Tuning the ISA for increased heterogeneous computation in MPSoCs

Pedro H. E. Becker, Jeckson D. Souza, Antonio C. S. Beck
Institute of Informatics, Universidade Federal do Rio Grande do Sul (UFRGS), Brazil
{phebecker, jdsouza, caco}@inf.ufrgs.br

***Abstract*—Heterogeneous MPSoCs are crucial to meeting energy efficiency and performance, given their combination of cores and accelerators. In this work, we propose a novel technique for MPSoCs design, increasing their specialization and task-parallelism within a given 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 make room to add extra (full and simpler) in-order cores and hardware accelerators. While applications must migrate from Partial-ISA cores when they need the removed ISA support, they also execute at lower power consumption during their ISA-extension-free phases, since partial cores have much simpler datapaths compared to their full-ISA counterparts. On top of it, the additional cores and accelerators increase task-level parallelism and make the MPSoC more suitable for application-specific scenarios. We show the effectiveness of our approach by composing different MPSoCs in distinct execution scenarios, using the FP instructions and RISC-V ISA as a case study. To support our system, we also propose two scheduling policies, performance- and energy-oriented, to coordinate the execution of this novel design. For the former policy, we achieve 2.8x speedup for a neural network road sign detection, 1.53x speedup for a video-streaming app, and 1.2x speedup for a task-parallel scenario, consuming 68%, 75%, and 33% less energy, respectively. For the energy-oriented policy, partial-ISA reduces energy consumption by 29% over a highly efficient baseline, with increased performance.**

***Index Terms*—MPSoC Design, Partial-ISA, Heterogeneous Systems**

I. Introduction

Heterogeneous Multiprocessor Systems on Chip (MPSoCs) combine performance-driven and energy-efficient cores - based on different microarchitectures - *and* dedicated hardware accelerators. The former is to equate performance, area, power, and energy constraints of general purpose workloads while the latter is to handle specific time-costly jobs such as cryptography, graphics, and Artificial Intelligence (AI) [1].

However, while MPSoCs cores benefit from microarchitecture heterogeneity, they are frequently restricted by an homogeneous Instruction Set Architecture (ISA) [2], [3] (i.e. all cores execute the same instruction set). Hence, the increased number of ISA extensions (e.g., Intel Intrinsics and ARM NEON) that emerged to deal with trendy applications niches [4] (e.g., multimedia) can be a symptomatic burden to the system. As every core has to replicate full-ISA support, including resource-hungry instruction extensions, processor's complexity mounts. Indeed, Blem et al. [5] pointed out that the ISA impact on a system's power and performance mainly depends on the presence or absence of specialized ISA extensions (details in Section II). Nonetheless, implementing such instructions can introduce high resource overheads in the core. Floating-Point (FP) operations, which we explore in this work, are straightforward examples. In our experimentation, the amount of FP instructions in applications such as image processing and graph traversal is minimal (0%-2%), while math-driven linear algebra workloads reach moderate usage (15%). However, a complex Out-of-Order (OoO) processor dedicates 37% of its area to support FP, as we show in Table I. Therefore, we observe that supporting these extensions in *some* (instead of *all*) processors can be more area and power-wise.

For such, we propose to trim-out the parts of the datapath related to such extensions, as long as it does not compromise the overall system performance. By removing a subset of the ISA of some cores (transforming them into *partial-ISA* cores) while maintaining full support in others (*full-ISA* cores), we preserve compatibility with existent binaries. This approach makes room for extra hardware (either simpler full cores or application-specific accelerators) respecting the same area and power constraints as the baseline. As case study, we remove Floating-Point Units (FPUs) from some of the OoO cores. With the extra area, we add extra energy-efficient in-order cores and accelerators for emerging applications (Convolutional Neural Network (CNN) and video-decoding), increasing performance and reducing energy consumption. We also present two mapping strategies (for either performance

[1]A 4-issue BOOM processor [6] (based on ARM A15), in 15nm synthesis [7], [8]. The floating-point datapath includes a FP register file and two OoO FP functional-units. L1 caches considered. More details in Section IV.

TABLE I: The impact of FP support on an OoO processor[1]

Processor	Area (mm^2)
Out-of-order processor	0.34257
Out-of-order floating-point datapath	0.12453
% Area for floating-point support	∼37%

This work has been supported by the the CoCoUnit ERC Advanced Grant of the EUs Horizon 2020 program (grant No 833057) and the Spanish State Research Agency under grant TIN2016-75344-R (AEI/FEDER, EU). This study was financed in part by the Coordenao de Aperfeioamento de Pessoal de Nvel Superior - Brasil (CAPES) - Finance Code 001, the Fundao de Amparo Pesquisa do Estado do RS (FAPERGS) and the Conselho Nacional de Desenvolvimento Cientfico e Tecnolgico (CNPq).

or energy-oriented execution) to support this new level of heterogeneity.

To evaluate our system, we compose different partial-ISA MPSoCs designs to execute different multi-task scenarios such as AI road sign detection, video-streaming, and a typical smartphone app. We then experiment these execution scenarios under the performance and energy-oriented scheduling policies. As we detail in Section IV, our solution reduces energy consumption by up to 75% in the former policy, and up to 53% in the latter (where the baseline is already very efficient). The increased number of computing nodes (accelerators and cores) increase performance for both specialized (up to 2.8×) as task-parallel (up to 1.2×) scenarios. In the overall, we show that partial-ISA MPSoCs always lead to better Energy-Delay Product (EDP) in *every* configuration evaluated when compared to traditional MPSoC designs.

The remaining of this article is organized as follows. Section II presents an overview of related works, their differences and the novelty of this work. After, in Section III, we present the architecture of MPSoCs aided with partial-ISA. Subsequently, in Section IV, we explain the methodology for our experimentation and discuss the results. Finally, Section V summarizes the conclusions of this work.

II. Related Work

A few works had previously covered the impact of the ISA on the design of a system. The work of Venkat and Tullsen [9] exploits the advantages of different architectures to build an ISA-rich system, which combines cores with x86-64, thumb, and Alpha ISAs to exploit their different semantic characteristics. In [10] and [5], Blem et al. discuss how the RISC and CISC models affect modern architectures. By the combination of [9] and [5], 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 vectorization, FP, crypto) instructions. In a recent study, Venkat et al. [11] proposed a composite-ISA that comprise features from different ISAs in a single one. This approach allows a Design Space Exploration (DSE) in both the architecture and organization of the processor cores. In [12], an extensive analysis of ISA aging and the cost of the decoder for keeping old operations in the architecture set is performed. The authors propose a technique to remove and recycle instructions that are not used by compilers anymore.

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 [13], Li et al. discuss the challenges of implementing the Operating System (OS) support for overlapping ISAs and propose a fair scheduling algorithm for these systems. In [14], 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. [15] place an *extra* ARM A15, with a reduced-ISA, to alternate execution with a former single full-ISA A15 core in a single application environment only. They evaluate a single application executing on the partial-ISA core, and migrating to the full core whenever a specialized instruction is fetched. There have also been efforts from industry on relieving ISA-extensions burden. For instance, in the UltraSPARC T1 two cores share the same FPU.

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 [13], [14], where they focus on the software/OS support;
- *use the budget* to design richer MPSoC configurations, with more in-order cores and also application-specific accelerators, *respecting* former baseline constraints. Differing from [15] where neither in-order cores or accelerators are adopted, nor former constraints are respected;
- consider partial-ISA in *simultaneous multi-task* scenarios, differently from the single-task in [15], similarly to ARM's global scheduling [3] (but with partial-ISA cores);
- detail *two mapping strategies* to benefit from the novel design, coping with microarchitecture heterogeneity and ISA at the same time, also not covered by previous work;
- demonstrate and discuss the superior performance and efficiency of partial-ISA MPSoCs;

III. MPSoCs and Partial-ISA Arrangement

A. Exploiting partial-ISA

Figure 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 the instruction set (full-ISA) to utilize the entire area/power budget available (Figure 1-a). Alternatively, it is possible 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 (Figure 1-b). This increases heterogeneity since more (and possibly different) components are added to the system.

As a use case, we have excluded the FP instructions from the RISC-V ISA in BOOM [6] OoO cores, which stands as a high source of logic overhead. For that, the entire FP pipeline was removed from the execution unit of the processor; the decoder stage and the issue queue were trimmed by removing support for these instructions; the register renaming table was simplified, and the FP register file in the dispatch stage was also removed. Other secondary hardware components were indirectly influenced, such as the routing logic, write-back, and forwarding structures. The BOOM processor eases the process of changing the microarchitecture, since it is made to be parametric.

This new partial-ISA MPSoC methodology prosper by manifold reasons. First of all, it enlarges the design space, so

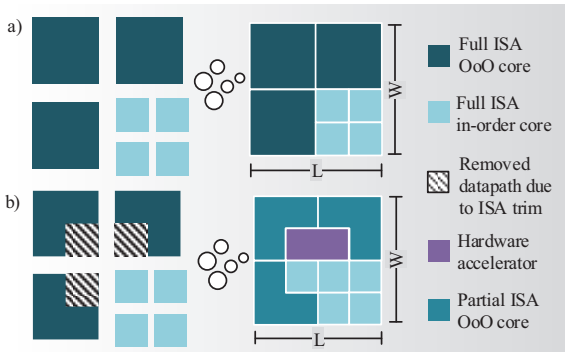


Fig. 1: The system's composition possibilities due to partial-ISA adoption. a) The traditional arrangement. b) Partially trimming the ISA of OoO processors allow adding more computing nodes (in-order processors and accelerators).

the designer can explore diversified modules to compose the final chip. Since partial-ISA cores decrease area and power consumption, the system can leverage an increased number of computing components for task-parallelism and specialization.

B. Partial-ISA MPSoC Flow and Application Mapping

As already explained, 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 full-ISA cores whenever necessary). To coordinate the execution flow in the partial-ISA MPSoC, we developed a scheduler that holds a queue of workloads and the list of computing nodes available in the system. It is aware of which cores are OoO or are in-order (also referred as big and little, respectively); their supported ISA (full or partial); and which nodes are hardware accelerators. The scheduler takes place periodically and verify whether there are workloads to dispatch and preempt[2].

We propose two possible policies for mapping workloads to cores: the *performance-oriented policy* and the *energy-oriented policy*. The policy defines how the scheduler assigns a task when more than one computing node is available (if there is a single node available, the workload is assigned to it). In the *performance-oriented policy*, the scheduler prioritize using OoO cores. In the *energy-oriented policy* otherwise, in-order cores are preferred. Regardless 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. In the exception situations - when full-ISA is necessary (indicated by the *full-ISA dependency* mark, as explained below) - the scheduler assigns the workload to a full-ISA core.

A preemption can occur earlier if a partial-ISA core fetches an ISA-extension instruction (FP in our study), which it does not support. In this case, the core throws a trap[3], informing the scheduler the workload needs a full-ISA core. The workload is marked as *full-ISA dependent*, so the scheduler transparently migrates it to a full-ISA core afterwards. It is also important to remove the *full-ISA dependency* when FP support is no longer mandatory, so the workload can be scheduled to partial-ISA cores again. It is removed from the workload after it executes 10K cycles without fetching a FP instruction.

Finally, applications that have accelerated target regions (which are marked by special function calls or code annotation) trigger a migration if an appropriate accelerator is available. Therefore, when the program reaches an accelerated target region, the application is preempted and marked to execute in the accelerator (similarly to the full-ISA dependency). The accelerator dependency is removed after executing the target region, or as soon as the scheduler asserts that there is no accelerator available. Then, the workload can be mapped to a GPP again.

From the OS perspective, the core's ISA heterogeneity is the only change to be handled, since heterogeneous microarchitectures and accelerators are already in commercial releases. For such, there must be adjustments in the OS's trap handlers and scheduler to manage migrations from full to partial cores. The work by Li et al. [13] details the necessary modifications in the Linux kernel to support this sort of task migration in ISA heterogeneous systems.

IV. Results

A. Methodology

To evaluate our proposed *MPSoCs with partial-ISA processors*, we have used the Rocket (for in-order cores) [17] and BOOM (for OoO cores) [6] RISC-V core generators. Both generators provide a toolchain for a parametrizable 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 [2], [18]. For the partial-ISA cores, the FPU (as well as all the related hardware, as discussed in section III-A) was removed. By modeling these configurations in the core generators, and providing the generated Verilog and frequency constraints to a synthesis tool (Cadence Genus), we could extract precise information regarding area occupation and power consumption of those cores. In our core synthesis, we use a 15nm cell library [7], while for estimating area and power of L1 caches we use FinCACTI [8] (under the same technology). We had also employed an h264 video-decoding accelerator and a convolutional CNNs accelerator to explore design space with partial-ISA. The h264 video decoder [19] is an open-source project, and hence we could synthesize it as we did with the GPP cores. The CNN accelerator data was extracted from the work on [20], with results scaled from 65nm to 15nm based on the study in [21].

We have used the gem5 simulator [22] for performance analysis, running RISC-V binaries. Cores were modeled with the same parameters used on Rocket and BOOM toolchains, and applications were compiled with *riscv-gcc* using the -O3

[2]Workloads are preempted from General Purpose Processors (GPPs) after executing for around 160K cycles, a period that introduces minimal impact on performance as suggested by previous studies [16], avoiding starvation and allowing other workloads to execute on the core.

[3]We consider a trap to be like the *X86_TRAP_UD* for the x86 ISA in Linux.

TABLE II: The execution-scenarios evaluated in this work.

Scenario	Tasks
Smartphone app	rsynth qsort jpeg-d susan-c susan-e susan-s stringsearch
Multitasking	Mix of >30 FP and INT instances of MiBench apps
Road sign detection	aes crc32 cnn_inference
Video streaming	aes crc32 h264_video_dec
FP-driven	correlation gemm 2mm 3mm cholesky ludcmp gramschmidt jacobi-1d

flag. We have modified gem5 to trace FP instructions, not supported by partial-ISA processors, and also the points of interest for the accelerators usage. We use these gem5 traces to feed a simulated scheduler, which carries out the migrations policy, and extracts performance and energy (using cycles/time from gem5 and power from cadence genus synthesis) on the different scenarios. With gem5 we had also assessed the required time to flush a cache and refill it with new data, which we considered for the migration costs (when preempting and dispatching workloads).

B. Execution-Scenarios and HW-Configurations

We define a set of workloads, which composes *execution-scenarios*, and a set of HW components to create different MPSoCs, which represents different system *hw-configurations*. We consider representative spots on the design space by mimicking real-life situations. The *execution-scenarios* are described in Table II. The first scenario is composed of a set of daily-use smartphone applications such as text-to-speech, image processing, and string manipulation. This set presents some degree of FP instructions (1%-5%). The *Multitasking* scenario is meant to analyze how the increased number of cores (allowed by partial-ISA) may impact the MPSoC when a larger number of tasks is demanded at the same time, like in a server workload. The next two scenarios were chosen to measure how the addition of accelerators (also allowed by partial-ISA) can influence the MPSoC performance and energy outcome on latency-critical systems. In these two scenarios, we execute aes and crc32 [23] - simulating data acquisition and checksum - and two data processing kernels: an h264 video-decoding application [24], and a neural network for road sign detection [25]. Finally, we defined a scenario with FP only benchmarks (from PolyBench, with 10%-20% FP instructions), to stress the host MPSoC and verify how it behaves under different *hw-configurations*.

Table III shows the four different *hw-configurations* for an MPSoC with partial-ISA cores. In these configurations, partial-ISA is adopted in OoO cores only, since removing their complex FPU has more impact on the design space (results in 37% reduction of the OoO area) compared to a in-order FPU, which represents $\frac{1}{7}$ of the former. **The baseline hw-configuration is a 4 OoO core + 4 in-order core** system, based on ARM's heterogeneous setup with the DynamIQ technology [3]. Table IV details how the evaluated hw-configurations are within the area and power constraints limited by the baseline.

TABLE III: 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
Video Accelerated	2 OoO 2 OoO (Partial-ISA) 5 In-Order 1 H264 HW Accelerator (Nova)	Video Streaming
CNN Accelerated	1 OoO 3 OoO (Partial-ISA) 4 In-Order 1 CNN Accelerator (Origami)	Road Sign Detection

TABLE IV: The area and power of evaluated configurations.

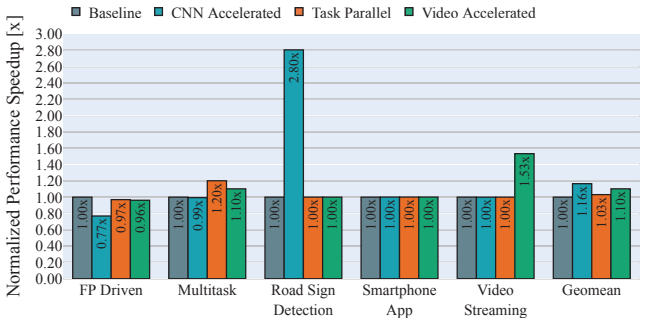
Configuration	Area (mm^2)	Power (W)
Baseline	1.8131	3.4097
Task Parallel	1.7854	2.7626
Video Accelerated	1.7460	2.7062
CNN Accelerated	1.7946	2.3065

Each configuration in Table III aims to optimize one specific scenario from Table II. However, all hw-configurations are experimented under all scenarios, in both *performance-oriented* and *energy-oriented* scheduling policies.

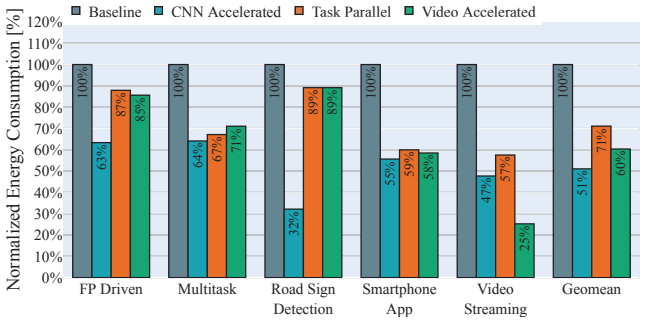
C. Performance and Energy Analysis

Next, we analyze each *hw-configuration* executing on each *scenario*. Figure 2 presents performance and energy under a *performance-oriented* policy. Similarly, Figure 3 depicts results for the *energy-oriented* policy. Performance speedup is measured by comparing the time taken for all workloads within a scenario to finish. 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 configurations with accelerators) and multi-tasking (in the 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.

Task Parallel. 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, either in the performance-oriented, as in the energy-oriented policy. Because of the high core occupation in this scenario, 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 dispatch queued workloads as soon as a core is ready. This leads to the similar performance results among policies. For the remaining scenarios, the trimmed FPUs impacts minimally the performance because the remaining full-ISA cores are wisely leveraged by the scheduler, and FP-forced migrations are small compared to regular time-based

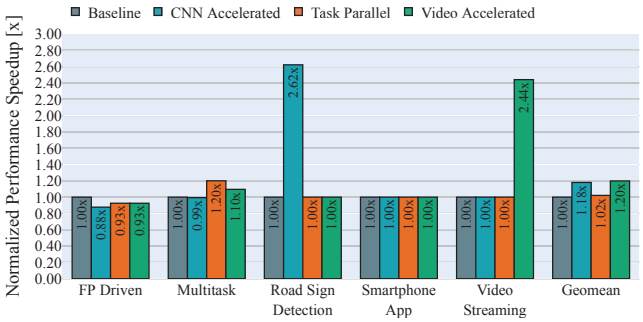


(a) Normalized performance speedup for each configuration and scenario. Higher is better.

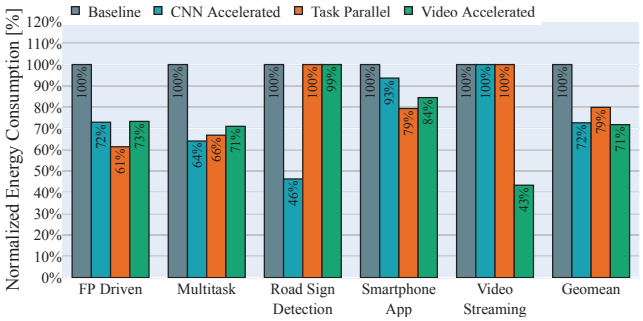


(b) Normalized energy consumption for each configuration and scenario. Lower is better.

Fig. 2: All hw-configurations, including baseline, executing under a performance-oriented policy.



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



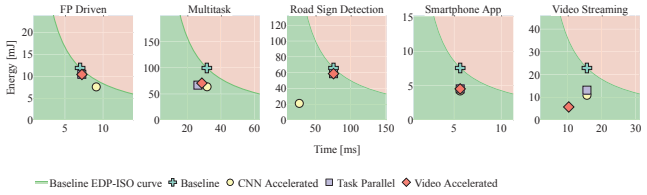
(b) Normalized energy consumption for each configuration and scenario. Lower is better.

Fig. 3: All hw-configurations, including baseline, executing under an energy-oriented policy.

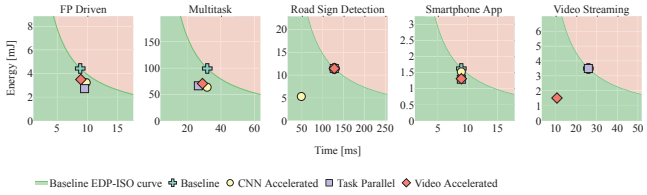
preemption. In the *FP-driven* scenario, however, there is a small reduction in performance because the simpler in-order cores frequently handle FP operations that OoO cores do in the *baseline*. Nevertheless, the benefits in energy consumption are even more evident. This comes from the reduced peak-power 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* policy (*Video streaming*), and 39% in the *energy-oriented* policy (*FP-driven*), reducing by 29% and 21% in the geomean, respectively. Since the baseline executing in the energy-oriented policy (in-order cores preferred) is more efficient than the baseline executing in the performance-oriented policy (OoO cores preferred), it is more challenging to improve energy consumption in the former.

Video Accelerated. 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 1.53× the *Video Streaming* scenario in the performance-oriented policy, and 2.44× in the energy-oriented policy. The speedup is limited by the simultaneously executed *aes* kernel, which becomes the higher latency job after the accelerator is introduced. Because the baseline prioritize in-order cores in the energy-oriented policy, the adoption of accelerators in this case leads to higher speedup when comparing to the performance-oriented policy (where OoO are preferred). We also note the extra in-order core helps task distribution in the *Multitask* scenario, improving performance by 10%. The *Video accelerated* configuration reduces up to 75% the energy consumption on the execution of Video streaming (performance-policy, Figure 2), 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 40% (*performance-oriented*) and 29% (*energy-oriented*). As with the *Task-parallel* configuration, reducing energy consumption is more challenging in the energy-oriented policy because the baseline is already very efficient, leveraging its in-order cores.

CNN Accelerated. In this hw-configuration we add a convolutional CNN hardware accelerator, which require trimming the ISA of three OoO cores. Likewise the *Video Accelerated* configuration, the accelerator improves performance for its target scenario (*Road sign detection*). It improves 2.8× and 2.62× for the performance and energy-oriented policies, respectively. In this case the *cnn_inference* remains as the longest-latency workload, hence overall speedup depends on the share of the program where the accelerator applies. The accelerated region takes the same time in the accelerator in both policies, but it is proportionally less of the total time in the energy policy, where the baseline takes longer in the non-accelerated regions of the kernel, causing a slightly different speedup. Moreover because there is no additional in-order



(a) Performance-policy



(b) Energy-policy

Fig. 4: EDP on *performance* and *energy* oriented policies.

cores in this case there is no improvement in performance for other scenarios, and also the overhead for the *FP-driven* is higher. However, the 3 partial-ISA cores gracefully reduce energy. This *hw-configuration* consumes about half the geomean energy of the *baseline* in the performance-oriented policy, and 28% less in the energy-oriented policy.

D. EDP Analysis

Finally, Figure 4 depicts an overview of the Energy-Delay Product (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 to asses that partial-ISA adoption deliver improved EDP configurations for every scenario, including *FP-Driven* where the need for FPUs rises. For the scenarios with speedup (Multitasking, Video streaming, Road sign detection), both energy and performance improvements places the partial-ISA designs nearer to the optimal EDP (in the origin of the axes). For the *Smartphone app*, where longest-latency is not altered by partial-ISA, the existence of simpler partial-ISA and extra in-order cores reduces energy through lower peak-power and higher throughput, narrowing the number of cores in use until the last workload finishes. For the *FP-Driven* scenario, the significant energy reduction from partial-ISA configurations compensates the minor performance overheads, showing the proposed system can overcome a heavily FP scenario, delivering improved EDP.

V. Conclusion

In this work, we proposed a novel idea for MPSoCs design, improving its computation density under area and power constraints. For such, we trim-off the support for costly ISA extensions in some MPSoC's cores. Also, we present two mapping strategies to cope with full-ISA support when applications need. We tested our proposal regarding the FP operations of a RISC-V ISA extension as a case study, evaluating power and area reduction with real-life processor descriptions. Results showed that at least one (often multiple) hw-configuration reduces energy consumption, in each evaluated scenario. Through an extra in-order cores and accelerators, some scenarios also have sensitive performance improvements. Finally, the proposed partial-ISA presents better energy-delay trade-offs, reducing EDP in some scenarios while being as good as the baseline when it does not.

References

[1] Apple, "iPhone XS - A12 Bionic - Apple," 2018. [Online]. Available: https://www.apple.com/iphone-xs/a12-bionic/

[2] P. Greenhalgh, "Big.little processing with arm cortex-a15 and cortex-a7," *ARM White Paper*.

[3] ARM, "Technologies — DynamIQ Arm Developer," 2017. [Online]. Available: https://developer.arm.com/technologies/dynamiq

[4] A. C. S. Beck *et al.*, *Adaptable Embedded Systems*. Springer New York, 2013.

[5] 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.

[6] K. Asanovic *et al.*, "The berkeley out-of-order machine (boom): An industry-competitive, synthesizable, parameterized risc-v processor," UC at Berkeley, Tech. Rep., 2015.

[7] M. Martins *et al.*, "Open Cell Library in 15Nm FreePDK Technology," in *International Symposium on Physical Design*. ACM, 2015.

[8] A. Shafaei *et al.*, "FinCACTI: Architectural analysis and modeling of caches with deeply-scaled FinFET devices," in *ISVLSI*, 2014.

[9] A. Venkat and D. Tullsen, "Harnessing ISA diversity: Design of a heterogeneous-ISA chip multiprocessor," *ISCA*, pp. 121–132, 2014.

[10] E. Blem *et al.*, "Power struggles: Revisiting the RISC vs. CISC debate on contemporary ARM and x86 architectures," in *HPCA*, 2013.

[11] A. Venkat *et al.*, "Composite-ISA Cores: Enabling Multi-ISA Heterogeneity Using a Single ISA," in *HPCA*, 2019.

[12] B. Lopes *et al.*, "Shrink: Reducing the ISA Complexity Via Instruction Recycling," *ISCA*, pp. 311–322, 2015.

[13] T. Li *et al.*, "Operating system support for overlapping-ISA heterogeneous multi-core architectures," *HPCA*, pp. 1–12, 2010.

[14] D. Reddy *et al.*, "Bridging functional heterogeneity in multicore architectures," *ACM SIGOPS Operating Systems Review*, p. 21, 2011.

[15] W. Lee *et al.*, "Exploring Heterogeneous-ISA Core Architectures for High-Performance and Energy-Efficient Mobile SoCs," *GLSVLSI*, 2017.

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

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

[18] F. Endo *et al.*, "Micro-architectural simulation of embedded core heterogeneity with gem5 and McPAT," *RAPIDO*, pp. 1–6, 2015.

[19] K. Xu and C.-S. Choy, "A power-efficient and self-adaptive prediction engine for H. 264/AVC decoding," *IEEE Transactions on very large scale integration (VLSI) systems*, vol. 16, no. 3, pp. 302–313, 2008.

[20] L. Cavigelli, D. Gschwend, C. Mayer, S. Willi, B. Muheim, and L. Benini, "Origami: A Convolutional Network Accelerator," ser. GLSVLSI. ACM, 2015, pp. 199–204.

[21] A. Stillmaker and B. Baas, "Scaling equations for the accurate prediction of CMOS device performance from 180nm to 7nm," *Integration, the VLSI Journal*, vol. 58, pp. 74–81, 2017.

[22] N. Binkert *et al.*, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, 2011.

[23] M. R. Guthaus *et al.*, "MiBench: A free, commercially representative embedded benchmark suite," in *IEEE WWC-4*. IEEE, 2001.

[24] K. Suehring, "H. 264/avc jm reference software download," *Iphome. hhi. de. Retrieved*, pp. 5–17, 2010.

[25] M. Peemen, B. Mesman, and H. Corporaal, "Speed sign detection and recognition by convolutional neural networks," in *Proceedings of the 8th international automotive congress*, 2011, pp. 162–170.