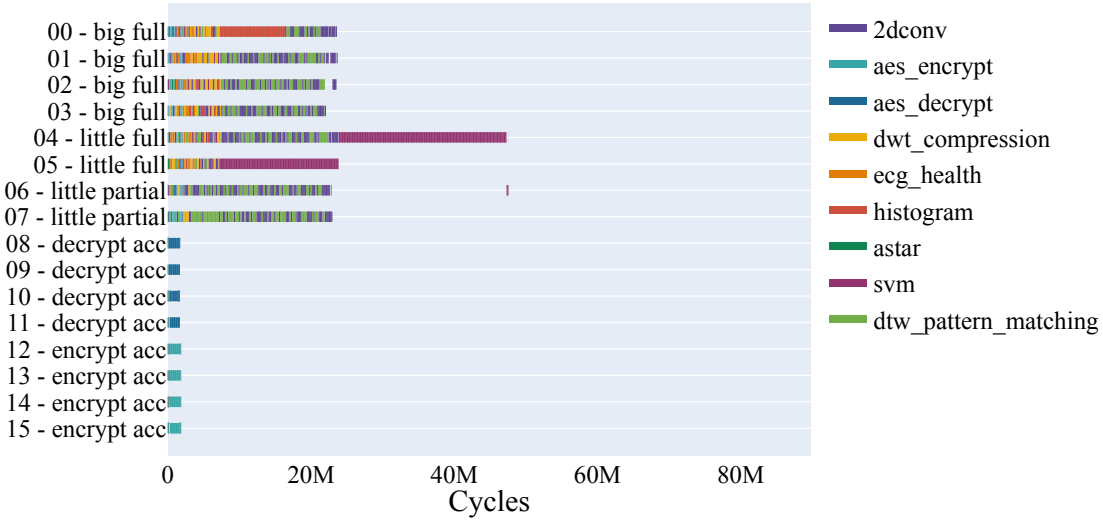


Source: The author.

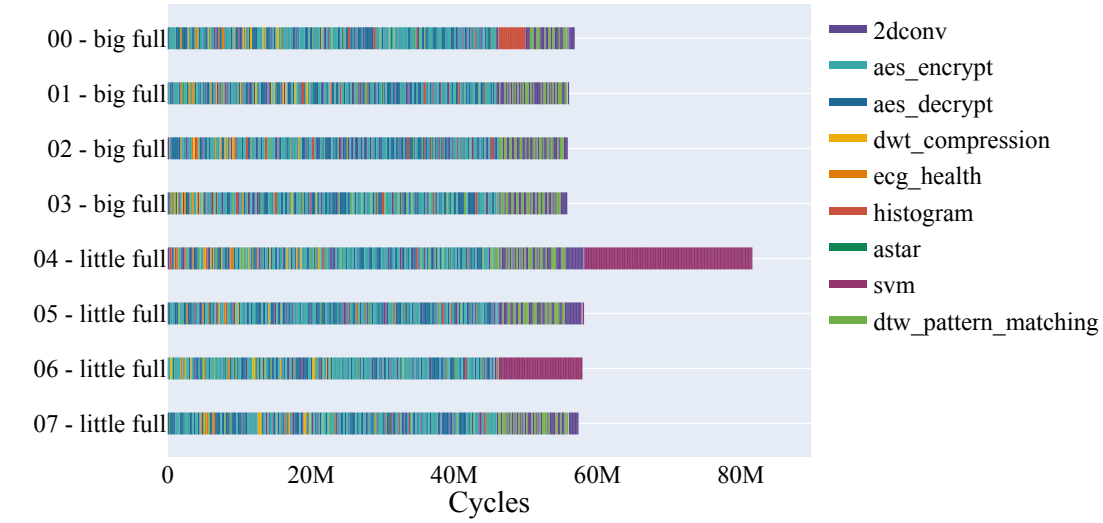
This behavior also occurs in the *energy-oriented* scheduling policy, depicted in figure 5.6. Because accelerators are preferred over GPP cores regardless of the scheduling policy, they are used similarly. The mapping strategies only decide where the remaining tasks should be executed when there is more than one core available (preferring little cores).

Figure 5.6: Execution of the *Edge Computing* scenario in the *energy-oriented* policy.

(a) AES Accelerated Hw-Configuration



(b) Baseline Hw-Configuration



Source: The author.

5.2.3 Video Accelerated Hw-configuration

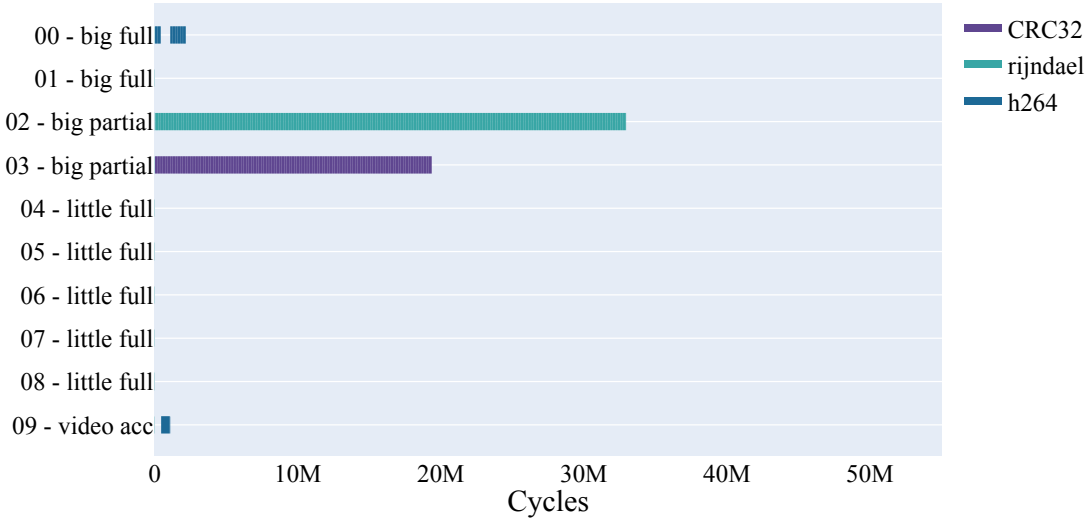
The *Video Accelerated* hw-configuration is compared with the *Baseline* in the performance-oriented scheduling policy (Figure 5.7) and energy-oriented scheduling policy (Figure 5.8). In the *Video Accelerated* hw-configuration, we recall, there is a h264 ASIC. Under both scheduling policies, the migration to the video accelerator provides considerable performance gains. The total time, however, is limited by the *rijndael* benchmark (from MiBench), which performs AES decryption (in a GPP in this hw-

configuration, since it does not have the Avalon accelerator).

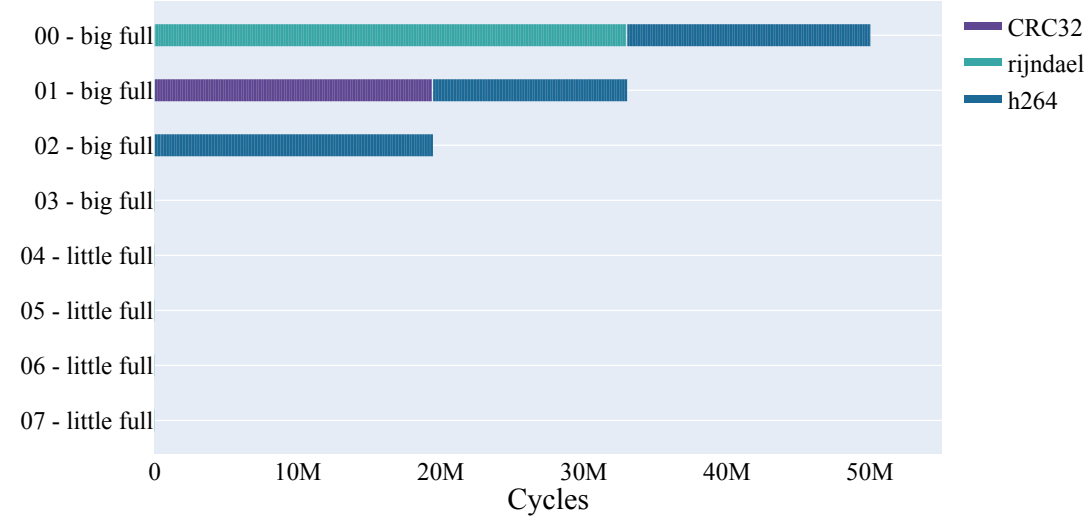
As depicted in the chart, there are parts of the h264 application that cannot execute in the accelerator. This is mainly due to Input/Output (IO) management, such as data preparation (loading the stream to the main memory, for example), and output the results at the end of the execution. For the other applications, there is no overhead in migration since they are integer only. This allows partial-ISA cores to be used with no performance penalty.

Figure 5.7: Execution of the *Video Streaming* scenario in the *performance-oriented* policy.

(a) Video Accelerated Hw-Configuration



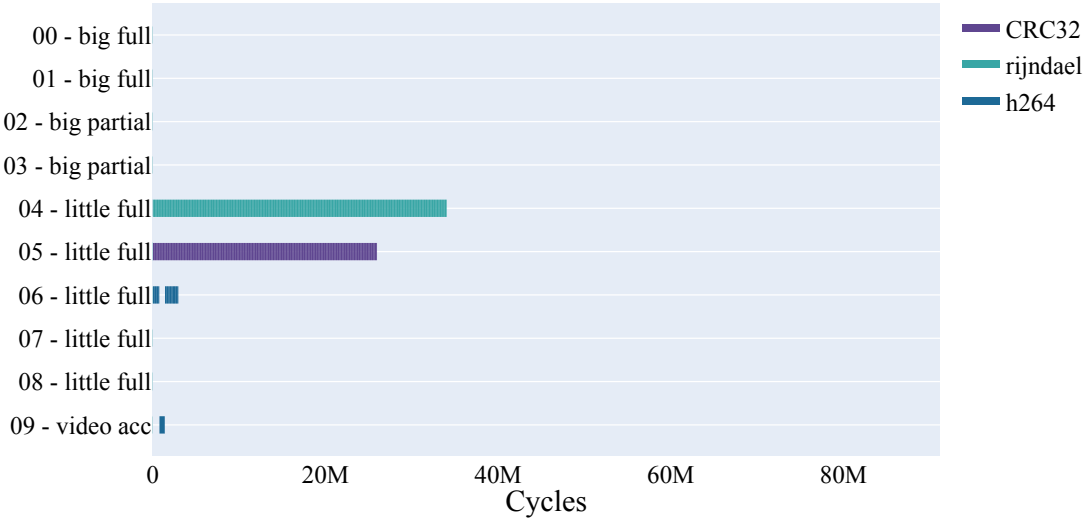
(b) Baseline Hw-Configuration



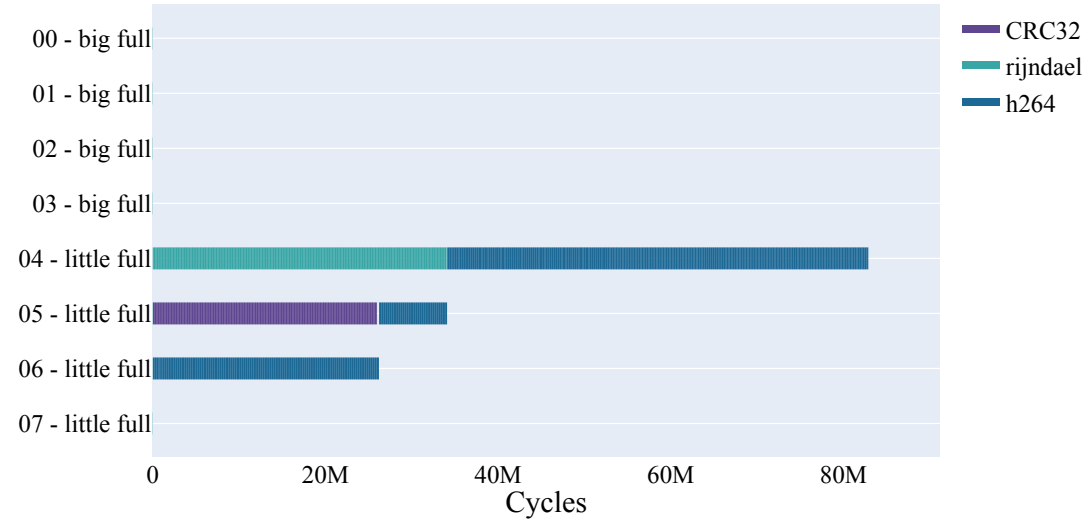
Source: The author.

Figure 5.8: Execution of the *Video Streaming* scenario in the *energy-oriented* policy.

(a) Video Accelerated Hw-Configuration



(b) Baseline Hw-Configuration



Source: The author.

5.2.4 CNN Accelerated Hw-Configuration

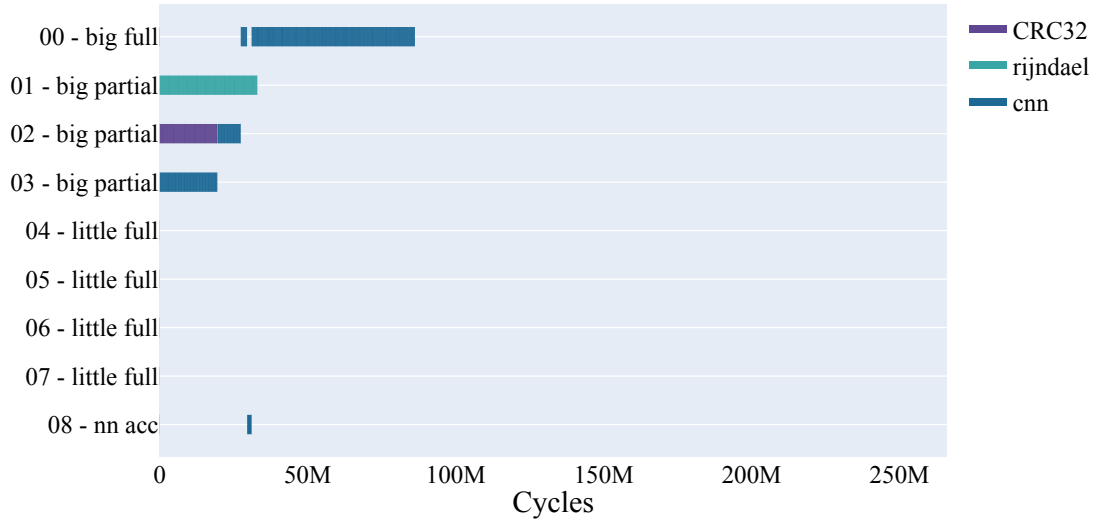
Figure 5.9 presents the *Road Sign Detection* executing-scenario executing in the performance-oriented scheduling policy for both the *CNN Accelerated* hw-configuration (which contains a CNN accelerator) and the *Baseline*. Similarly to the *Video Accelerated* hw-configuration, the accelerator is leveraged to execute the accelerated phase of the application - in this case a CNN inference for road sign detection.

During the neural network kernel initialization, where there is weights and inputs

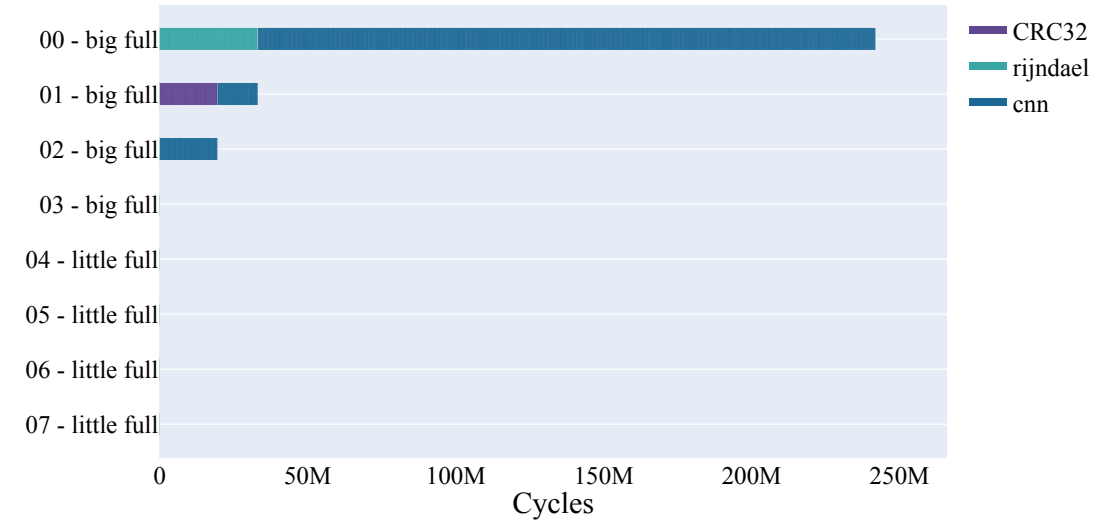
reading, the workload runs in a GPP. However, since this initial portion of code does not execute FP instructions, it can be executed in the partial cores, which are indeed explored by the scheduler as the figure depicts.

Figure 5.9: Execution of the *Road Sign Detection* scenario in the *performance-oriented* policy.

(a) CNN Accelerated Hw-Configuration



(b) Baseline Hw-Configuration



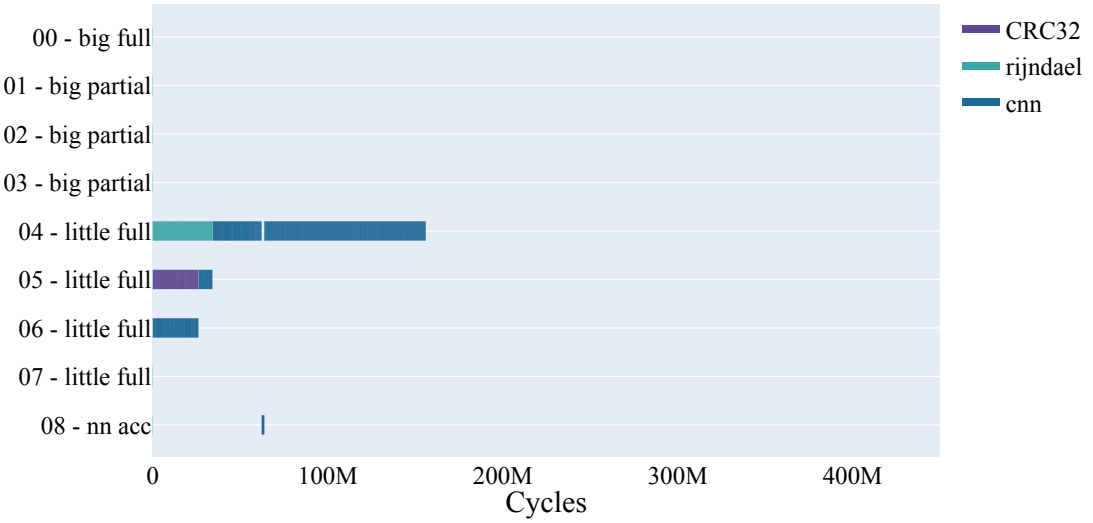
Source: The author.

After the convolutional kernel, however, the application proceeds with fully connected layers calculation (which cannot be accelerated with the convolutional accelerator, since their topology is different). These fully connected layers are used to trigger the decision upon a road sign detection and its value (e.g., there is a plate in the y,x position depicting 80mp/h). Because these layers contain FP weights, full-ISA GPP cores are

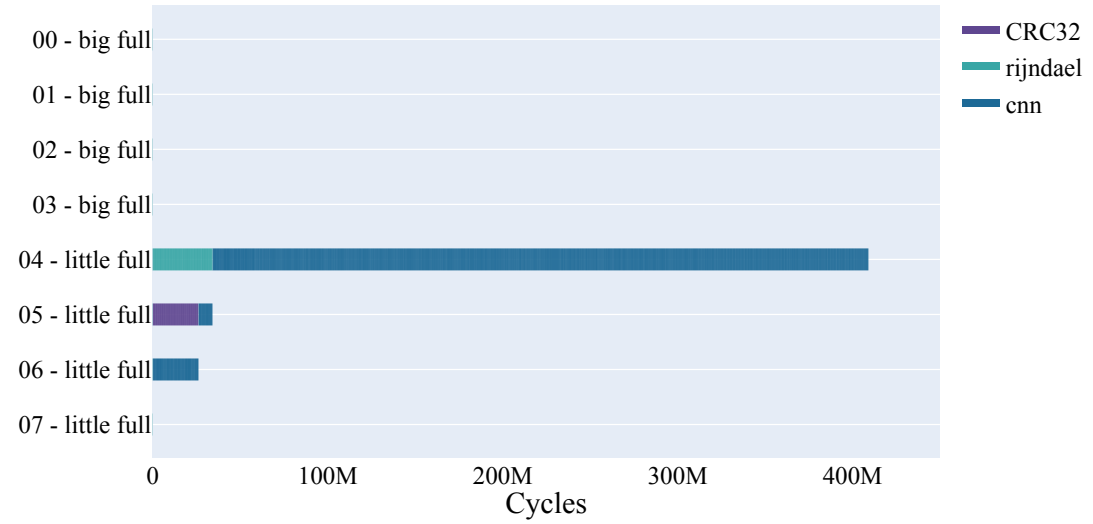
necessary. A similar trend occurs in the energy-oriented performance (Figure 5.10), except the scheduler prefers little cores, and there is no partial little core that the scheduler can leverage (neither for the CNN initialization nor for the remaining workloads in the executing-scenario).

Figure 5.10: Execution of the *Road Sign Detection* scenario in the *energy-oriented* policy.

(a) CNN Accelerated Hw-Configuration



(b) Baseline Hw-Configuration



Source: The author.

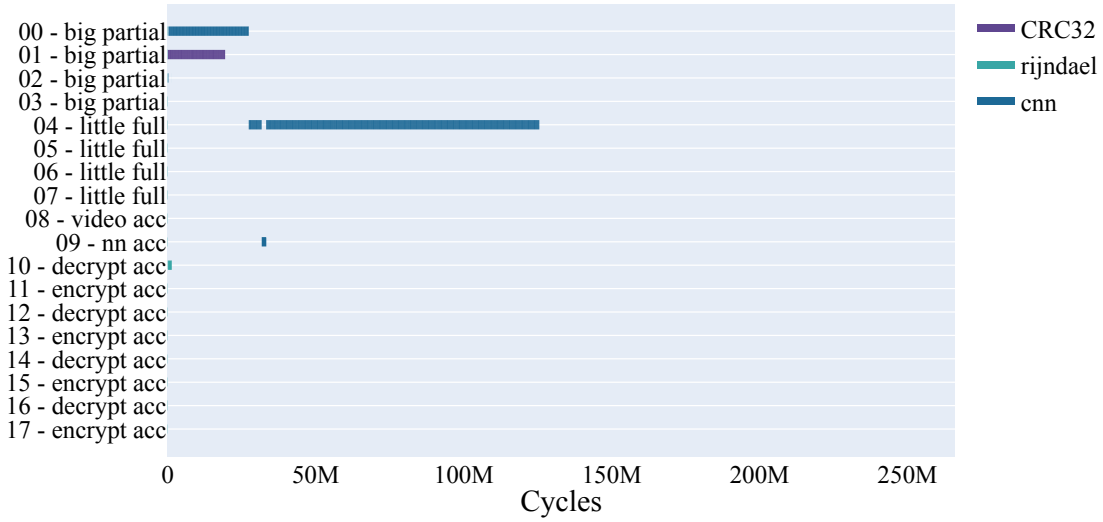
5.2.5 Accelerator Rich Hw-Configuration

For the *Accelerator Rich* hw-configuration, we analyze two executing-scenarios in the performance-oriented scheduling policy. First, we present the scheduling for the

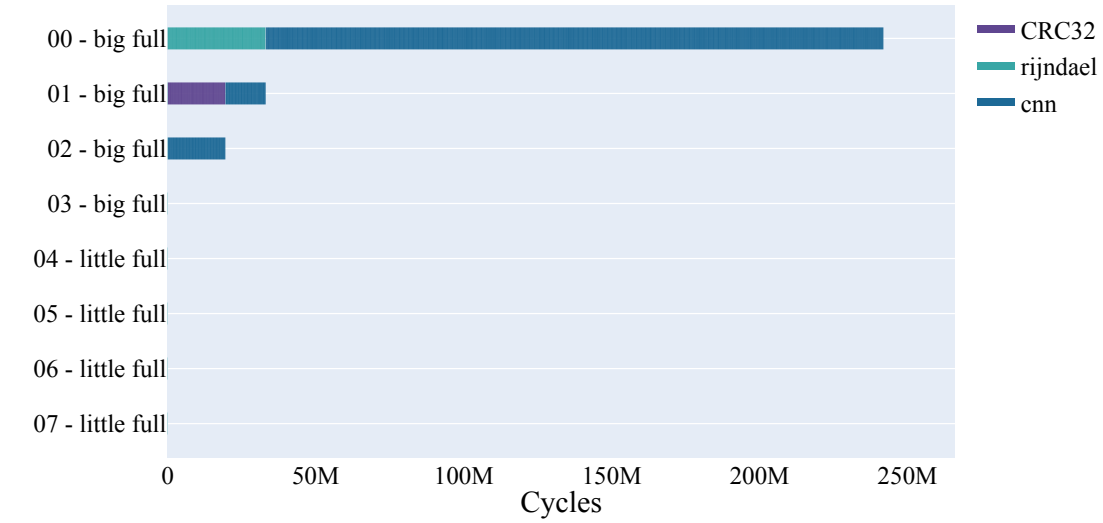
Road Sign Detection scenario again, in Figure 5.11, but this time on the *Accelerator Rich* hw-configuration, which contains more than one accelerator.

Figure 5.11: Execution of the *Road Sign Detection* scenario in the *performance-oriented* policy.

(a) Accelerated Rich Hw-Configuration



(b) Baseline Hw-Configuration



Source: The author.

In this situation, both kernels (*cnn* and *rijndael*) execute in their respective accelerators (Origami-CNN and Anova-AES). In the *Accelerated Rich* hw-configuration (Figure 5.11-b) the *rijndael* execution is almost negligible, which reliefs energy consumption of this hw-configuration by avoiding the use of GPP cores, which cannot be avoided in the Baseline (Figure 5.11-b). At the same time, the CNN accelerator is used. The usage of this ASIC combined leads to significant performance speedup, as also lower energy con-

sumption. In fact, performance gain is only limited because of the GPP parts of the *cnn* application, which is necessary for some parts of the road sign detection workload as we discussed in subsection 5.2.4. Thus, this hw-configuration shows how partial-ISA can be used to increase heterogeneity in an MPSoC, with extra and different accelerators.

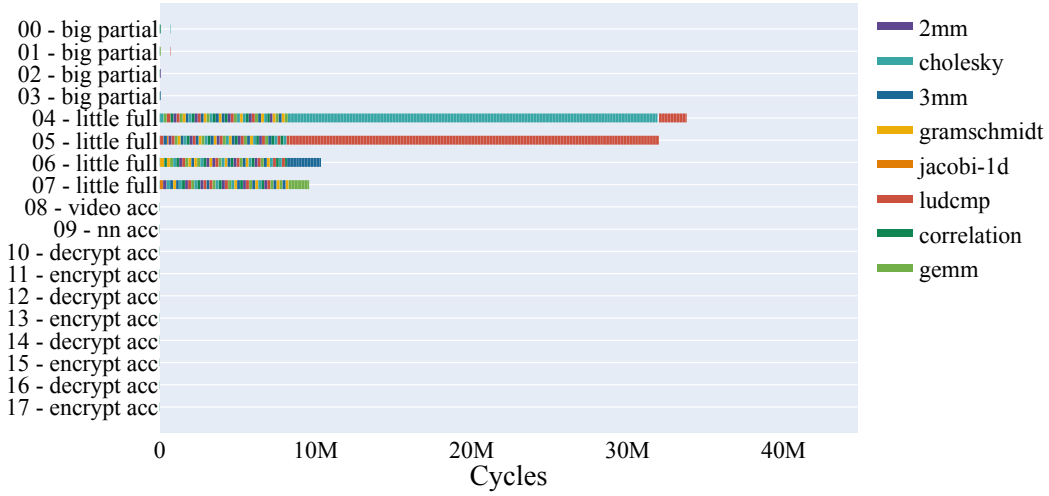
Although the *Accelerated Rich* hw-configuration will be suitable for different accelerated scenarios, as we will see in the next section (5.3), it may struggle to cope with high loads of FP applications. This is because we had to remove the FP of four OoO cores to embrace the accelerators, and still respect the former *Baseline's* area and power budget.

To illustrate this, Figure 5.12 depicts the *Accelerator Rich* hw-configuration executing the *FP-Driven* scenario, in (a). This scenario was designed to stress the FPU of the cores. Because all big cores are partial, they rarely can be used to execute the mathematical applications of the scenario. The Baseline hw-configuration (c), otherwise, can use all of its eight full-ISA cores (particularly it uses the full-ISA OoO cores to execute the most prolonged workloads faster), hence delivering better results.

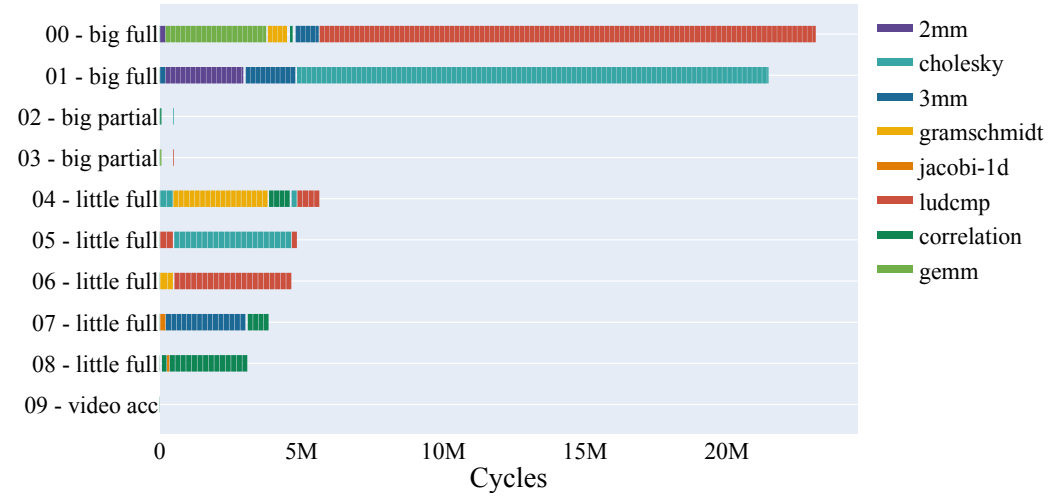
For comparison, we also depict how the *Video Accelerated* hw-configuration executes the same scenario, to demonstrate how the selection of the number of partial-ISA cores can be explored by the designer to adjust the generality of the design. The *Video Accelerated* hw-configuration appears in Figure 5.12-b (with its two partial-ISA cores, instead of four as the *Accelerated Rich configuration*). Although *two* big cores are turned into partial, and the video accelerator cannot be used for the set of benchmarks, there is an extra full-ISA little core in the *Video Accelerated* hw-configuration, which helps the task mapping. Moreover, there still are two big cores with full-ISA support, amortizing the performance gap between the *Accelerated Rich* hw-configuration and the *Baseline* for the *FP-Driven* scenario.

Figure 5.12: Execution of the *FP-Driven* scenario in the *performance-oriented* policy.

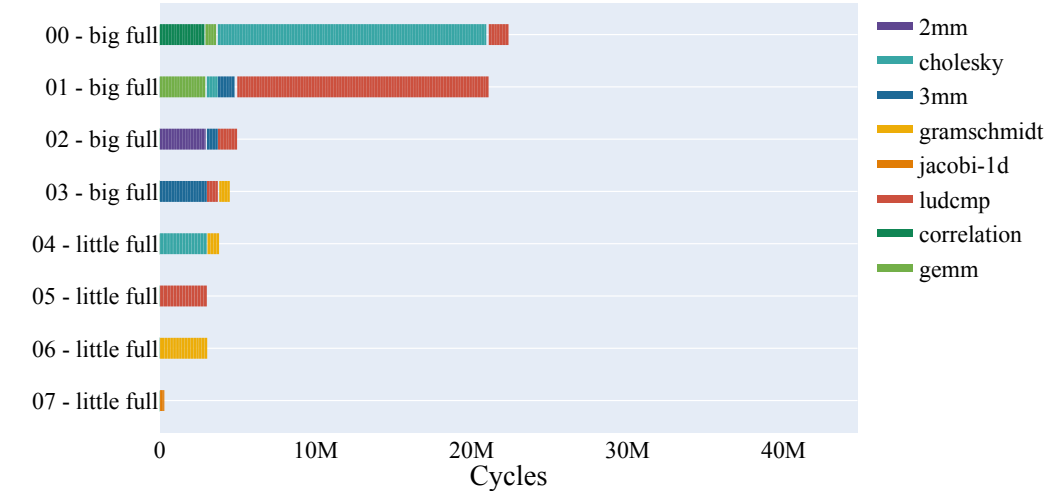
(a) Accelerated Rich Hw-Configuration



(b) Video Accelerated Hw-Configuration



(c) Baseline Hw-Configuration



Source: The author.

5.3 Performance and energy analysis

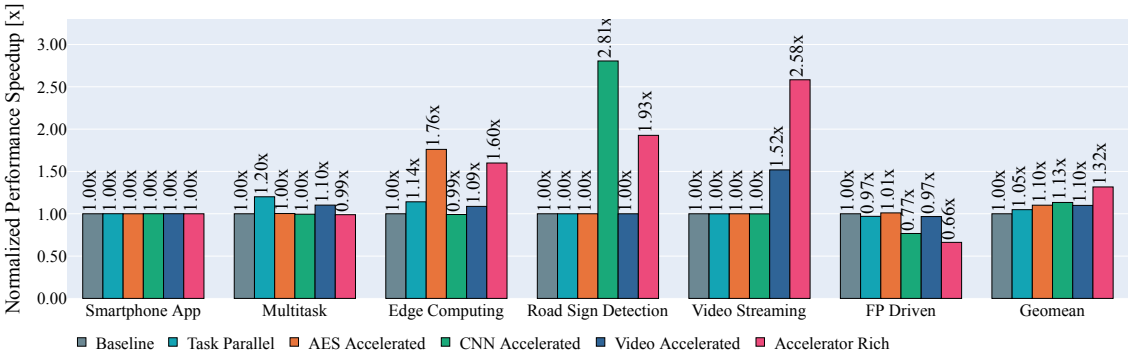
After analyzing a variety of executions and how workloads are mapped depending on the hw-configuration, we now discuss results for performance and energy regarding every *hw-configuration* under every *scenario* and scheduling policy. Figure 5.13 presents performance (a) and energy (b) under the *performance-oriented* scheduling policy. Similarly, Figure 5.14 depicts results for the *energy-oriented* scheduling policy.

The performance speedup is measured by comparing the time taken for all workloads to finish in the scenario. Take for example Figure 5.12 from the scheduling analysis in the previous section. In (a), the *ludcmp* is the last one to finish, ending the execution of the scenario after around 35M cycles (recall x-axis depicts the cycles w.r.t. the big core cycle, regardless the computing node). In (b), however, the *ludcmp* is again the last application to finish, but after around 25M cycles. This difference dictates performance speedup.

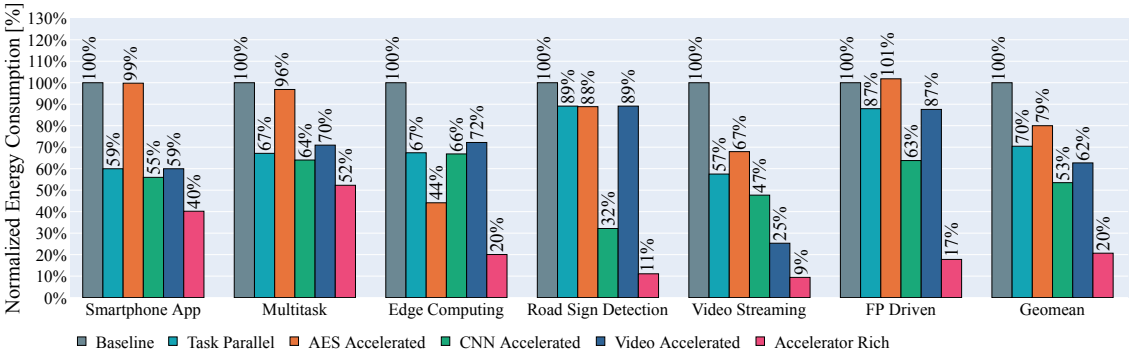
Energy comparison is based on the cumulative usage of the available nodes during a scenario execution, considering idle cores as power-gated. As we detail next, the best results for our partial-ISA *hw-configurations* are when executing latency-critical applications (in the hw-configurations with accelerators) and multi-tasking (in the hw-configuration with more in-order cores), while getting closer to the baseline results when FP demand grows (*FP-driven scenario*). We separate the analysis by *hw-configurations*, discussing the executing-scenarios in each of them. In the figures, the *hw-configurations* appear in the legend, having the same color throughout the charts.

Figure 5.13: All hw-configurations executing under a performance-oriented policy.

(a) Normalized performance speedup for each configuration and scenario. Higher is better.



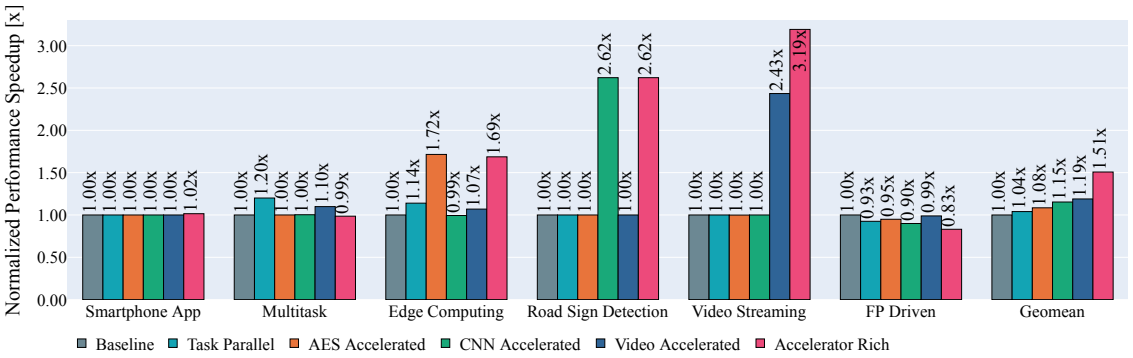
(b) Normalized energy consumption for each configuration and scenario. Lower is better.



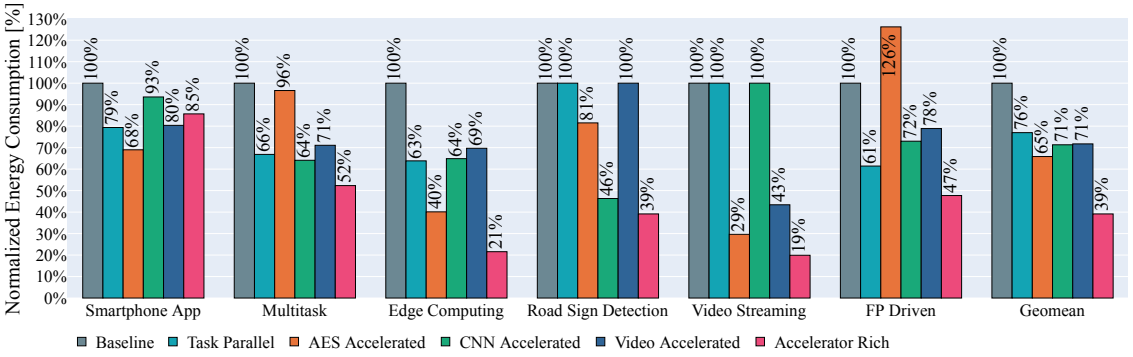
Source: The author.

Figure 5.14: All hw-configurations executing under an energy-oriented policy.

(a) Normalized performance speedup for each configuration and scenario. Higher is better.



(b) Normalized energy consumption for each configuration and scenario. Lower is better.



Source: The author.

5.3.1 Task Parallel Hw-Configuration

In this *hw-configuration* we remove the FP support of two OoO cores from the baseline, to add two full-ISA in-order cores. With this, it is possible to increase performance by up to 1.2 $\times$ in the *Multitasking* scenario and 1.14 $\times$ in the *Edge Computing* scenario, both in the performance-oriented, and in the energy-oriented scheduling policy. Because of the high core occupation in these scenarios (see Figure 5.4 and 5.5 for example), the impact of the scheduler is reduced. The scheduler cannot apply the policy decision of the preferred core (in-order or OoO) because it rarely finds more than one core available. Instead, it dispatches queued workloads as soon as a core is ready. This leads to similar performance results among policies. For the remaining scenarios, the trimmed FPUs impacts minimally or none in the performance because the remaining full-ISA cores are assigned by the scheduler to execute FP instructions. In the *FP-driven* scenario, however, there is a small reduction in performance because the simpler in-order cores frequently handle FP operations that OoO cores do in the *baseline*. In this case, there is a 3% slowdown.

Nevertheless, the benefits of energy consumption are evident in every scenario. This comes from the reduced peak-power of partial-ISA OoO cores and extra efficient in-order cores, increasing overall throughput to anticipate power-gating of cores as applications finish. Remarkably, *Multitasking* reduces up to 43% of the energy consumption in the *performance-oriented* scheduling policy (*Video streaming*), and 39% in the *energy-oriented* scheduling policy (*FP-driven*), reducing by 29% and 21% in the geomean, respectively. Since the Baseline executing in the energy-oriented scheduling policy (in-order cores preferred) is more efficient than the Baseline executing in the performance-oriented scheduling policy (OoO cores preferred), it is more challenging to improve energy consumption in the former.

5.3.2 AES Accelerated Hw-Configuration

In this *hw-configuration* we trim the FP support from two little cores, to adopt four AES accelerators (encrypt + decrypt). For the performance, there are gains for the *Edge Computing* scenario, improving the performance-oriented by 1.76$\times$, and the energy-oriented by 1.72$\times$ (see Figures 5.5 and 5.6). The performance loss from the reduced number of full little cores only exists in the *FP-Driven* scenario in the energy-oriented

scheduling policy, where the FP usage is higher. For the performance-oriented scheduling policy, there is a small performance gain, since FP applications are forced to execute in the faster OoO cores (this increases energy, however, Figure 5.12a), since there are fewer full little cores. For the remaining scenarios, the migrations to handle *FP* degrades performance below the third decimal place.

The *AES Accelerated* hw-configuration reduces energy consumption mainly because of the accelerator adoption. The accelerated scenarios, *Edge Computing*, *Road Sign Detection*, and *Video Streaming* have significant reduction (up to 71% in the *Video Streaming* under the energy-oriented scheduling policy, Figure 5.13b). When the accelerator cannot be leveraged, the partial little cores also assure energy savings, but smaller. For the *FP-Driven* scenario, the need for FP support mounts the usage of the costly OoO cores, which increase the energy in the energy-oriented scheduling policy by 26%. In the overall, the *AES Accelerated* reduces the geomean energy consumption by 21% and 35% for the performance-oriented and energy-oriented scheduling policy, respectively.

5.3.3 CNN Accelerated Hw-Configuration

In this *hw-configuration* we add a convolutional CNN hardware accelerator, which requires trimming the ISA of three OoO cores (so we respect the Baseline's constraints, recall Tables 5.3 and 5.4). However, the results in performance are very positive for the target scenario. The accelerator improves performance for the (*Road sign detection*) scenario by 2.81× and 2.62× for the performance and energy-oriented policies, respectively.

As we demonstrated in the scheduling analysis (Figures 5.9 and 5.10) even after the accelerator adoption the *cnn* benchmark remains as the longest-latency workload. Hence, overall speedup depends on the share of the program where the accelerator applies, which also depends on the GPP that executes the *cnn* benchmark when the accelerator cannot be used (calculation outside the convolutional layers scope). This causes a slightly different speedup depending on the scheduling policy. Moreover, because there are no additional in-order cores in this *hw-configuration*, there is no improvement in performance for other scenarios. In fact there is overhead in the *FP-driven* (23% slowdown).

However, the three partial-ISA cores gracefully reduce energy consumption. This is clearly noted in the performance-oriented scheduling policy, where the full-big cores are preferred by the scheduler. When FP instructions are not executed, the big partial cores can execute the applications normally. As a result, it consumes about half the geomean

energy of the *baseline* in the performance-oriented scheduling policy. In the energy-oriented scheduling policy, where the big cores usage is smaller, the gains of partial big cores are also smaller, but still considerable. It reduces energy consumption by 29% compared to the Baseline executing in this scheduling policy.

5.3.4 Video Accelerated Hw-Configuration

In this *hw-configuration*, we make room for an h264 video-decoder hardware and one additional in-order core, respecting the area constraints of the Baseline. With the accelerator, this *hw-configuration* speedups up 1.52× the *Video Streaming* scenario in the performance-oriented scheduling policy, and 2.43× in the energy-oriented scheduling policy. The speedup is limited by the simultaneously executed *aes* kernel, which becomes the higher latency job after the accelerator is introduced (recall Figures 5.7 and 5.8). Because the Baseline prioritizes in-order cores in the energy-oriented scheduling policy, the adoption of accelerators, in this case, leads to higher speedup when comparing to the performance-oriented scheduling policy (where OoO are preferred).

We also note the extra in-order core helps task distribution in the *Multitask* and *Edge Computing* scenarios, improving the performance of these two scenarios by 10% and 9% in the performance-oriented scheduling policy, and 10% and 7% in the energy-oriented scheduling policy.

The *Video accelerated* hw-configuration reduces up to 75% the energy consumption on the execution of Video streaming (performance-policy, Figure 5.13), mainly because of the accelerator. Energy reduction also occurs in the remaining scenarios because of the two partial-ISA cores. In the geomean, it reduces energy consumption in both policies by 38% (*performance-oriented*) and 29% (*energy-oriented*). As with the *Task-parallel* hw-configuration, reducing energy consumption is more challenging in the energy-oriented scheduling policy because the Baseline is already very efficient, leveraging its in-order cores.

5.3.5 Accelerator Rich Hw-Configuration

The *Accelerator Rich* hw-configuration removes the FP support from all big cores existent in the Baseline, and adds accelerators from previous accelerated hw-configurations

altogether. For the *Video Streaming* scenario, for example, the video accelerator and the AES accelerator combine to a 3.19× speedup in the energy-oriented scheduling policy (where the in-order cores are preferred, and the Baseline is less performing) and 2.58× speedup in the performance-oriented scenarios. In the other accelerated scenarios, the behavior depends on the policy. For the performance-oriented scenarios, the lack of big *OoO* cores causes the FP operations in non-accelerated snippets of code to be executed in the in-order GPPs, achieving less speedup than the scenarios' target configuration (e.g., the CNN Accelerated in the Road Sign Detection). In the energy-oriented scheduling policy, the lack of OoO cores is not much impactful, since the in-order cores are already preferred. There is a small reduction in the performance gains in the Edge computing scenario (comparing to the AES Accelerated, from 1.72× to 1.69× speedup), because of its higher number of workloads, which causes big cores to be required in this scenario.

The speedup in multiple scenarios makes the *Accelerator Rich* hw-configuration to achieve the higher performance gains in the geomean (1.32× in the performance-oriented scheduling policy and 1.51× in the energy-oriented scheduling policy). When the FP usage increases (*FP-Driven* scenario), the missing big full-ISA cores result in a slowdown (34% in the performance-oriented scheduling policy, and 17% in the energy-oriented scheduling policy). Hence, the designer should acknowledge the possible penalties of adopting partial-ISA in aggressive hw-configurations, as the *Accelerator Rich*.

In terms of energy, this hw-configuration is very successful. It reduces energy consumption in every scenario, regardless of the scheduling policy. For the general-purpose scenarios (*Smartphone App*, *Multitask*, and *FP-Driven*) energy reduction comes from the use of partial big cores instead of their full-ISA counterparts formerly used by the baseline. For the accelerated scenarios (*Edge Computing*, *Road Sign Detection*, and *Video Streaming*), the energy savings are even higher, since accelerators compute the accelerated regions faster and with lower peak power then GPP. For example, in the *Video Streaming* scenario in the performance-oriented policy, the *Accelerator Rich* hw-configuration consumes only 9% of the *Baseline* consumption.

Since we apply partial-ISA in the big cores, preferred in the performance-oriented scheduling policy, the normalized energy consumption has a more impacting reduction in this scheduling policy. This is reflected in the geomean of all scenarios, where the *Accelerator Rich* hw-configuration consumes only 20% of the *Baseline* in the performance-oriented scheduling policy, and 39% of the *Baseline's* energy in the energy-oriented scheduling policy.

5.4 Energy-Delay Product analysis

Finally, Figure 5.15 depicts an overview of the EDP of the *hw-configurations* in each scenario, and for both scheduling policies. The metric combines performance and energy for a higher-level view of partial-ISA impacts. In the figures, the EDP of the Baseline is used as a constant to sweep along varying values of energy ($y - axis$) and delay ($x - axis$), creating a frontier line. Being under this line means reduced (and better) EDP compared to the Baseline.

From the figure, it is clear that adopting partial-ISA cores leads to improved designs, in terms of EDP, compared to the Baseline in every scenario. Notwithstanding, we highlight that not only the partial-ISA configuration targeting each scenario improves over the Baseline, but all except one partial-ISA configuration consistently appear as a better choice over the Baseline when considering the EDP metric, regardless the scenario.

Let us now look in more detail how the different scenarios and configurations behave. For the *Smartphone App* scenario, additional in-order cores or accelerators are not useful for performance gains, as we presented in Figures 5.12a and 5.13a. However, removing FP support drives minimal performance overheads but great power savings, thus gracefully reducing energy and placing all partial-ISA hw-configurations under the Baseline EDP-ISO curve (in green, as explained above). Note that energy gains are higher in the performance scheduling policy (as also the Baseline's total energy), since partial-ISA simplify OoO cores which are most used in this scheduling policy.

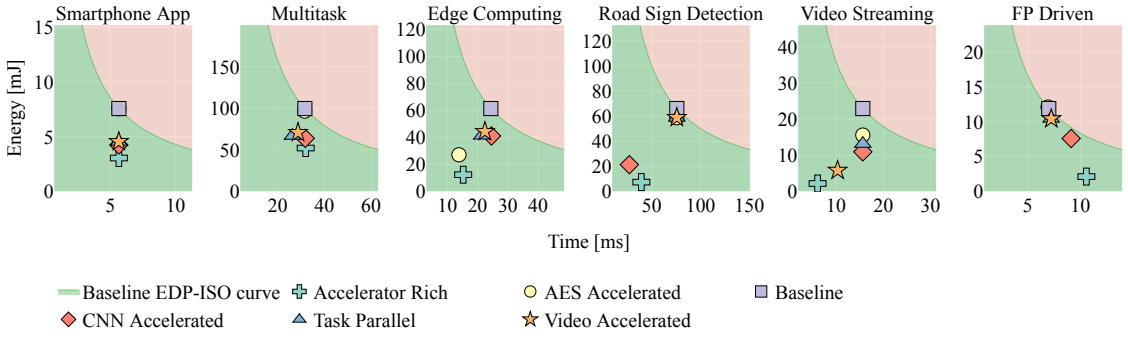
For the scenarios with speedup (*Multitask*, *Edge Computing*, *Road Sign Detection*, and *Video Streaming*), both energy and performance improvements place the partial-ISA designs nearer to the optimal EDP (in the origin of the axes). Also, we highlight the hw-configurations were well designed for their target scenarios, achieving outstanding improvements in EDP in comparison with the Baseline and also other partial-ISA hw-configurations for the same set of applications.

Finally, the *FP-Driven* scenario proves to be the most challenging for our proposal. The necessity for FP support regularly appears along with the execution of the workloads, and pushes partial-ISA configurations to the right border of the charts (with higher execution times). What grants the EDP improvement, in this case, is the power reduction in the partial cores, which also reduces energy. In the *energy-oriented scheduling policy*, however, the *AES Accelerated* hw-configuration (yellow circle) - the only hw-configuration proposed with *partial in-order cores* - cannot compensate the execution time overhead

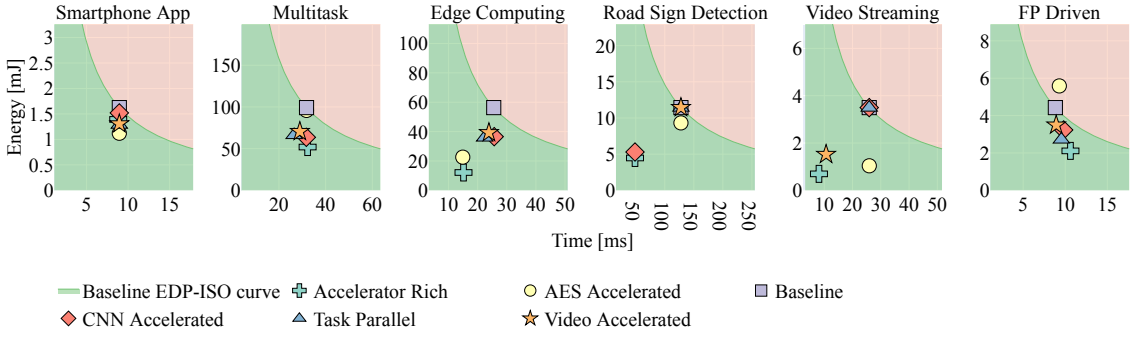
with such small power savings, and ends up as the single partial-ISA configuration with EDP higher than the Baseline.

Figure 5.15: EDP-ISO curves for *performance* and *energy* oriented policies.

(a) Performance-policy



(b) Energy-policy



Source: The author.

6 CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

In this work, we have presented a novel design methodology together with an execution mechanism to reduce energy consumption and improve the performance of MPSoC architectures in multiple situations. We propose to simplify a few cores in the MPSoC, neglecting their support for costly ISA-extensions, which creates slack in power and area. As a case study, we considered the FP instructions and the related datapath in RISC-V ISA processors, observing the savings in area and power through hardware description synthesis.

We exploit the extra budget in both power and area to add more computing nodes into the system. We considered the adoption of either more in-order cores, to increase the ability of the system to process multiple workloads concurrently, and the addition of ASICs to increase the ability of the system to handle time-costly jobs such as video decoding, neural network inference, and cryptography. Additionally, to this design methodology, we detail a mechanism to coordinate the execution of workloads in such designs. This mechanism details how to mark workloads dependency on full-ISA cores or accelerators, and two mapping strategies to execute the workloads, either for performance or energy.

We have explained in detail how we combine research and industry tools to collect data for partial-ISA experimentation. Moreover, we have discussed how our in-house simulator computes such data to simulate the scheduling policies and associated task migration, the associated overheads, the gains in performance, and savings in energy. We define a variety of scenarios and hardware configurations considering partial-ISA to have a broad idea on the pros and cons of our proposal. We observed that adopting partial-ISA leads to energy savings because of the removal of power-hungry FP datapath, especially in OoO processors. At the same time, we note that our scheduling policies were capable of taking advantage of the additional hardware, allowing task-parallelism and hardware acceleration to bloom. Finally, we have discussed how both energy and performance gains are generally combined in the proposed designs, reducing the EDP for almost every scenario and configuration analyzed.

6.2 Work Limitations

Although we have presented several benefits in adopting partial-ISA for MPSoC's design, we now acknowledge some limitations of the present work. The discussion certainly does not present an exhaustive list of possible limitations, but it is made with our best efforts, expecting to serve as a valuable appointment in the improvement spots that can be tackled by future works on the area.

First, we acknowledge our hardware configuration compositions are based on the area and power of the computing nodes individually (either full or partial cores, and accelerators). However, it is likely that the power and area slack created from the partial cores may not be entirely converted into the extra nodes. Notably, the interconnection cost of having more nodes in the MPSoC is something to be studied under our proposal. Also, the additional nodes may pressure on the L2 caches, since they are generally shared, and may require their size to be enlarged to compensate. This is also an open question in our work, and requires simulation at the system level, considering the whole memory hierarchy, which our simulation lacks to evaluate.

It is also essential to have accurate data on the performance of the adopted accelerators, so the gains in performance and energy also became more trustworthy. The speedup relationship between executing a kernel in accelerators and GPPs depends on the hardware descriptions and the software implementations counterparts (depending on the quality of both implementations). We can reach a fair comparison submitting the software kernel into High-Level Synthesis (HLS), deriving a hardware description precisely for the software provided. Even though, there are a variety of algorithms and hardware implementations that can be adopted for solving a given problem, which makes it difficult for us to select a single case and define it as the most representative. A design exploration of the different accelerators is also an option for covering more situations but requires different hardware descriptions for the same problem at hand, which may be challenging to have. Nevertheless, we want to execute tasks in the accelerators to collect their speedup more precisely.

Notwithstanding, there are aspects of the scheduling and software level that can also be tackled by future work. For example, although we adopt round-robin scheduling in a time preemptive approach as in any modern OS, we do not study task priority and Dynamic Voltage and Frequency Scaling (DVFS). Adopting both mechanisms would place our analysis even closer to real-life implementations. However, the number of possible

combinations for the executing scenarios under these circumstances would also make it difficult to separate the impact from the scheduler and the partial-ISA adoption. Also, the existence of heterogeneous ISA cores may bring issues for scheduling multi-thread programs, where balanced execution can be threatened if FP is required, for example. Since multi-thread execution contains synchronization points, partial-ISA cores can delay the end of a task (e.g., if it has to migrate to another core to execute FP instructions), impacting the synchronization. These issues exist in nowadays heterogeneous systems (e.g., big.LITTLE), since big and little processors have unbalanced performance, but may increase with partial-ISA.

Finally, we highlight adopting hardware accelerators brings on performance and energy gains but also introduces issues with programmability. This discussion already exists today, since many of the application-specific hardware that has been proposed in past years demands extra programming effort. We recall there are well-succeed application-specific languages (such as the CUDA language for Graphics Processing Units (GPUs)). However, the burden to program a non-conventional design must be taken into account when designing MPSoC, such as in our proposal.

6.3 Future Work

Based on the conclusions and limitations mentioned above, we observe different paths we can cover in future work. Particularly, we aim to investigate how the addition of computing nodes into an MPSoC impacts its interconnections and the L2 cache. After, we will modify our current simulation tools accordingly and also check if the proposed hardware configurations will remain under the baseline constraints. Additionally, we aim to implement the hardware accelerators in HLS tools, so we have a 1:1 software and hardware algorithm, increasing the certainty of our experiments. Another enhancement we look forward to implementing in our simulation flow is a priority queue for the workloads. With this, we can prioritize latency-critical workloads, for instance, resulting in wiser scheduling policies.

6.4 Publications

Following, we present the publications achieved by the author during the dissertation period. The first two manuscripts are related with the present work, and had been accepted for international conferences. The remaining publications refer to works that are not related with this manuscript, but were also published during the authors' MSc period.

- Increasing MPSoCs Design Space with partial-ISA Processors, **IEEE International Conference on Electronics Circuits and Systems, ICECS**, (BECKER; SOUZA; BECK, 2019)

- Tuning the ISA for increased heterogeneous computation in MPSoCs, **Design, Automation and Test in Europe Conference, DATE**, (BECKER; SOUZA; BECK, 2020)

- BRAM-based function reuse for multi-core architectures in FPGAs, **Microprocessors and Microsystems**, (BECKER et al., 2018b)

- A Low-Cost BRAM-Based Function Reuse for Configurable Soft-Core Processors in FPGAs, **Applied Reconfigurable Computing. Architectures, Tools, and Applications, ARC** (BECKER et al., 2018a)

- Machine Learning-Based Processor Adaptability Targeting Energy, Performance, and Reliability, **IEEE Computer Society Annual Symposium on VLSI, ISVLSI**, (SARTOR et al., 2019)

- Dynamic Trade-off among Fault Tolerance, Energy Consumption, and Performance on a Multiple-Issue VLIW Processor, **IEEE Transactions on Multi-Scale Computing Systems**, (SARTOR et al., 2018)

- A fast and accurate hybrid fault injection platform for transient and permanent faults, **Design Automation for Embedded Systems, DAES**, (SARTOR; BECKER; BECK, 2018)

- Simbah-FI: Simulation-Based Hybrid Fault Injector, **Brazilian Symposium on Computing System Engineering, SBESC**, (SARTOR; BECKER; BECK, 2017)

REFERENCES

ANGIOLINI, F. et al. An integrated open framework for heterogeneous MPSoC design space exploration. In: EUROPEAN DESIGN AND AUTOMATION ASSOCIATION. **Proceedings of the conference on design, automation and test in Europe: proceedings**. [S.l.], 2006.

APPLE. **iPhone XS - A12 Bionic - Apple**. 2018. Available from Internet: <https://www.apple.com/iphone-xs/a12-bionic/>.

ARM. **Technologies | DynamIQ – Arm Developer**. 2017. Available from Internet: <https://developer.arm.com/technologies/dynamiq>.

ARM Limited. **Cortex-A15 MPCore Technical Reference Manual**. [S.l.], 2012. Available from Internet: <http://www.arm.comhttp://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0438e/index.html>.

ARM Limited. **ARM ® Cortex ®-A75 Core Technical Reference Manual**. [S.l.], 2016. Available from Internet: <http://www.arm.com/about/trademark-usage-guidelines.php>.

ASANOVIC, K. et al. **The berkeley out-of-order machine (boom): An industry-competitive, synthesizable, parameterized risc-v processor**. [S.l.], 2015.

ASANOVIC, K. et al. The rocket chip generator. **EECS Department, UC, Berkeley, Tech. Rep. UCB/EECS-2016-17**, 2016.

ASANOVIĆ, K.; PATTERSON, D. A. **Instruction Sets Should Be Free: The Case For RISC-V**. [S.l.], 2014. Available from Internet: <http://www.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-146.html>.

BECKER, P. H. E. et al. A Low-Cost BRAM-Based Function Reuse for Configurable Soft-Core Processors in FPGAs. In: VOROS, N. et al. (Ed.). **Applied Reconfigurable Computing. Architectures, Tools, and Applications**. Cham: Springer International Publishing, 2018. p. 499–510. ISBN 978-3-319-78890-6.

BECKER, P. H. E. et al. BRAM-based function reuse for multi-core architectures in FPGAs. **Microprocessors and Microsystems**, v. 63, p. 237–248, nov 2018. ISSN 01419331.

BECKER, P. H. E.; SOUZA, J. D.; BECK, A. C. S. Increasing MPSoCs Design Space with partial-ISA Processors. In: **IEEE International Conference on Electronics Circuits and Systems, ICECS**. [S.l.: s.n.], 2019.

BECKER, P. H. E.; SOUZA, J. D.; BECK, A. C. S. Tuning the ISA for increased heterogeneous computation in MPSoCs. In: **Design, Automation and Test in Europe Conference, DATE**. [S.l.: s.n.], 2020.

BINKERT, N. et al. The gem5 simulator. **ACM SIGARCH Computer Architecture News**, ACM, 2011.

BLEM, E. et al. Power struggles: Revisiting the RISC vs. CISC debate on contemporary ARM and x86 architectures. In: **HPCA'13**. [S.l.: s.n.], 2013. ISBN 978-1-4673-5587-2.

BLEM, E. et al. ISA Wars: Understanding the Relevance of ISA being RISC or CISC to Performance, Power, and Energy on Modern Architectures. **ACM Transactions on Computer Systems**, 2015. ISSN 07342071.

CAVIGELLI, L. et al. Origami: A Convolutional Network Accelerator. In: **Proceedings of the 25th Edition on Great Lakes Symposium on VLSI**. New York, NY, USA: ACM, 2015. (GLSVLSI '15), p. 199–204. ISBN 978-1-4503-3474-7. Available from Internet: <http://doi.acm.org/10.1145/2742060.2743766>.

CONSTANTINOU, T. et al. Performance implications of single thread migration on a chip multi-core. **SIGARCH Comput. Archit. News**, ACM, New York, NY, USA, 2005.

Cota, E. G. et al. An analysis of accelerator coupling in heterogeneous architectures. In: **2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)**. [S.l.: s.n.], 2015. p. 1–6. ISSN 0738-100X.

ENDO, F. et al. Micro-architectural simulation of embedded core heterogeneity with gem5 and McPAT. **RAPIDO '15**, p. 1–6, 2015.

ESMAEILZADEH, H. et al. Dark silicon and the end of multicore scaling. In: **2011 38th Annual International Symposium on Computer Architecture (ISCA)**. [S.l.: s.n.], 2011. p. 365–376. ISSN 1063-6897.

GREENHALGH, P. Big.little processing with arm cortex-a15 and cortex-a7. **ARM White Paper**, v. 17, 2011.

GUTHAUS, M. R. et al. MiBench: A free, commercially representative embedded benchmark suite. In: **IEEE WWC-4'01**. [S.l.]: IEEE, 2001. p. 3–14.

HENNESSY, J. L.; PATTERSON, D. A. **Computer Architecture: A Quantitative Approach (The Morgan Kaufmann Series in Computer Architecture and Design)**. 6. ed. [S.l.]: Morgan Kaufmann, 2017. ISBN 0128119055.

HENNESSY, J. L.; PATTERSON, D. A. A New Golden Age for Computer Architecture. **Commun. ACM**, ACM, New York, NY, USA, v. 62, n. 2, p. 48–60, 2019. ISSN 0001-0782. Available from Internet: <http://doi.acm.org/10.1145/3282307>.

Intel Corporation. **Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3 (3A, 3B, 3C & 3D): System Programming Guide**. [S.l.], 2016. Available from Internet: <http://www.intel.com/design/literature.htm.>

Intel Corporation. **Intel Intrinsics Guide**. 2018. Available from Internet: <https://software.intel.com/sites/landingpage/IntrinsicsGuide/{\#}expand=2815,2434,5658,5022,4804,665,631,628,44https://software.intel.com/sites/landingpage/IntrinsicsGui>.

JIMBOREAN, A.; LOECHNER, V.; CLAUSS, P. Handling multi-versioning in llvm: Code tracking and cloning. In: **WIR 2011: Workshop on Intermediate Representations, in conjunction with CGO 2011**. [S.l.: s.n.], 2011.

KARAGIANNOPOULOS, L. **'Facebook effect' turns Swedish steel town into tech hot-spot - Reuters**. 2018. Available from Internet: <https://uk.reuters.com/article/us-sweden-facebook-datacentre/facebook-effect-turns-swedish-steel-town-into-tech-hot-spot-idUKKBN1I827I>.

LEE, W. et al. Exploring Heterogeneous-ISA Core Architectures for High-Performance and Energy-Efficient Mobile SoCs. **GLSVLSI'17**, 2017.

LI, S. et al. The mcpat framework for multicore and manycore architectures: Simultaneously modeling power, area, and timing. **ACM Transactions on Architecture and Code Optimization (TACO)**, ACM, v. 10, n. 1, p. 5, 2013.

LI, T. et al. Efficient operating system scheduling for performance-asymmetric multi-core architectures. In: **ACM/IEEE Conference on Supercomputing**. [S.l.: s.n.], 2007.

LI, T. et al. Operating system support for overlapping-ISA heterogeneous multi-core architectures. **International Symposium on High-Performance Computer Architecture, HPCA**, 2010.

LOPES, B. et al. Shrink: Reducing the ISA Complexity Via Instruction Recycling. **International Symposium on Computer Architecture, ISCA**, p. 311–322, 2015.

MARTINS, M. et al. Open Cell Library in 15Nm FreePDK Technology. In: **International Symposium on Physical Design**. [S.l.]: ACM, 2015. p. 171–178. ISBN 978-1-4503-3399-3.

PARK, J. et al. Microarchitecture-Aware Code Generation for Deep Learning on Single-ISA Heterogeneous Multi-Core Mobile Processors. **IEEE Access**, v. 7, p. 52371–52378, 2019.

PAVER, N. C.; KHAN, M. H.; ALDRICH, B. C. Accelerating mobile multimedia using intel wireless MMX^{TM} technology. In: **Proceedings - IEEE Sixth International Symposium on Multimedia Software Engineering, MSE 2004**. [S.l.: s.n.], 2004. p. 491–498. ISBN 0769522173.

PEEMEN, M.; MESMAN, B.; CORPORAAL, H. Speed sign detection and recognition by convolutional neural networks. In: **Proceedings of the 8th international automotive congress**. [S.l.: s.n.], 2011. p. 162–170.

POUCHET, L. N.; YUKI, T. PolyBench/C 4.1. **Retrieved May2015 from http://web. cse. ohio-state. edu/˜ pouchet/software/polybench**, 2015.

RABAEY, J. M.; CHANDRAKASAN, A. P.; NIKOLIC, B. **Digital integrated circuits**. [S.l.]: Prentice hall Englewood Cliffs, 2002.

REDDY, D. et al. Bridging functional heterogeneity in multicore architectures. **ACM SIGOPS Operating Systems Review**, p. 21, 2011.

RICHARDSON, I. E. **The H.264 Advanced Video Compression Standard**. 2nd. ed. [S.l.]: Wiley Publishing, 2010. ISBN 0470516925, 9780470516928.

ROSVALL, K.; SANDER, I. A Constraint-based Design Space Exploration Framework for Real-time Applications on MPSoCs. In: **Proceedings of the Conference on Design, Automation & Test in Europe**. [S.l.]: European Design and Automation Association, 2014. (DATE '14). ISBN 978-3-9815370-2-4.

ROSVALL, K.; SANDER, I. Flexible and Tradeoff-Aware Constraint-Based Design Space Exploration for Streaming Applications on Heterogeneous Platforms. **ACM Transactions on Design Automation of Electronic Systems**, ACM, New York, NY, USA, v. 23, n. 2, nov 2017. ISSN 10844309.

RUSCHIVAL, T. **Overview :: Avalon AES ECB-Core (128, 192, 256 Bit) :: OpenCores**. 2017. Available from Internet: <https://opencores.org/projects/avs{_}>.

SAMSUNG. **Exynos 9 Series 9820 Processor: Specs, Features | Samsung Exynos**. 2018. Available from Internet: <https://www.samsung.com/semiconductor/minisite/exynos/products/mobileprocessor/exynos-9-series-9820/>.

SARTOR, A. L.; BECKER, P. H. E.; BECK, A. C. S. Simbah-FI: Simulation-Based Hybrid Fault Injector. In: **Brazilian Symposium on Computing System Engineering, SBESC**. [S.l.: s.n.], 2017. v. 2017-Novem. ISBN 9781538635902. ISSN 23247894.

SARTOR, A. L.; BECKER, P. H. E.; BECK, A. C. S. A fast and accurate hybrid fault injection platform for transient and permanent faults. **Design Automation for Embedded Systems**, nov 2018. ISSN 1572-8080. Available from Internet: <https://doi.org/10.1007/s10617-018-9217-0>.

SARTOR, A. L. et al. Dynamic Trade-off among Fault Tolerance, Energy Consumption, and Performance on a Multiple-Issue VLIW Processor. **IEEE Transactions on Multi-Scale Computing Systems**, v. 4, n. 3, p. 327–339, 2018. ISSN 2332-7766.

SARTOR, A. L. et al. Machine Learning-Based Processor Adaptability Targeting Energy, Performance, and Reliability. In: **IEEE Computer Society Annual Symposium on VLSI, ISVLSI**. [S.l.: s.n.], 2019. p. 158–163.

SHAFAEI, A. et al. FinCACTI: Architectural analysis and modeling of caches with deeply-scaled FinFET devices. In: **Proceedings of IEEE Computer Society Annual Symposium on VLSI, ISVLSI**. [S.l.: s.n.], 2014. ISBN 9781479937639. ISSN 21593477.

SHAFIQUE, M. et al. The EDA Challenges in the Dark Silicon Era: Temperature, Reliability, and Variability Perspectives. In: **Proceedings of the 51st Annual Design Automation Conference**. New York, NY, USA: ACM, 2014. (DAC '14), p. 185:1—-185:6. ISBN 978-1-4503-2730-5. Available from Internet: <http://doi.acm.org/10.1145/2593069.2593229>.

SINGH, A. K. et al. Mapping on multi/many-core systems: survey of current and emerging trends. In: IEEE. **2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)**. [S.l.], 2013.

SMITH, J.; SOHI, G. The microarchitecture of superscalar processors. **Proceedings of the IEEE**, v. 83, n. 12, p. 1609–1624, 1995. ISSN 00189219. Available from Internet: <http://ieeexplore.ieee.org/document/476078/>.

STILLMAKER, A.; BAAS, B. Scaling equations for the accurate prediction of CMOS device performance from 180nm to 7nm. **Integration, the VLSI Journal**, v. 58, p. 74–81, 2017. ISSN 01679260. Available from Internet: <http://www.sciencedirect.com/science/article/pii/S0167926017300755>.

SUEHRING, K. **H. 264/AVC jm reference software download**. 2010. Available from Internet: <http://iphome.hhi.de/suehring/tml/download/>.

TAN, C. et al. Locus: Low-power customizable many-core architecture for wearables. **ACM Trans. Embed. Comput. Syst.**, ACM, New York, NY, USA, v. 17, n. 1, nov. 2017.

THEIS, T. N.; Philip Wong, H. S. The End of Moore's Law: A New Beginning for Information Technology. **Computing in Science and Engineering**, IEEE Computer Society, v. 19, n. 2, p. 41–50, mar 2017. ISSN 15219615.

Van Stralen, P.; PIMENTEL, A. Scenario-based design space exploration of MPSoCs. In: IEEE. **Proceedings - IEEE International Conference on Computer Design: VLSI in Computers and Processors**. [S.l.], 2010. ISBN 9781424489350. ISSN 10636404.

VENKAT, A. et al. Composite-ISA Cores: Enabling Multi-ISA Heterogeneity Using a Single ISA. In: **2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)**. [S.l.: s.n.], 2019.

VENKAT, A.; TULLSEN, D. Harnessing ISA diversity: Design of a heterogeneous-ISA chip multiprocessor. **ISCA'14**, p. 121–132, 2014. ISSN 10636897.

WALDROP, M. M. The chips are down for Moore's law. **Nature News**, v. 530, n. 7589, p. 144, 2016.

WESTE, N. E. H.; HARRIS, D. M. **CMOS VLSI Design: A Circuits and Systems Perspective**. [S.l.]: Pearson Education India, 2013. 1689–1699 p. ISSN 1098-6596. ISBN 9788578110796.

WOLF, W.; JERRAYA, A. A.; MARTIN, G. Multiprocessor system-on-chip (MPSoC) technology. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, IEEE, v. 27, n. 10, p. 1701–1713, 2008.

XILINX. **Zynq UltraScale+ MPSoC**. 2019. Available from Internet: <https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html>.

XU, K.; CHOY, C.-S. A power-efficient and self-adaptive prediction engine for H. 264/AVC decoding. **IEEE Transactions on very large scale integration (VLSI) systems**, IEEE, v. 16, n. 3, p. 302–313, 2008.