Improving Multitask Performance and Energy Consumption With Partial-ISA Multicores

Jeckson Dellagostin Souza[a], Pedro Henrique Exenberger Becker[a,b], Antonio Carlos Schneider Beck[a]

[a] *Universidade Federal do Rio Grande do Sul - Porto Alegre, Brazil*
[b] *Universitat Politècnica de Catalunya - Barcelona, Spain*

Abstract

Modern GPPs implement specialized instructions in the form of ISA extensions aiming to increase the performance of emerging applications. These extensions impose a significant overhead in the area and power of the processor because of their specific datapaths (e.g. hardware for SIMD and FP instructions may represent more than half of the core area). Considering that some devices (e.g., edge computing), must be as energy- and area-efficient as possible, and the sporadic usage of specialized instructions in many applications, we propose PHISA multicores. PHISA is composed of heterogeneous cores of the same single base ISA, but asymmetric functionality: some of the cores do not fully implement the costly instruction extensions, making room for the designers to add more efficient cores. We show that PHISA increases performance in (32%) and reduces energy consumption in (82%) compared to full-ISA systems with the same power budget, in multi-workload environments.

Keywords: heterogeneity, partial-isa, overlapping-isa, reduced-isa, energy efficiency, scheduling

1. Introduction

The Internet of Things (IoT) domain is composed of systems of different complexity: from very simple nodes driven by ultra-low power micro-

Email addresses: jeckson.souza@inf.ufrgs.br (Jeckson Dellagostin Souza), pedro@ac.upc.edu (Pedro Henrique Exenberger Becker), caco@inf.ufrgs.br (Antonio Carlos Schneider Beck)

controllers forming wireless sensors networks [1] to complex wearables that demand high-performance processors [2]. It is natural that, as these systems evolve, computation will be brought closer to the user, which makes edge computing important for IoT applications [3]. Nonetheless, although it requires high-performance processors, power and area will always be a strong constraint in IoT systems. Therefore, in an environment of fast-paced application evolution, the adaptability of executing different applications provided by General Propose Processors (GPP), allied with techniques for reducing energy consumption, will be essential for IoT devices.

Current embedded systems implement a variety of strategies to efficiently deliver high-performance throughput at small power budgets, which can inspire IoT processor designs. One of the strategies that engage directly into the processor core efficiency is the single-ISA (Instructions Set Architecture) heterogeneous processors [4]. Examples adopted by the industry are the ARM big.LITTLE[5], and - more recently - ARM DynamIQ[6], which comprise distinct cores with different performance and energy characteristics in the same die. As they all implement the same ISA, threads can transparently migrate between processors, allowing a scheduler to allocate jobs according to the applications needs and non-functional requirements, such as performance and energy.

The ISA of these GPPs has been incrementally tailored to increase the performance of emerging applications. Each architectural iteration adds newer instructions in the form of extensions (e.g., SSE and AVX in the x86, and NEON and SVE in the ARM), increasing the complexity of the microarchitecture. However, not all applications will take advantage of such instructions. For instance, x86 AVX SIMD (Single Instruction Multiple Data) instructions are specifically used for highly vectorized applications.

As the figure 1 shows, this is no different for NEON instructions in ARM architectures, which is the instruction extension that contains both the Floating Point (FP) and the Single Instruction Multiple Data (SIMD) instructions. This figure shows the percentage of dynamic instructions executed in a wide range of workloads from different benchmark sets (details on the experiments are in Section 3). It demonstrates how NEON instructions (i.e., both SIMD and FP operations) are underused, with many of the analyzed benchmarks not issuing any Floating Point or SIMD instructions at all. Besides, the NEON functional unit adds considerable area overhead, as one can observe in figure 2, which shows the area breakdown of components for two ARM processors. According to our experiments, the ARM A7, an in-order processor,

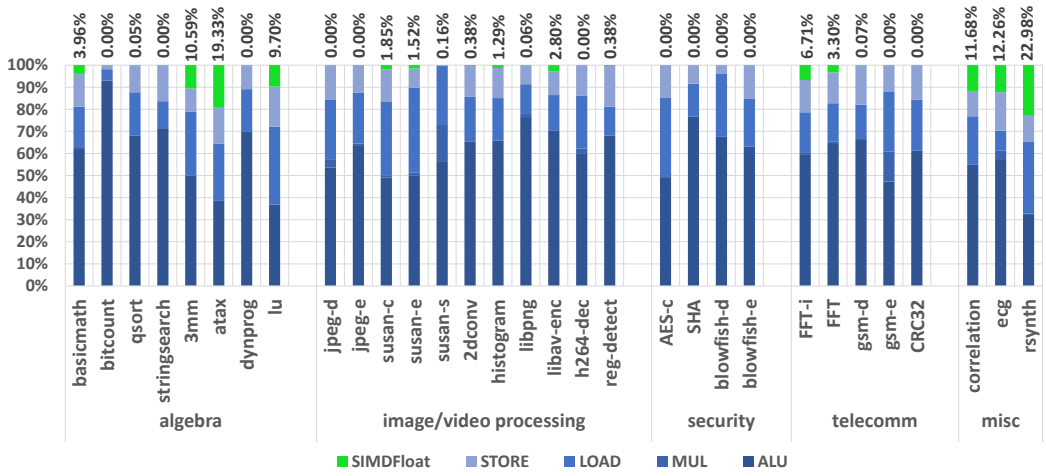
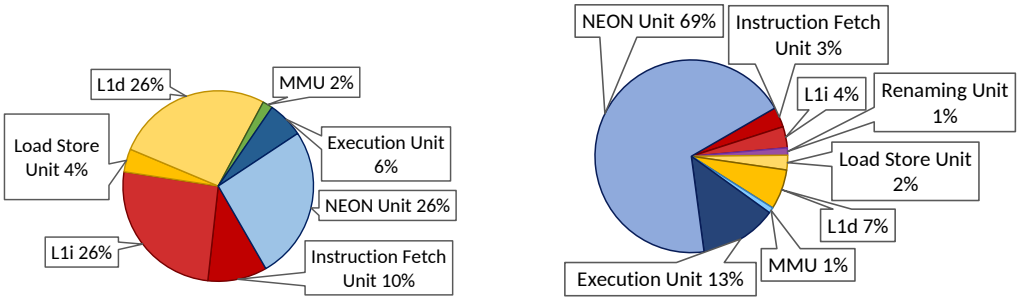


Figure 1: Instruction breakdown by category. The percentages in the top are from the SIMDFloat (NEON) instructions only.

has a single NEON pipeline that occupies 26% of its total core area, while the A15, an out-of-order processor, has two larger NEON pipelines that fill 69% of its core area. This data suggests that, even though there should be some kind of hardware support to execute these extensions - as they can be important for specific applications, they will come at high implementation costs. Furthermore, it is very likely that implementing such support in every core in a multicore system is neither performance- nor energy-wise.

Given this scenario, we propose the PHISA (Partially Heterogeneous ISA) Multicore. A PHISA system comprises cores that partially implement an ISA, removing selected instruction extensions and all the physical components that are specific to the execution of those instructions. The best candidates for removal are the instructions that require vast portions of the processor area, such as shown in figure 2, and that are not commonly used, as demonstrated in figure 1. Although full-ISA cores may still be required to execute the extensions, partial-ISA cores can replace a portion of these full cores, reducing the area and the power dissipation of the entire multicore system. With the resultant freed area and power, the designer can introduce new cores, which may significantly improve performance on multi-workload environments. The base reasoning of the PHISA system is that the designer can trade part of the performance provided by the ISA extension units (in the form of processed Instructions Per Cycle (IPC) *for specific applications*) for more cores that can increase the system throughput (in the form of Task



(a) A7 in-order processor (b) A15 out-of-order processor

Figure 2: Area breakdown by processor component.

Parallelism (TP) *for general applications*). Therefore, instead of introducing more area and power to the highly constrained IoT system environment, we use the resources once given to the expensive instruction extensions to improve our system.

As in any heterogeneous system of homogeneous ISA, correctly allocating threads accordingly to their needs is also essential in our partial-ISA system. Nonetheless, our system introduces an extra challenge: the scheduler must deal with tasks that require removed instruction extensions and allocate them to full-ISA cores. Therefore, we also propose different approaches for scheduling and emulation of such instructions, aiming for both performance and energy optimizations.

We evaluate the proposed system with different ratios of Full/Partial ISA cores, using as a case study the ARM architecture with and without the NEON instruction set, even though PHISA can be generalized to most ISAs and extensions. Furthermore, our evaluation extends to different organization scenarios, comparing both systems of symmetric and asymmetric performance and partial- and full-ISA. We show that a PHISA multicore can improve performance and reduce energy in traditional edge computing scenarios when compared to its full ISA system counterpart, considering the same power budget. Furthermore, we show how PHISA compares to existing heterogeneous processors (such as DynamIQ designs) and how it can also improve performance and energy consumption in these scenarios using different scheduling policies.

The remaining of this article is organized as follows. In section 2 we present details on the implementation of PHISA multicores and its scheduler

requirements. Section 3 describes the methodology used to evaluate the proposed system, while section 4 presents the many analyzed results for the different scenarios. Section 6 discusses recent works related to this paper. Finally, section 7 concludes this work.

2. PHISA Multicore

2.1. Proposed System

The PHISA multicore design removes hardware components specifically used by an ISA extension while leaving the remaining microarchitecture of a core unaltered. This core, which we call a Partial ISA core, keeps its ability to execute instructions from its base ISA. Therefore, performance is only affected for the removed instructions, as parameters such as issue-width, execution order, and branch prediction are all kept the same. With the area and power freed from removing these ISA extensions, we propose to increase the core count of the system with smaller and simpler in-order cores. Figure 3 demonstrates our proposed design.

To handle the support for asymmetric ISA cores in a PHISA system, a scheduler must be aware of the presence of faulting instructions. Whenever a partial-ISA core fetches an instruction from a non-implemented extension, it sends an unsupported op exception to the Operating System (OS) (e.g. the already existent X86_TRAP_UD trap in linux), which must handle this exception - through the scheduler -, either emulating the instruction or migrating it to a full core.

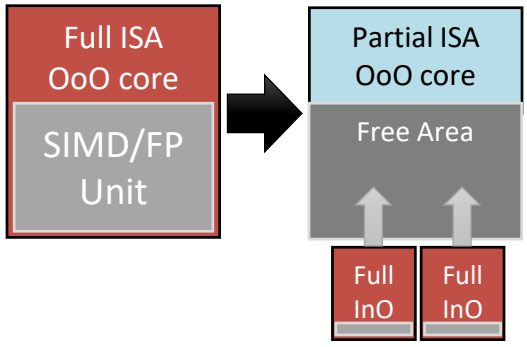


Figure 3: Example of PHISA configuration. The resources freed by an instruction extension are used to increase the core count.

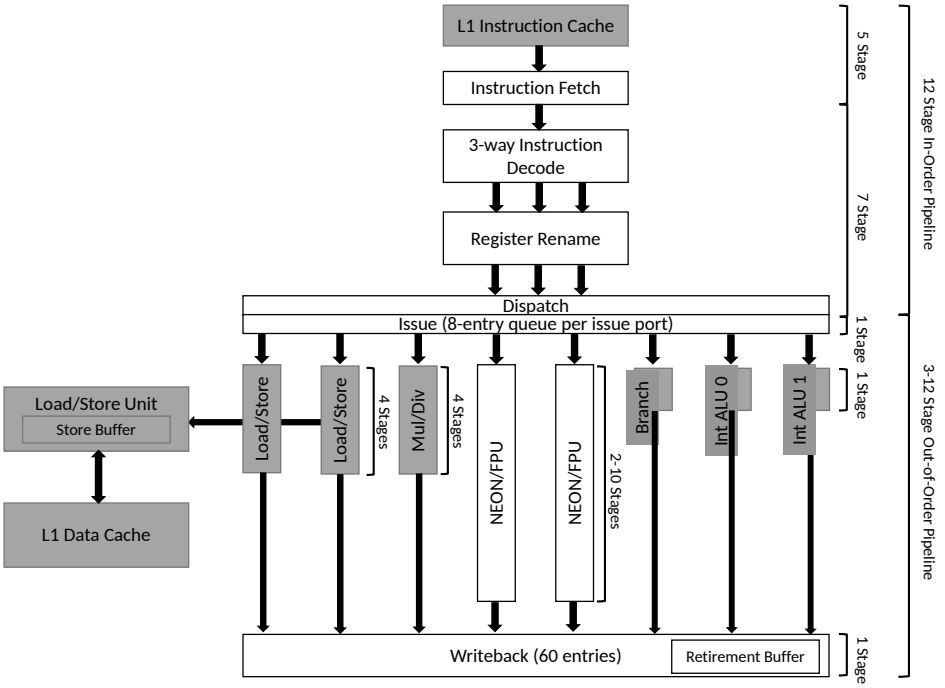


Figure 4: A15 pipeline blocks. White blocks are components that can be simplified when removing NEON support.

Each ISA extension adds its particular logic complexity. In this work, we have focused on the NEON instructions from the ARM architecture (both SIMD and FP operations) as a use case, as it is a high source of logic overhead in the processors (figure 2). When excluding the NEON extension, we can safely remove the entire SIMD and floating point pipelines from the execution unit of the processor. The decoder stage can also be trimmed by removing support for these instructions, as well as the FP instruction window in the fetch stage of out-of-order designs. Separate FP register renaming table and the FP register file (assuming the multiplication unit is adapted to use the integer RF) in the dispatch stage can also be removed. Other general targets for trimming include routing logic like write-backs, forwarding and even the clock-tree of the processor. Figure 4 shows a diagram of the Cortex A15 pipeline. The white boxes indicate the processor components that can be simplified when removing NEON instructions.

Other ISA extensions (not considered in this work) would incur into more logic trimming in different regions of the processor. For instance, DSP in-

structions would simplify the integer pipeline of the processor, by removing the Multiply-Accumulate (MAC) operations. It is important to notice that some extensions are simpler to remove - from a design perspective - than others. For instance, the NEON components are quite modular in ARM architectures and could be easily removed - in fact, they are optional in some ARM processor families, such as the A9. Nonetheless, design modifications are unavoidable if the goal is to achieve an efficient PHISA system.

Considering that a PHISA system is composed of a combination of full- and partial-ISA cores, it will always be able to execute any instruction from the architecture set (by migrating tasks to the full cores whenever necessary). *Therefore, while PHISA multicore does not require any special compilers or tools and can execute any application that has already been deployed, it requires modifications in micro architecture and in the OS, especially in its scheduler.*

2.2. Scheduling

As with any heterogeneous processor, an efficient scheduler plays a major role in the performance and energy consumption. In a PHISA multiprocessor, the scheduler must be aware of which cores are capable of executing the ISA extensions so that it can migrate workloads from partial to full cores when necessary. Figure 5 shows a graphical representation of the scheduler decisions. Each core (which can be a partial- or full-ISA core) keeps a queue of workloads (different applications) to execute (1). When a core is idle, it fetches a workload from its queue in a FIFO manner and executes the workload until a migration event is triggered.

There are two events in which a workload can migrate from a core to another. The first is when a partial core fetches an unimplemented instruction (2). When a typical processor fetches an instruction that cannot be decoded, it generates a trap to the operating system, which would signal a kill command for the process. In this work, we implement a fault-and-migrate strategy [7] to handle the reallocation of workloads. Instead of treating the trap with a signal kill, the operating system activates the ISA-aware scheduler that migrates the workload to the less busy full core in the system (with the shortest workload queue)(3). The second event for migration is activated after a workload has been executed for a minimum time in a full core(4). Similarly, the workload will be migrated to the less busy core (which can be either partial or full)(5).

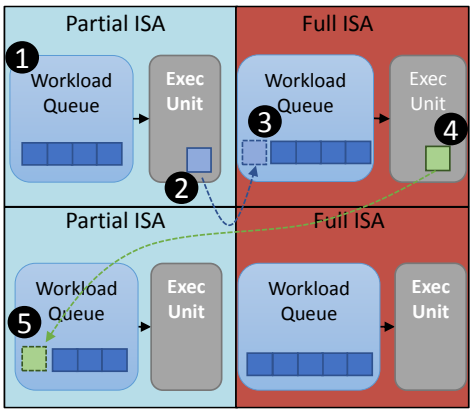


Figure 5: Scheduler events between full and partial ISA cores.

Many applications present the behavior of interleaving integer and floating point operations, which would cause frequent back and forth migrations. To handle it, we consider a minimum time a thread must stay on full cores of 160K cycles, as per suggested by previous studies as a period that introduces minimal impact on performance [8]. The established minimum time on a core helps to reduce these migrations when the application is executing on the full core. However, they will still happen if the interleaved application is rescheduled to a partial core after the minimum time. Our scheduler handles this situation by *(a) prioritizing migration of FP applications to full cores or (b), if available to the system, by triggering FP instruction emulation in software.* In the case of software emulation, the task requiring the non-implemented instruction can remain to execute in the partial core for a threshold time (we use the same 160K cycles as threshold). We explore both these solutions in our experiments.

2.3. Advanced Scheduler and Policies

In the previous subsection we introduced a simple scheduler to cope with partial-ISA migrations that arise from our proposed PHISA system. However, this scheduler lacks support for different optimization goals (e.g., maximize performance), which is an essential feature in a scheduler for heterogeneous processors. To overcome this, we go one step further and improve the former scheduler, introducing scheduling policies on it. The scheduling policies coordinate task allocation in order to prioritize either *overall performance* or *energy efficiency*.

In this improved version of the scheduler, we introduce the following modifications:

- **The workload queue is unified.** Instead of using a queue for each core, all workloads are sent to the same queue, where they can be prioritized.
- **Preemption in all cores.** In the former scheduler we time-preempt applications running in the full-cores to share NEON resources among different tasks. The advances scheduler adopts time preemption on partial-cores, so all applications have the same time slice to execute, giving fairness to the system.
- **Application annotation.** Since the queue is now unified, we need to keep track of applications that were preempted by partial-cores when tried to execute a NEON instruction. The scheduler mark those applications to be assigned to full-cores in their next allocation (otherwise they would be preempted from the partial-cores once again). The mark is removed when applications are time preempted without requiring NEON instructions during their last execution time slice.

Given these modifications, we create two versions of the advanced scheduler, one trying to optimize execution for performance, and the other for energy consumption. For performance optimization, we assume that the system's OoO cores always present better performance than the in-order cores, thus we prioritize allocation in the OoO cores first. For energy, we assume that the in-order cores will always have better energy efficiency, prioritizing the allocation on them. Furthermore, applications that are not specifically marked to use instruction extensions (are executing instructions from the base ISA) will prioritize execution in partial-ISA cores, leaving the full free for applications that need these cores.

In this new scheduler scheme, all workloads are released from their cores after the preemption phase (again of 160K cycles) and sent back to the now unified workload queue. Cores that fetch non-supported instructions will immediately release their workloads and call the scheduler for a new assignment. These workloads are annotated as ISA dependent on their next allocation. The algorithm is executed after the preemption release for all workloads in the queue, in a FIFO manner, until no core is left idle or the queue is empty.

3. Methodology

Modeling and Simulation: We have used the gem5 simulator [9] to model the different versions of the ARM's A7 and A15 processors. For area and power measurements, we have modeled the same processors in McPAT [10] using a node technology of $28nm$, with both running at the same frequency of 2GHz. Our models consider the entire core (including MMU and instruction and data L1 caches) without L2 caches. Although McPAT models its components according to an A9 processor, we have used an approach similar to the one proposed in [11] to model the A7 and A15. The authors show that this approach results in models very close to the real processors. McPAT also allows for configurations without FP and SIMD units (by simply setting the FP related tags in the template to zero), which also triggers the exclusion of the FP instruction window, the FP Register File (RF) and the FP register renaming structures. Nonetheless, this approach is a conservative model, as removing the NEON extension and all its hardware would also affect other structures, such as the instruction decoder and the clock tree [12]. Thus, our model very likely represents a pessimistic view of the potential area and power reductions.

Workload and Scheduling: Our workload set uses applications from different sources [13] [14] [15] [16] to form representative use case scenarios for edge computing, as listed in figure 1. We aim to simulate traditional but assorted scenarios. We assume scenarios in which the applications run either completely in parallel, or in a pipeline manner - applications can output partial results to feed the input of the next benchmark. These scenarios are illustrated in table 1, in which the column 'Task' briefly describes the goal of the scenario, column 'Workloads' lists the benchmarks executed, column

Table 1: Workloads in each scenario.

	Task	Workloads	Exec Mode	% NEON
Scenario 1	Image processing	Susan (smooth, edges, corners); 2dconv; histogram; reg-detect; libpng; aes; CRC32; FFT	pipeline	0.5%
Scenario 2	Video encoding	FFT-i; libav-enc; aes; CRC32; FFT	pipeline	3.34%
Scenario 3	Video decoding	FFT-i; aes; h264-dec; CRC32; FFT	pipeline	0.04%
Scenario 4	Health app	FFT-i; ecg; libpng; aes; CRC32; FFT	pipeline	2.55%
Scenario 5	Voice synthesis	rsynth; aes; CRC32; FFT	pipeline	3.48%
Scenario 6	Multitasking	basicmath; bitcount; qsort; stringsearch; 3mm; atax; dynprog; correlation	parallel tasks	8.92%

'Exec Mode' specify if the scenario runs in pipeline or in parallel and '% NEON' shows the percentage of dynamic NEON operations executed.

In scenario 1, we include a series of image filters and kernel operations that represent an image processing application. Scenario 2 and 3 include opensource libraries for encoding and decoding videos, along with kernels that represent data transmission (FFT and FFT-i), criptografy (AES) and redundancy and fault tolerance checks (CRC32)[17]. These latter kernels are also used in scenario 4 - a health app that performs an ECG and uses the opensource libpng library to create an image from the source -, and scenario 5, an app that uses rsynth to generate synthetic voice for user-device conversation. Finally, scenario 6 represents a multitasking environment in which the edge device is receiving tasks from multiple sources. For instance, 3mm and atax are matrix multiplication, transpose and vector multiplication kernels used in graphic processing, and dynamic programming (dynprog) and correlation are commonly used in data analitics.

Most of the chosen applications contain some degree of NEON usage. For instance, the selected kernels (correlation, 3mm, atax) are known for generating vectorized instructions, while the opensource libraries libav and libpng are optimized to use NEON operations. In our scenarios, from the 23 benchmarks used, only the applications bitcount, stringsearch, dynprog, h264-dec, CRC32 and AES do not present NEON instructions, representing common integer-only workloads. Although the selected applications - and their NEON usage - are representative for an assorted edge computing environment, a scalability study, in which we further increase the number of NEON operations executed, will be presented in the section 5.

Finally, to compile our workloads, we have used the gcc arm cross compiler arm-linux-gnueabihf-gcc version 7.3.0 with -O3 optimization flag, which includes flags to generate vectorized instructions. The open source libraries were also configured to use optimizations for NEON.

During our experimentation we consider the reallocation cost that occur each time a workload is selected from the queue and assigned to a core. This cost considers the amount of time necessary to populate the L1 data cache, which is claimed to be the dominant time of task migration [18]. The A15 processors need an average of 12K cycles to fill its data cache, while the A7 requires 17K cycles. This value may be improved, since we are not using any data prefetch technique when migrations are applied. We performed a number of experiments to estimate the cost of emulating NEON instructions. We have run applications with high NEON usage both using

Table 2: Multicore configurations. *PHISA core using emulation

Configuration	A7		A15		Area (mm^2)	Power (W)
	Full	PHISA	Full	PHISA		
A15(4F0P)	0	0	4	0	14.12	2.76
A15(3F1P)	0	0	3	1	11.76	2.66
A15(2F2P)	0	0	2	2	9.41	2.56
A15(1F3P)	0	0	1	3	7.05	2.45
A15(1F0P)	0	0	1	0	3.53	0.69
A15(0F1P)A7(2F0P)	2	0	0	1	2.19	0.69
A15(0F1P)A7(1F1P)	1	1	0	1	2.07	0.69
A15(0F1E)A7(2F0P)	2	0	0	1*	2.19	0.69
A15(0F1E)A7(1F1E)	1	1*	0	1*	2.07	0.69
A15(1F0P)A7(2F0P)	2	0	1	0	4.54	0.80
A15(0F1P)A7(4F0P)	4	0	0	1	3.20	0.80
A15(0F1E)A7(2F2E)	2	2*	0	1*	2.96	0.79

the NEON hardware and using software emulation and found that for each cycle executing in the NEON unit, an average of 40 cycles are required for emulation. We use this data for our emulation scenarios. Again, this is a higher cost than considered in previous works [12].

Experiments: We have built several PHISA configurations using A7 and A15 processors with different ratios of full and partial cores. Table 2 shows all the tested configurations with their area and power characteristics, while table 3 brings a summary of all experiments (which we name Setups) evaluated in this work, with a description, goals, the selected baseline, and the tested configurations. Below we briefly describe all the setups shown in the Results section. In Setup 1 (4.1), we progressively replace full A15 cores by partial-ISA A15 cores to observe the impact of excluding the instruction extension datapaths. The first block of configurations in table 2 shows all the tested scenarios of this experiment, along with their extracted peak power and area, where the configuration names represent the type of cores they implement. For instance, the A15(3F1P) is a 4-Core processor with 3 Full cores and 1 PHISA.

We then build extra PHISA configurations composed of A15 cores without NEON units (partial cores) and full A7 cores in Setup 2 (4.2). The second block in table 2 provides details on them. In this experiment, we compare the PHISA configurations that have the same peak power as a single-, full-ISA A15 core. We also try to extrapolate - aiming to further reduce energy - replacing one of the A7 full cores with a partial core.

Table 3: The experiments and their goals

	Description	Goal	Baseline	Configurations	Section
Setup 1	Homogeneous PHISA organization	Measure the impact of removing ISA extensions from a multicore processor	A15(4F0P)	A15(4F0P); A15(3F1P); A15(2F2P); A15(1F3P)	4.1.1
Setup 2	Heterogeneous PHISA organization with same power budget of single core	Use the extra area and power of removing ISA extensions to create a heterogeneous system	A15(1F0P)	A15(1F0P); A15(0F1P)A7(2F0P); A15(0F1P)A7(1F1P)	4.2.1
Setup 3	Heterogeneous PHISA organization vs DynamIQ-like configuration	Understand which gains are derived from the heterogeneous PHISA organization and which are from the use of big and little cores	A15(1F0P)A7(2F0P)	A15(1F0P)A7(2F0P); A15(0F1P)A7(2F0P); A15(0F1P)A7(1F1P)	4.3.1
Setup 4	Heterogeneous PHISA organization with emulation vs DynamIQ-like configuration	Using emulation to reduce migrations in the PHISA system and amortize the performance loses	A15(1F0P)A7(2F0P)	A15(1F0P)A7(2F0P); A15(0F1E)A7(2F0P); A15(0F1E)A7(1F1E)	4.3.3
Setup 5	Heterogeneous PHISA organization with same power budget of DynamIQ-like	Reestabilish the power budget to compare the DynamIQ-like system with the PHISA multicores	A15(1F0P)A7(2F0P)	A15(1F0P)A7(2F0P); A15(0F1P)A7(4F0P); A15(0F1E)A7(2F2E)	4.3.4

We perform another experiment in Setup 3 (4.3) in which the power and area constraints are lifted to compare the PHISA system against a traditional single-ISA heterogeneous processor - reflecting an ARM DynamIQ configuration. The goal is to understand if the gains observed in Setup 2 were due to the PHISA configuration or because of the heterogeneous environment. We also discuss, in Setup 4 (4.3.3), how these configurations would perform using emulation in the partial-ISA cores before migrating to a full core, without considering any power or area constraints. Finally, we apply the power constraints back to show how a PHISA system of same power as the DynamIQ configuration would perform, both with and without emulation (Setup 5, 4.3.4). These are presented in the third block of configurations in table 2.

The experiments presented in the setups of table 3 were all executed using the simple scheduler without any optimization policies. In subsection 4.4 we analyze the same experiments of setups 2 and 5 (single core and DynamIQ-like baselines against same power budget PHISAs) using scheduling policies to optimize performance and energy.

We also perform a final analysis in section 5, in which we estimate the behavior of PHISA in environments with high NEON usage. We have tested configurations similar to those from the previous experiments to analyze the behavior of Energy-Delay Product (EDP) when hypothetical applications with high usage of NEON instructions are executed.

4. Results

In this section, we present the results for the Setups described in table 3. From subsections 4.1 to 4.3, we present all results using the simple scheduler

described in subsection 2.2. This scheduler is meant to provide the behavior of a PHISA system without necessarily optimizing it for any particular requirement. Later, in subsection 4.4, we introduce new results for some of the most interesting Setups, but using an advanced scheduler with policies to optimize for either performance or energy consumption, as described in subsection 2.3.

4.1. Impact of Partial ISA Cores

4.1.1. Experiment discussion:

In this experiment, we measure the impact of implementing partial ISA cores in a multicore environment. The goal is to evaluate if removing cores capable of executing NEON operations would impact the performance and energy of the system, and in which ratio (full:partial) this impact would become relevant. This is done - as described in table 3 as setup 1 - by progressively replacing full A15 ISA cores by partial A15 ISA ones in systems with 4 cores. The first block of configurations in table 2 shows how area and power behave in the modeled systems of this experiment, with an expressive reduction in area and a smaller, but considerable, decrease in power for configurations that comprise partial ISA cores. For example, when aggressively replacing full cores in the quad core processor, the area is reduced by 50%, while power decreases by 11%.

Given the expected area and power decrease, we now analyze how they influence performance and energy consumption. We have executed all the six scenarios in table 1 in all systems from the first block in table 2. Figure 6 shows results for this experiment. The x-axis contains the different A15 multicore versions, separated by each evaluated application scenario. The y-axis shows the normalized number of cycles (the lower the number of cycles, the better is the **performance**), energy and EDP with respect to the A15(4F0P) configuration, which represents a traditional full-ISA multicore processor. For all the metrics, the lower the bar, the better. As the figure 6 shows, for most of the scenarios, the number of cycles increases as we include more partial cores, which is expected, as partial cores will need to migrate tasks that require NEON instructions. Energy, on the other hand, remains almost constant in most cases, due to the power reductions of the partial cores.

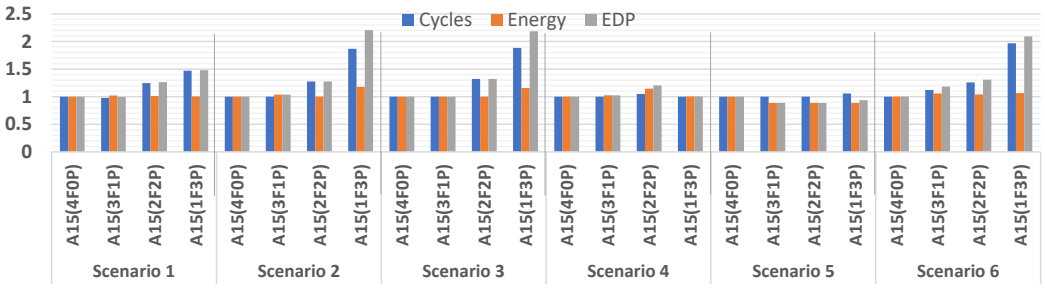


Figure 6: A15 Full cores being progressively replaced by their partial ISA counterparts considering each application scenario. Performance, Energy and EDP are normalized to the A15(4F0P) configuration

4.1.2. Observations from this experiment:

Two observations should be highlighted at this point. (i) Although the cycle count increase is significant in the tested scenarios, this increase is much smaller when the proportion of full cores is high. For example, in the configuration with 75% of full ISA cores and 25% partial ISA cores (the A15(3F1P)), the increase in cycle counts is only relevant in scenario 6 (about 10%). In other words, *partial cores can be introduced in the system as long as we provide enough full cores for NEON execution.* (ii) As seen in Table 4, the full ISA A15 processor power is about 14x higher than the full A7 and occupies about 7x more area. A single partial A15 ISA core has 66% less area from a full A15 core, and 15% of less power. The freed area represents 4 times the area of a full A7, while the freed power is approximately the same as 2 full A7 cores. Thus, *extra A7 cores (which may be full or partial) can be introduced in the freed area of the system, while still respecting a power budget.* Next, we explore the trade-off between replacing full A15 by partial A15 cores - which consequently decreases performance - and adding A7 cores to recover some of the performance and increase energy efficiency.

Table 4: Area and power of full and partial A15 and A7 processors.

	A15		A7	
	Full	PHISA	Full	PHISA
Area(mm^2)	3.53	1.17	0.5	0.38
Power(W)	0.69	0.59	0.05	0.046

4.2. Full Core vs PHISA Multicore - Sharing a Power Budget

According to the discussion provided by the previous subsection, let us consider that most IoT systems are battery-powered and it is necessary to limit their designs to a particular peak power supplied by their batteries. Thus, we establish a power budget for our system and use the extra area and power provided by PHISA multicores to retrieve the lost performance (due to fewer NEON units) using extra A7 cores. For that, we create heterogeneous multicore configurations (in organization and ISA) that fit in the same area and power budgets of a traditional single-core processor. For that, we have built Setup 2 as described in table 3, with configurations A15(0F1P)A7(1F1P) and A15(0F1P)A7(1F1P), which have approximately the same peak power as the traditional single-core A15 processor (A15(1F0P), as shown in table 2).

4.2.1. Experiment discussion:

Figure 7 shows the performance and energy consumption of the PHISA configurations A15(0F1P) A7(0F2P), A15(0F1P) A7(1F1P) and the traditional single-core A15(1F0P). PHISA multicores can significantly decrease energy consumption while also improving performance, as long as enough full cores are provided. A15(0F1P) A7(2F0P) reduces energy by 3.11x while improving performance by 1.94x in scenario 1 (Image Processing) when compared to the baseline. Similar results are observed in the other scenarios. The performance increase is mainly attributed to the extra cores present in the system, which can execute more workloads in parallel. As the workloads are independent (apart from their pipelined behavior), the scheduler can easily distribute the applications between cores, so more cores result in more performance. On the other hand, energy is reduced by the use of much less power hungry cores. Not only the partial ISA A15 cores have reduced power when compared to the traditional design, but the full A7 processors added to the system are also much more efficient. The exception is when execution scenario 6 (Mathematics & Algorithms), which is also the scenario that uses NEON operations the most. In this scenario, the pressure on the full cores (A7 cores) is much higher, thus the performance drop.

4.2.2. About the cores usage:

Figure 8 shows the usage of the cores in configuration A15(0F1P) A7(2F0P) running scenario 6. In the figure, Core0 is the partial A15 core, and the others are the full A7. The figure shows how the big cores are idle (low step) during

a great part of the execution. The "solid bar" in Core0 are constant changes in the core state, which happen when the core has no more tasks to run (idle), receives a task (active) and the task fetches a NEON instruction and migrates again (idle). Such periods of inactivity are usually of 160K cycles, which is the period of migration in the full cores. This smaller load in the big cores greatly affects the energy consumption, as the A7 cores have a peak power nearly 10x lower than the A15. Thus, although the maximum peak power of the PHISA is the same as the baseline, the dynamic peak power of the system is smaller because the workload is not fully concentrated in the big cores.

4.2.3. Introducing partial-ISA A7:

Extrapolating further the reduction of power using partial cores, configuration A15(0F1P)A7(1F1P) - also in figure 7 - replaces a full A7 core with its partial version, leaving the configuration with only one full core. The energy consumption shows a small decrease in the scenarios, but the extra pressure in the only full core in the system causes a high increase in the number of cycles, which in turn, prevents higher energy reductions. This is consistent with the experiments in section 4.1.1, in which the ratio of full cores should be bigger than partial to maintain performance. In general, the trade-off between performance and energy consumption - the EDP - is worse in configuration A15(0F1P)A7(1F1P), mainly because of the poor performance of such a system. Scenarios 1 and 6 show a huge reduction in energy, and, controversially, a high increase in the number of cycles. This is because both of these scenarios include many applications that execute NEON instructions and compete for the only full core, an A7 core. As most of the

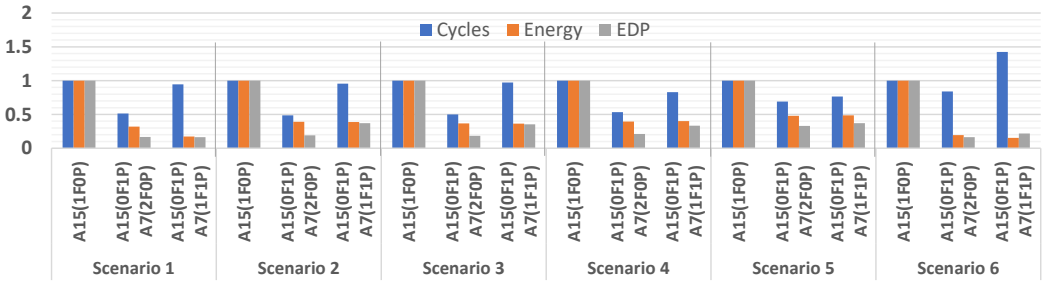


Figure 7: Evaluation PHISA multicores against a homogeneous baseline under a 700mW power budget. Performance, Energy and EDP are normalized to the A15(1F0P) configuration

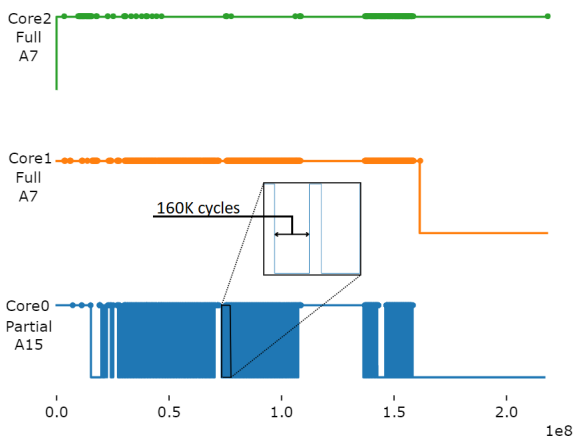


Figure 8: Core usage in configuration A15(0F1P)A7(2F0P) running scenario 6. High step means the core is in usage, low step is idle. Solid bars are constant idle-active changes. Dots represent migrations.

execution happens in the A7 core, which is extremely energy efficient, the energy consumption falls, but the cycle count increase. Other scenarios have higher NEON usage than scenario 1, however, in these scenarios, the NEON operations are concentrated in fewer applications. For instance, in scenario 5 the NEON operations are only required by rsynth and FFT, which makes it simpler for the scheduler to manage the full A7 resources. In general, the power reduction from replacing a full A7 core with a partial version is too small, and do not show enough advantages as is the case of the bigger A15 cores.

In this experiment, we have seen that it is possible to create heterogeneous systems with PHISA and have better energy consumption than power equivalent full processors. Nonetheless, if the power budget is not considered, this is as expected from all heterogeneous processors. In the next section, we show through setup 3 which are the real gains provided by the PHISA system.

4.3. PHISA vs Traditional Heterogeneous Systems (DynamIQ)

4.3.1. Experiment discussion:

Heterogeneous processors naturally deliver better energy efficiency than homogeneous multicores. To understand which gains are derived from the usage of PHISA and which are simply from having additional cores, we now

evaluate configuration A15(1F0P)A7(2F0P), which represents a DynamIQ heterogeneous processor in figure 9. Nonetheless, it is important to highlight that, in this configuration, **the power and area budgets are completely ignored**. Configuration A15(1F0P)A7(2F0P) is much bigger (more than twice the size) and has higher power (about 14%) than the PHISA equivalents of same core count. Thus, it is expected that the PHISA system will be worse in performance in this scenario, as the DynamIQ has much more resources to use.

As can be seen in the figure 9, the cycle count of the configurations with partial cores is higher than those from the DynamIQ configuration, which was expected, as the full core configuration does not require migrations to execute NEON instructions. On the other hand, energy consumption in the PHISA systems is usually lower, showing that partial ISA cores are essential to decrease energy. In fact, the EDP of the PHISA system is usually lower, showing that the partial cores can deliver better trade-off between performance and energy consumption. Energy consumption in the PHISA configurations is reduced due to the partial A15 cores. The original A15 processor has a peak power 15% higher than its counterpart partial ISA version. Besides, since in the PHISA configuration the A15 cannot execute NEON instructions, the scheduler must migrate the workloads from the power-hungry A15s to the efficient A7s more frequently than in the traditional system. Effectively, these migrations increase the usage of the A7 cores, leaving the A15 idler and reducing energy consumption.

The DynamIQ configuration, on the other hand, tends to use the A15 core more often to increase performance, which comes at the price of energy. If the scheduler of the DynamIQ were to be changed to optimize energy - and use

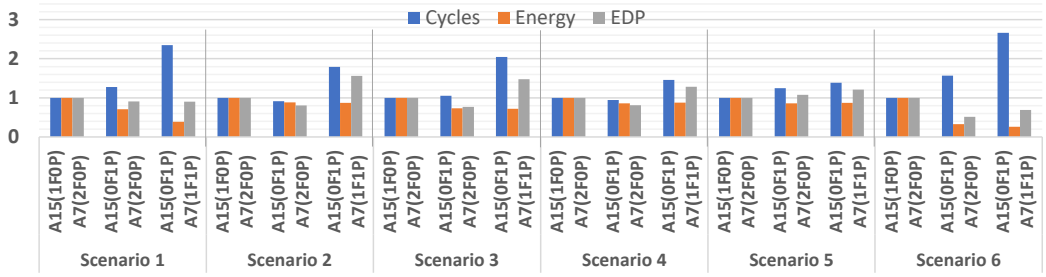


Figure 9: Evaluation of PHISA multicores against a DynamIQ baseline. The baseline has the same amount of cores as the PHISA configurations, thus there is no power/area budget. Performance, Energy and EDP are normalized to the A15(1F0P)A7(2F0P) configuration

the A7 cores at the same ratio as PHISA - it would still consume more energy, as the full A15 core dissipates more power than the partial-ISA version. This balance is clearly seen with configuration A15(0F1P)A7(1F1P) in scenario 6, which frequently requires the NEON unit. There is only one full A7 in the system and it has to execute all the NEON requests from the workloads. This pressure increases the time required to execute all applications, but also reduces energy consumption, as the A7 is much more efficient than the A15.

4.3.2. About task migration:

Scheduling in PHISA multicores is tied to the usage of NEON instructions by a workload. In figure 8 we have seen that the full A7 cores are constantly receiving tasks by the scheduler and this is due to two reasons: (i) every time a workload migrates to a full core, it must stay for a minimum amount of cycles (160K in this work); and (ii) when a partial A15 core fetches a NEON instruction, it must migrate the job to an A7. In figure 8, to avoid constant migration, the simple scheduler we used (from subsection 2.2) will prioritize NEON applications to full cores: if a full core migrates an application and there are two possible targets (same size of workload queue), being one full and the other a partial core, it will schedule the application to the full core. This helps to avoid constant back and forth migrations from the partial cores, as they will be assigned more integer workloads. However, this does not completely remove the problem, as when the full cores are all busy, and the partial ones are free, workloads will be assigned to the partial cores, independently of the type of instruction they hold. This can be seen, as already mentioned, in the Core 0 of figure 8 as a "solid bar." This event is frequently observed in scenario 6 that is composed of many applications with NEON usage.

These excessive migrations are also the reason for the high increases in the cycle counts of PHISA systems observed in the experiment of figure 9. To mitigate this problem and reduce the number of migrations we have implemented the ability to emulate NEON instructions in the partial cores. We have established a threshold time in which the partial cores will emulate NEON instructions before migrating the task to a full core. In other words, every time a NEON instruction is fetched by a partial core, the scheduler will decide whether the instruction should be emulated in software or migrate to a capable core: if the workload has already been executed for more than the threshold time in that core, it migrates. Otherwise, it emulates the instruction. For the sake of compatibility of the migration times, we have set

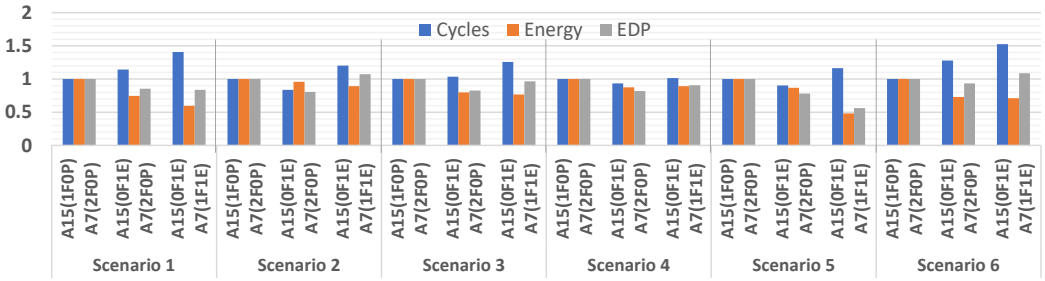


Figure 10: Evaluation of PHISA multicores allowing emulation against a DynamIQ baseline. The baseline has the same amount of cores as the PHISA configurations, thus there is no power/area budget. Performance, Energy and EDP are normalized to the A15(1F0P)A7(2F0P) configuration

this threshold to 160K cycles, the same time as the original migration event for full cores.

4.3.3. The impact of emulation:

Figure 10 shows the results for configurations A15(0F1E)A7(2F0P) and A15(0F1E)A7(1F1E) compared to the DynamIQ-like configuration. In this processor, the 'E' in the name means a partial-ISA core that can emulate NEON instructions in software. As the figure shows, the emulation strategy can amortize some of the impacts in cycle counts caused by the partial cores. In some scenarios, such as 2, 4 and 5, the cycle count is even smaller than the baseline, due to the balancing of workloads in the cores. The energy, on the other hand, increases when compared to the non-emulation scenario, as now the partial A15 cores are used more frequently. When one considers the EDP, the PHISA systems are better than the baseline in almost all cases. Nonetheless, it is important to remember that, in this case, the original DynamIQ-like configuration is 2x bigger and has 14% higher peak power than the PHISA configurations (table 2).

4.3.4. PHISA vs DynamIQ with Power Parity:

When one considers a power budget parity between the DynamIQ and the PHISA configurations, it is possible to add two extra A7 cores in the system. This parity is represented in configuration A15(0F1P)A7(4F0P) and a version with more partial cores, but with emulation, A15(0F0E)A7(2F2P) in table 2. Figure 11 shows the results for running the scenarios in these configurations. In all the scenarios, the PHISA processors have better performance, energy

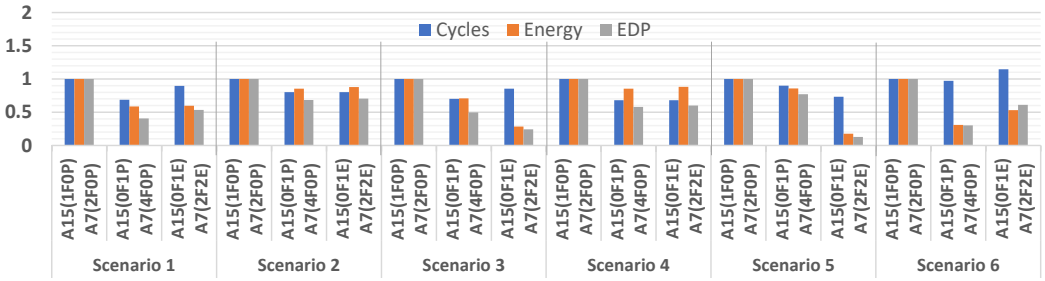


Figure 11: Evaluation of PHISA multicores with and without emulation against a DynamIQ baseline under a 800mW power budget. Performance, Energy and EDP are normalized to the A15(1F0P)A7(2F0P) configuration.

and, consequentially, EDP than the completely full-ISA processor. The best performance improvement is observed in scenario 4 (Health app) with 32% reduction in cycle count. This scenario presents applications that contain high NEON usage (ECG), but that are very fast to execute, creating a perfect scenario for the extra A7 cores. The best energy consumption is observed in scenario 5 (Voice synthesis) - with 82% reduction -, which is composed of only two NEON applications, which can execute in the two full cores of the system, while the other applications are executing in the (more energy efficient) partial-ISA cores. Furthermore, as shown in table 2, the PHISA configurations are still smaller, in area, than the DynamIQ-like processor. Thus, the PHISA designs present as an opportunity to create systems that are smaller and more energy efficient than the current industry trend.

4.4. Advanced Scheduler and Policies Impact

4.4.1. Scheduling for Performance

As previously discussed, we have redesigned the initial simple scheduler to make a best effort to optimize the system for different goals. In the performance policy, the scheduler always gives priority to allocate tasks in the big OoO cores (A15), assuming that this core will execute the application faster than the small in-order cores (A7). Although this prioritization can improve system performance, it might not be the best strategy if one expects higher energy efficiency.

In this experiment, we have simulated the same scenarios from table 1, comparing the the configurations from setup 2 (heterogeneous PHISA against single-core baseline with same TDP) and setup 4 without emulation (heterogeneous PHISA against traditional DynamIQ-like system with same area

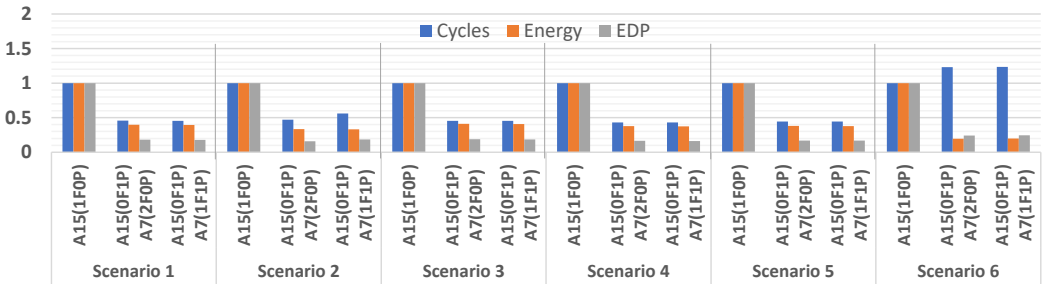


Figure 12: Evaluation PHISA multicores against a single-core baseline under a 700mW power budget. Scheduling of tasks follows a performance optimization policy for all configurations, including the baseline. Performance, Energy and EDP are normalized to the A15(1F0P) configuration.

TDP). Our goal is to evaluate how the performance policy affects the same scenarios already evaluated using the naive approach.

Figure 12 shows the performance, energy, and EDP of the PHISA configurations using the performance policy normalized by the single-core A15 processor. The figure shows that the PHISA configurations have better performance and energy consumption in all the scenarios. When compared to the naive scheduler, the performance is improved in every scenario for almost every configuration. The PHISA configurations with only one full core show even further improvements when compared to the naive scheduler. This is not only because of the policy itself but also because of the improvements in task preemption and workload queues that this new scheduler introduces. Again, scenario 6 is the only one in which the PHISA systems show worse performance than the baseline, and this is because the scenario was built with high NEON usage in mind. The scheduler will prioritize the partial A15 for performance, but will constantly have to migrate to workloads to A7 cores because of the high occurrence of SIMD and FP instructions.

Figure 13 shows the performance, energy and EDP of the PHISA configurations using the performance policy normalized by a traditional DynamIQ-like configuration. Again, the PHISA configurations show both better performance and energy consumption in all scenarios. What is mostly interesting in this evaluation is that the differences between the results in the PHISA systems (with one or two full cores) are very small (close to 2% only).

Finally, figure 14 shows the scheduler behaviour of the traditional DynamIQ configuration A15(1F0P) A7(2F0P) while executing the applications in scenario 1. Figure 15 shows the behaviour in the same scenario for the

PHISA A15(0F2P) A7(4F0P) configuration. As shown in the figure 15, the PHISA configuration has more cores to execute the multiple applications, increasing the throughput of the scenario. Migrations in the traditional DynamIQ happens only during preemption phases, while in the PHISA the applications have to change cores whenever a non-implemented function is fetched in a partial core. This is better observed in the final execution of the *FFT* application, as in figure 14 it is completely run in the big full core, while in figure 15 it has to migrate from the big partial to the little full several times.

4.4.2. Scheduling for Energy

In this experiment, we prepare the same configurations and scenarios from subsection 4.4.1, but change the scheduler policy to optimize energy consumption. This policy prioritizes the allocation of tasks in the little cores, assuming that these will be more energy efficient. It is important to notice that in these scenarios, all the configurations use the energy consumption optimization policy, including the baseline.

Figure 16 shows the results for this experiments. The PHISA configurations using this policy are able to reduce the energy consumption further when compared to the results in subsection 4.4.1. Performance is also improved in relation to the baseline, as the baseline is also running the energy policy.

Figure 17 shows the same experiment for the PHISA configuration with same TDP as a traditional DynamIQ. Again, PHISA is able to further reduce the energy consumption, performance, and - consequently - their trade-off in

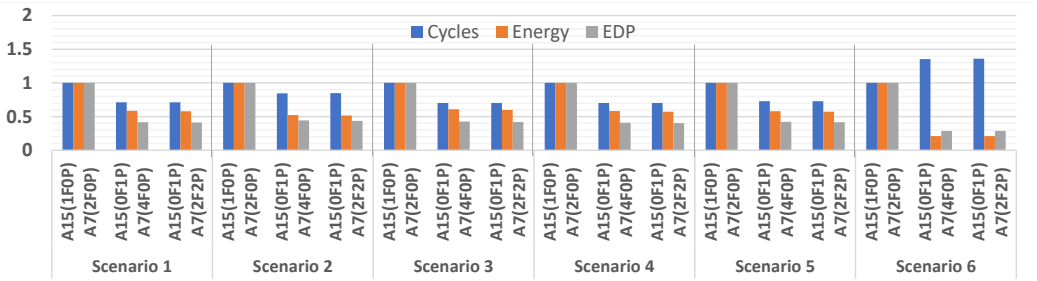


Figure 13: Evaluation of PHISA multicores against a DynamIQ baseline under a 800mW power budget. Scheduling of tasks follows a performance optimization policy for all configurations, including the baseline. Performance, Energy and EDP are normalized to the A15(1F0P)A7(2F0P) configuration.

the form of EDP.

Finally, figures 18 and 19 show the scheduling behaviour of configurations A15(1F0P)A7(2F0P) (DynamIQ) and A15(0F1P)A7(4F0P) (PHISA) using the energy policy respectively. Once more, the PHISA configuration can deliver more throughput than the DynamIQ. However, as the full little cores are prioritized due to their energy efficiency, the number of migrations is reduced. This is seen in the final execution of the *FFT* application, in which in both figures, it finishes executing in one of the full little cores (in contrast with the performance policy).

5. Analysis of PHISA on High NEON Usage

Our experiments have been using scenarios with some of the single-threaded workloads presented in figure 1. Although the selected set of workloads covers a wide range of applications from the embedded system and IoT market, one may question the behavior of the system when exposed to higher amounts of NEON instructions. Considering that the number of instructions from removed extensions will directly influence the behavior of a PHISA multicore, we now use an analytic model of hypothetical applications, in which

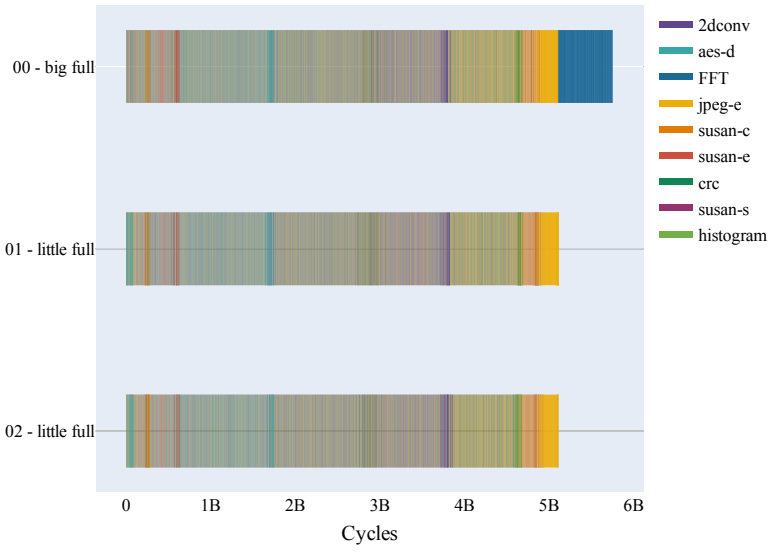


Figure 14: Scheduler migrations using the performance policy in the traditional DynamIQ configuration A15(1F0P)A7(2F0P) during execution of scenario 1. Bars represent each application being run over time in each processor core.

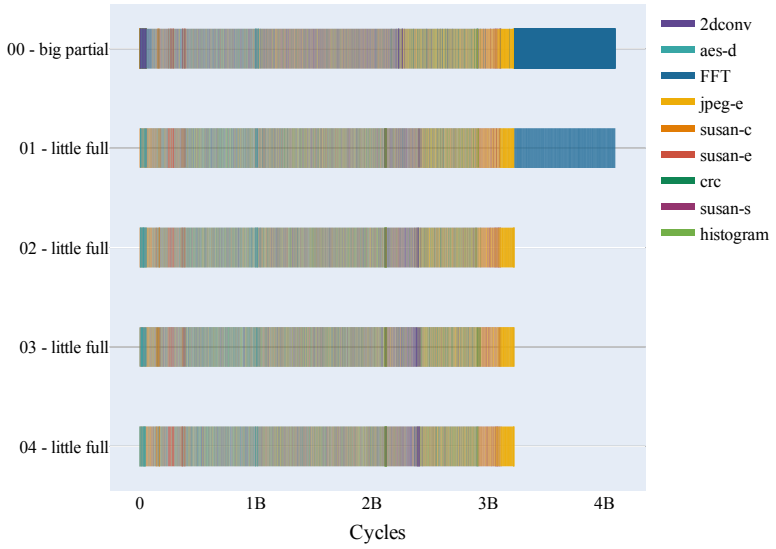


Figure 15: Scheduler migrations using the performance policy in the PHISA configuration A15(0F2P)A7(4F0P) during execution of scenario 1. Bars represent each application being run over time in each processor core.

we can vary the number of issued NEON instructions, as shown in figure 20. The goal is to observe how the different PHISA configurations scale with the number of NEON instructions compared to a traditional full-ISA system.

In this new environment, we assume configurations from the previous experiments, in which the **partial cores only execute integer operations, and the full cores execute both integer and NEON operations**. In

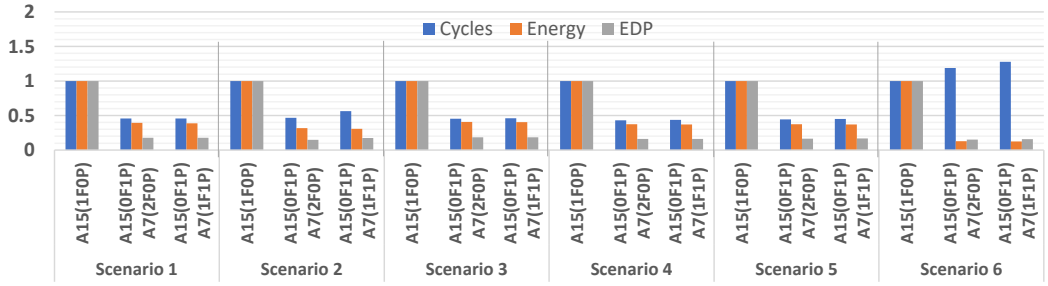


Figure 16: Evaluation PHISA multicores against a single-core baseline under a 700mW power budget. Scheduling of tasks follows a energy consumption optimization policy for all configurations, including the baseline. Performance, Energy and EDP are normalized to the A15(1F0P) configuration.

the configuration A15(0F1E)A7(2F0P) the partial-ISA A15 core can also emulate NEON instructions. As a best-case comparison, we have selected configuration A15(1F0P)A7(2F0P), which represents a similar processor, but with all full-ISA cores.

We assume there is a migration cost (12K for the A15 and 17K for the A7, the same as in previous configurations) and that the number of migrations

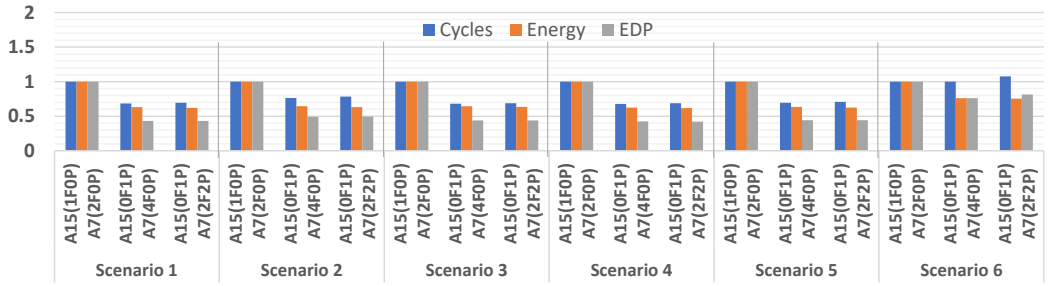


Figure 17: Evaluation of PHISA multicores against a DynamIQ baseline under a 800mW power budget. Scheduling of tasks follows a energy consumption optimization policy for all configurations, including the baseline Performance, Energy and EDP are normalized to the A15(1F0P)A7(2F0P) configuration.

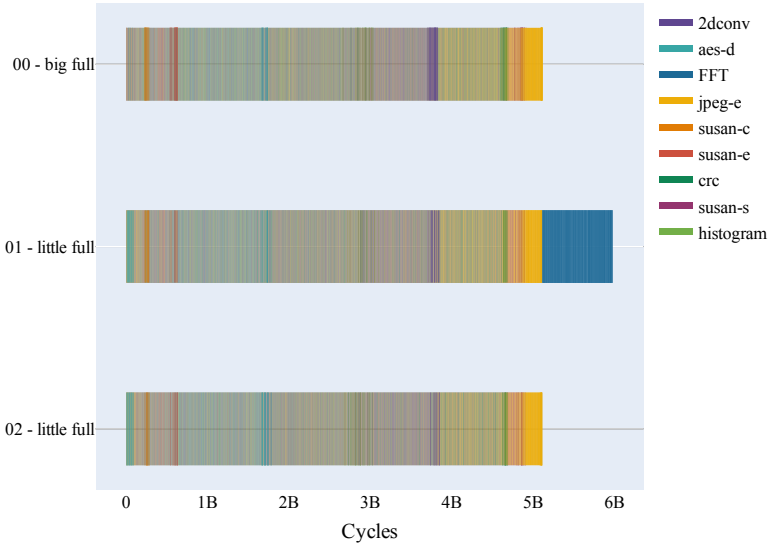


Figure 18: Scheduler migrations using the energy policy in the traditional DynamIQ configuration A15(1F0P)A7(2F0P) during execution of scenario 1. Bars represent each application being run over time in each processor core.

increases proportionally to the ratio between NEON and integer instructions: the higher is the ratio, the higher are chances of these instructions being interleaved, causing multiple migrations. This cost rises until 50% of NEON instructions, and from 60% forward, the cost decreases as the ratio inverts, and there is a lower chance of interleaved operations. For example, when the application has 10% NEON instructions, there will be nine integer instructions for one NEON instruction, which would cause one migration. For 50% NEON instructions, there will be five integer instructions for each five NEON, which can be interleaved (1 int, 1 NEON, 1 int, 1 NEON...), and would cause five migrations. However, *this scenario will reverse* if there are more NEON instructions than integers. We also assume that a NEON instruction takes twice as many cycles to execute than an integer instruction in the A15 core and eight times more in the A7 core. These are average numbers estimated from simulations.

Figure 20 shows the behavior of the EDP of the configurations as the number of NEON instructions in the application increase. As the figure shows, configuration A15(0F1P)A7(2F0P) has good scalability, which is tied to its types of cores. As the number of NEON instruction increases, the par-

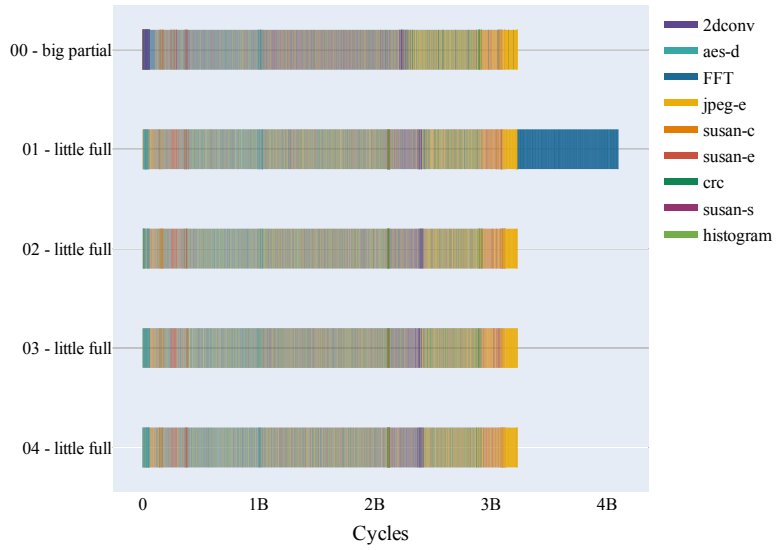


Figure 19: Scheduler migrations using the energy policy in the PHISA configuration A15(0F2P)A7(4F0P) during execution of scenario 1. Bars represent each application being run over time in each processor core.

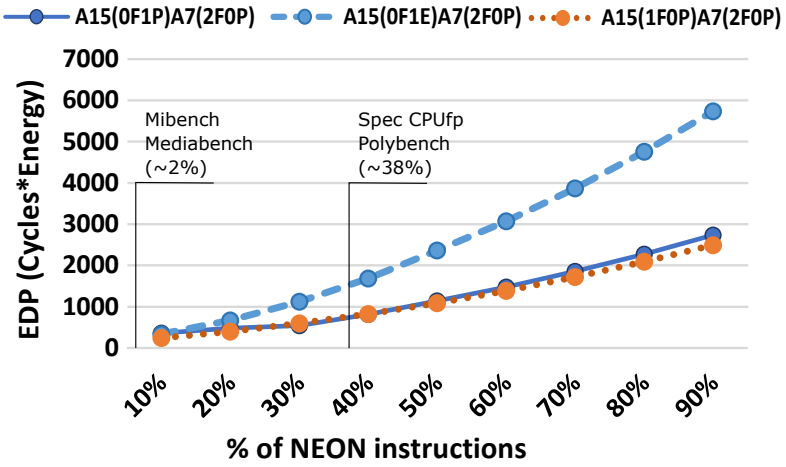


Figure 20: Behaviour on high NEON usage PHISA multicore (with and without emulation) and DynamIQ.

tial A15 will become idle more often. Although this becomes a burden for the processor performance, migrating the load to the A7 processors greatly reduces the system dynamic peak power, which decreases the energy consumption. This allows the PHISA configuration to stay very close in EDP to its full-ISA counterpart. It is important to notice that the configuration with full cores has more than twice the area and higher peak power than the PHISA version.

On the other hand, the same PHISA configuration but with emulation capacity A15(0F1E)A7(2F0P) shows bad scalability in higher rates of NEON. When executing few NEON instructions, the emulation has good performance, however as the NEON instructions increase, the A15 processors will be assigned to emulate more of these instructions, which will incur in high-performance overhead. Furthermore, the A15 is a power-hungry core, which will also increase consumption.

This experiment demonstrates that the PHISA multicore has potential even when the ratio of NEON instructions increases, as its expected EDP stays close to that of a full-ISA system. In fact, the difference in EDP from the full-ISA processor and the PHISA system in a modeled application with 90% of NEON instructions is of less than 10%. For both applications with low NEON usage (such as in the Mediabench and Mibench suites) and for high NEON usage applications (such as SPEC CPUfp and Polybench), the PHISA system can have similar scalability as the traditional DynamIQ-

like heterogeneous processor. For low NEON usage, such as in Mibench applications, emulation is also a good choice to balance workloads between cores.

6. Related Work

In this section we will present state-of-the-art works related to this paper. We will discuss works that have explored single-ISA heterogeneous processors, as well as researches on the impact of the different ISAs in a system. We also present works that have explored the concept of partial (or overlapping) ISAs, both by the software (and scheduling) and hardware sides. Finally, we summarize the novelty of our work when compared to the state-of-the-art.

Single-ISA heterogeneous processors have been proposed by Kumar et al. [4][19] as an alternative to power efficiency. The authors show how a mix of in-order and OoO Alpha processors with different issue widths can be used to adapt the system power usage accordingly to the application requirements. One of the main advantages of this technique is that it is transparent to the application and does not require special tools to deal with many different ISAs, as in the case of accelerator rich processors or MPSoC [20]. On this heterogeneous environment, a runtime system manager - such as the Operating System - can identify the resource requirements of the applications and schedule threads to cores that fulfill these requirements while minimizing energy consumption. As the entire system uses the same ISA, threads can easily migrate between cores using a shared memory space, as in traditional multicore processors. This approach has been used by the industry, leading to technologies such as the ARM big.LITTLE[5], which allows the use of clusters of A15 (big) and A7 (LITTLE) cores, and, more recently, the ARM DynamIQ[6]. In DynamIQ, cores from different sizes can be mixed in the same cluster and share a coherent cache for fast thread migration, allowing for more diverse design space exploration. Nonetheless, there are many works employing different strategies to implement heterogeneous processors of single-ISA [21], from using DVFS to reach performance asymmetry in different voltage domains [22] to using binary translation to keep transparency between cores and accelerators [23].

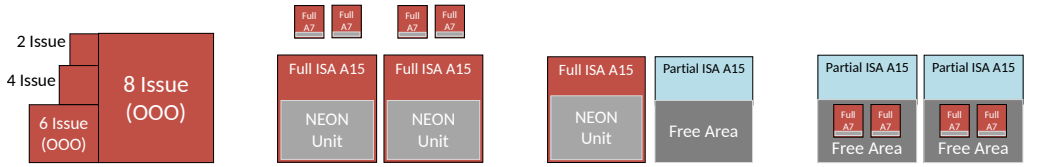
On the other hand, some current works evaluate the impact of the ISA in the microarchitecture. The work of Venkat and Tullsen[24][25] show that a system can exploit the many traits of different ISA to improve the effectiveness of heterogeneous processors. The authors combine cores of three ISAs

(32 bit Thumb, X86-64 and Alpha 64 bits) in a single system, classifying their performance according to aspects such as FP and SIMD operations, register pressure, code density, and dynamic instruction count. The results show that most applications have phases in which different ISAs would perform better, suggesting that a diverse ISA environment is more efficient than a single-ISA one. In [26] and [27], Blem et al. discuss how the RISC and CISC models affect modern architectures. By the combination of [24] and [27], Blem concludes that what affects power and performance in modern systems is the presence or absence of specialized ISA extensions (such as vectorization, FP, crypto instructions) that may appear in some ISAs, but not in others. In [28], Lopes et al. perform an extensive analysis of ISA aging and the cost of the decoder for keeping old operations in the architecture set. 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.

These mentioned works show that a diverse and renewed ISA is essential for processor performance, although most of the added instructions are used for specific applications. To try to balance instruction extensions between cores, overlapping-ISAs have been used on previous works. These are processors in which the cores individually implement different extensions, but all share a base ISA. In [7], Li et al. discuss the challenges of implementing Operating System (OS) support for overlapping ISAs and propose a fair scheduling algorithm for these systems. One of the major contributions of [7] is arguably the mechanism that allows for detection and migration of instructions that cannot be executed in a partial-ISA core, named Fault-and-Migrate. In [29], Reddy et al. present design techniques for bridging software to functionally asymmetric cores and discuss the pros and cons of each method.

On a microarchitectural level, Lee et al. [12] study the usage of different ISA extensions in an ARM processor and propose a system with a reduced- and a full-ISA core, both based on the ARM A15. The reduced A15 is deprived of the NEON, predicate, DSP and load/store multiple instructions, which makes it more power and area efficient due to the simpler datapath. As the goal is to optimize energy consumption, the reduced core is given full priority to execute the applications. On the other hand, the full core is assigned whenever a specialized instruction is fetched, at the cost of migrating the task and spending extra power.

Another similar strategy adopted by the industry is to share resource ex-



(a) Single-ISA heterogeneous multicore, as proposed by Kumar et al. [4]

(b) big.LITTLE/DynamIQ technology by ARM [5]

(c) Heterogeneous partial-ISA multicore, as proposed by Lee et al. [12]

(d) Our proposal: heterogeneous ISA and microarchitecture, exploiting partial-ISA area to increase the system throughput

Figure 21: Different approaches for multicore and our proposed system.

pensive processing units between more than a single core. The UltraSPARC T1 [30] was a SUN multicore processor that shared a single FPU between eight integer cores, while the AMD Bulldozer core presented one FPU for each two integer clusters, which were used as in a simplified Simultaneous Multithreading (SMT) approach.

Our Contribution: Figure 21 illustrates the most significant approaches discussed in this section. Our solution (figure 21d) exploits a new level of heterogeneity that has not been considered in any of the previous works: a multicore partial-ISA implementation coupled with performance asymmetric cores. We bridge the gap between the DynamIQ technology and the partial-ISA design, demonstrating how a processor can trade parts of its specialized units (which would provide single-task performance) to increase the core count (improve multi-task performance) and be more EDP efficient without increasing the power and area of traditional processors.

More specifically, [7] and [29] focus on OS and software support (scheduler and framework), while [12] investigates some of the microarchitectural impacts of a partial ISA implementation, but in a dual-processor/single task of different ISA extensions aprroach, and considering the execution of only one application at a time (i.e., either the reduced or the full core is active at a given moment). On the other hand, we propose a new environment that gracefully merges the heterogeneity of the micro-architectures that compose the system with the artificially imposed restrictions to their ISAs to improve over real designs in terms of performance, energy or EDP. This is only achieved with extra micro architectural changes and support to concurrent

multitasking.

Other works share some specialized units, such as the the UltraSPARC T1. However, its single FPU and loosely integration with the cores make floating point operations too costly for this system. Another example is the AMD Bulldozer, whose approach improve resource usage in the core, but may also be inefficient in single-threaded tasks, as only one integer/FPU cluster could be used per thread. Our solution avoids these problematic scenarios by implementing full-ISA cores that can execute both specialized and common control and integer instructions, thus that are no extra costs in executing applications that interleave these operations. The overhead of a PHISA multicore is associated only with the migration costs of the tasks and the amount of instructions competing for resources in the applications. In other words, we move the overhead from the instruction to the task level, which may be higher but has lower occurrence.

7. Conclusions

In this paper, we have proposed PHISA multicores as a mean to reduce area and power from multicore systems, by removing ISA extensions that are not constantly used. We show that it is possible to use partial ISA cores in edge computing systems, without incurring into large performance impacts. Furthermore, we show how the extra area and power can be used to add more cores to the multicore system and further increase performance and decrease energy consumption. When coupled with a specialized scheduler, our system is also able to further optimize its behavior for a specific goal, such as energy consumption or performance. Our approach can scale well, even if the applications issued to the multicore have high amounts of NEON instructions.

Acknowledgement

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, Brasil (CAPES) - Finance Code 001, the Fundação de Amparo à Pesquisa do Estado do RS (FAPERGS) and the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).

References

[1] L. Mainetti, L. Patrono, A. Vilei, Evolution of wireless sensor networks towards the internet of things: A survey, in: SoftCOM 2011, 19th International Conference on Software, Telecommunications and Computer Networks, 2011.

[2] Exynos 7 Dual 7270 Processor: Specs, Features — Samsung Exynos, https://www.samsung.com/semiconductor/minisite/exynos/products/mobileprocessor/exynos-7-dual-7270/ (2019).

[3] W. Shi, S. Dustdar, The Promise of Edge Computing, Computer 49 (5) (2016) 78–81. doi:10.1109/MC.2016.145.

[4] R. Kumar, et al., Single-ISA heterogeneous multi-core architectures: the potential for processor power reduction, MICRO-36. (2003).

[5] big.LITTLE, https://developer.arm.com/technologies/big-little (2019).

[6] DynamIQ, https://developer.arm.com/technologies/dynamiq (2019).

[7] T. Li, et al., Operating system support for overlapping-ISA heterogeneous multi-core architectures, HPCA'10 (2010) 1–12.

[8] T. Constantinou, et al., Performance implications of single thread migration on a chip multi-core, SIGARCH Comput. Archit. News (2005).

[9] N. Binkert, et al., The gem5 simulator, ACM SIGARCH Computer Architecture News 39 (2) (2011) 1. doi:10.1145/2024716.2024718. URL http://dl.acm.org/citation.cfm?doid=2024716.2024718

[10] S. Li, et al., McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures, in: MICRO 42: Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture, 2009.

[11] F. Endo, et al., Micro-architectural simulation of embedded core heterogeneity with gem5 and McPAT, RAPIDO '15 (2015) 1–6.

[12] W. Lee, et al., Exploring Heterogeneous-ISA Core Architectures for High-Performance and Energy-Efficient Mobile SoCs, GLSVLSI'17 (2017) 419–422.

[13] M. Guthaus, et al., MiBench: A free, commercially representative embedded benchmark suite, in: IEEE WWC-4'01, IEEE, 2001, pp. 3–14.

[14] J. E. Fritts, et al., Mediabench ii video: Expediting the next generation of video systems research, Microprocess. Microsyst. 33 (4) (2009) 301–318.

[15] L.-N. Pouchet, PolyBench/C – The Polyhedral Benchmark suite, http://web.cse.ohio-state.edu/~pouchet.2/software/polybench/ (2019).

[16] C. Tan, et al., Locus: Low-power customizable many-core architecture for wearables, ACM Trans. Embed. Comput. Syst. 17 (1) (Nov. 2017).

[17] T. Adegbija, et al., Microprocessor Optimizations for the Internet of Things: A Survey, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 37 (1) (2018) 7–20.

[18] T. Li, D. Baumberger, D. A. Koufaty, S. Hahn, Efficient operating system scheduling for performance-asymmetric multi-core architectures, in: ACM/IEEE Conference on Supercomputing, 2007. doi:10.1145/1362622.1362694.

[19] R. Kumar, D. Tullsen, P. Ranganathan, N. Jouppi, K. Farkas, Single-ISA heterogeneous multi-core architectures for multithreaded workload performance, in: Proceedings. 31st Annual International Symposium on Computer Architecture., IEEE, 2004, pp. 64–75. doi:10.1109/ISCA.2004.1310764.
URL http://ieeexplore.ieee.org/document/1310764/

[20] W. Wolf, A. A. Jerraya, G. Martin, Multiprocessor system-on-chip (MPSoC) technology, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 27 (10) (2008) 1701–1713. doi:10.1109/TCAD.2008.923415.

[21] S. Mittal, A Survey of Techniques for Architecting and Managing Asymmetric Multicore Processors, ACM Computing Surveys (2016).

[22] A. Annamalai, et al., An opportunistic prediction-based thread scheduling to maximize throughput/watt in amps, in: PACT'13, 2013, pp. 63–72.

[23] J. Souza, et al., A Reconfigurable Heterogeneous Multicore with a Homogeneous ISA, in: DATE'16, 2016, pp. 1598–1603.

[24] A. Venkat, D. Tullsen, Harnessing ISA diversity: Design of a heterogeneous-ISA chip multiprocessor, ISCA'14 (2014) 121–132.

[25] A. Venkat, H. Basavaraj, D. M. Tullsen, Composite-ISA Cores: Enabling Multi-ISA Heterogeneity Using a Single ISA, in: 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA), IEEE, 2019, pp. 42–55. doi:10.1109/HPCA.2019.00026.
URL https://ieeexplore.ieee.org/document/8675215/

[26] E. Blem, et al., Power struggles: Revisiting the RISC vs. CISC debate on contemporary ARM and x86 architectures, in: HPCA'13, 2013, pp. 1–12.

[27] E. Blem, 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).

[28] B. Lopes, et al., Shrink: Reducing the ISA Complexity Via Instruction Recycling, ISCA'15 (2015) 311–322.

[29] D. Reddy, et al., Bridging functional heterogeneity in multicore architectures, ACM SIGOPS Operating Systems Review (2011) 21.

[30] OpenSPARC T1.
URL https://www.oracle.com/technetwork/systems/opensparc/opensparc-t1-page-1444609.html

Jeckson Dellagostin Souza received his MSc and PhD degrees from UFRGS, Brazil, in 2015 and 2020, respectively. His primary research interests include binary compatibility, heterogeneous processors and multicore environments, particularly focusing on power reduction techniques. For more information, please visit http://www.inf.ufrgs.br/~jdsouza/.

Pedro Henrique Exenberger Becker received his BSc degree in Computer Engineering in 2018 and his MSc degree in Computer Science in 2019, both from Universidade Federal do Rio Grande do Sul (Brazil). In January 2020 he joined Universitat Politècnic de Catalunya (Spain) where he is currently pursuing his PhD. His research focuses on the area of computer architecture for performance- and energy-constrained applications, particularly targeting hardware support for autonomous driving systems. Contact him at pedro(at)ac.upc.edu.

Antonio Carlos Schneider Beck received his Dr. degree from UFRGS, Brazil, in 2008. Currently, he is an associate professor at the Applied Informatics Department at the Informatics Institute of UFRGS, in charge of Embedded Systems and Computer Organization disciplines at the undergraduate and graduate levels. His primary research interests include computer architectures and embedded systems design, focusing on power consumption. For more information, visit www.inf.ufrgs.br/~caco/.