Towards Energy-Efficient Heterogeneous Multicore Architectures for Edge Computing

Abdoulaye Gamatié, Guillaume Devic, Gilles Sassatelli, Stefano Bernabovi, Philippe Naudin, Michael Chapman

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier XX.XXXX/ACCESS.2019.DOI

Towards Energy-Efficient Heterogeneous Multicore Architectures for Edge Computing

ABDOULAYE GAMATIÉ[1], GUILLAUME DEVIC[1], GILLES SASSATELLI[1], STEFANO BERNABOVI[2], PHILIPPE NAUDIN[2], and MICHAEL CHAPMAN[2]

[1]LIRMM - CNRS and University of Montpellier, France (e-mail: first.last@lirmm.fr)
[2]Cortus S.A. Company, Mauguio, France (e-mail: first.last@cortus.com)

Corresponding author: Abdoulaye Gamatié (e-mail: Abdoulaye.Gamatie@lirmm.fr).

This work has been supported by the CONTINUUM French ANR project under the grant number ANR-15-CE25-0007-01, and the R&D ARPE-CONTINUUM project funded by Région Occitanie (France).

ABSTRACT In recent years, the edge computing paradigm has been attracting much attention in the Internet-of-Things domain. It aims to push the frontier of computing applications, data, and services away from the usually centralized cloud servers, to the boundary of the network. The benefits of this paradigm shift include better reactivity and reliability, reduced data transfer costs towards the centralized cloud servers, and enhanced confidentiality. The design of energy-efficient edge compute nodes requires, among others, low power cores such as microprocessors. Heterogeneous architectures are key solutions to address the crucial energy-efficiency demand in modern systems. They combine various processors providing attractive power and performance trade-offs. Unfortunately, no standard heterogeneous microcontroller-based architecture exists for edge computing.

This paper deals with the aforementioned issue by exploring typical low power architectures for edge computing. Various heterogeneous multicore designs are developed and prototyped on FPGA for unbiased evaluation. These designs rely on cost-effective and inherently ultra-low power cores commercialized by Cortus SA, a world-leading semiconductor IP company in the embedded ultra-low power microcontroller domain. Some microarchitecture-level design considerations, e.g. floating point and out-of-order computing capabilities, are taken into account for exploring candidate solutions. In addition, a tailored and flexible multi-task programming model is defined for the proposed architecture paradigm. We analyze the behavior of various application programs on available core configurations. This provides valuable insights on the best architecture setups that match program characteristics, so as to enable increased energy-efficiency. Our experiments on multi-benchmark programs show that on average 22% energy gain can be achieved (up to 45%) compared to a reference system design, i.e., a system with the same execution architecture, but agnostic of the task management insights gained from the comprehensive evaluation carried out in this work.

INDEX TERMS Edge computing, energy-efficiency, heterogeneous multicore architectures, programming model, embedded systems

I. INTRODUCTION

THE recent trend towards *edge computing* witnessed in the well-established Internet-of-Things (IoT) domain [1] will keep on increasing thanks to new promising hardware solutions enabling applications to meet computing task requirements at affordable costs in power [2]–[4]. This computing paradigm aims to push the frontier of computing applications, data, and services away from the usually centralized nodes located in the *cloud-first architecture*, to the periphery of the network. The resulting decentralization brings a number of benefits [2], [5], [6] including better reactivity and reliability thanks to local compute resources that fill parts of application demands in an isolated way; reduced data transmission costs towards cloud servers thanks to local data processing capabilities; enhanced confidentiality thanks to the ability of transforming sensible raw data before

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2910932, IEEE Access

the transfer on cloud server if required; and human-centered designs in which proprietary information remain under the control of their owners, who can also manage the links of their networks.

Among state-of-the-art compute platforms [7] entering the race to solve the edge computing challenges, we can mention the Intel Movidius Myriad technology [8], the Samsung Exynos 9 Series 9810 processor [9], the Jetson TX2 board [10] and the Machine and Object Detection processors announced by ARM in its Trillium project [11]. An important aim of these platforms is to provide power-efficient compute capabilities for embedded artificial intelligence. This favors autonomous decision-making in the edge. In this context, the integration of several low power processors within the corresponding chips has become the current practice. On the other hand, to deal with the data storage requirements in edge devices (e.g., saving weights in neural networks), Non-Volatile Memory (NVM) technologies [12] have been adopted. They enable low energy consumption while providing fast I/O accesses. They also promote emerging computing paradigms such as in-memory computing [13], which removes the costly data movements occurring in Von Neumann computer architecture, where memory and computing units are physically decoupled.

Harnessing the energy-efficiency of edge computing peripheral nodes, i.e., the amount of achieved work per watt, calls for ultra-low power hardware devices that are capable of delivering adequate computing performance to process data locally on the node. Typical approaches such as the aforementioned ARM Object Detection processors, rely on architecture specialization for particular applications tasks. Thus, they are insufficient for dealing with the general-purpose computing challenge on edge nodes [14]. Further computing platforms that could be considered at the edge include Raspberry Pi, Arduino, and Intel Galileo [7].

However, the real game-changers are expected to be *heterogeneous multicore architectures* supporting any kind of workload within a very tight power budget. The present paper focuses on this direction by exploring compute node designs, built from microcontrollers, with an overall power consumption that remains below a watt. It addresses both the architecture construction and its programming.

A. HETEROGENEOUS ARCHITECTURES

Heterogeneous computing usually refers to systems including various processing elements so as to meet both performance and power-efficiency requirements. Typical heterogeneous architectures combine CPUs and compute accelerators such as Graphical Processing Units (GPUs). While the former are well-suited for executing sequential workloads and the operating system, the latter are rather devoted to massively regular parallel workloads, e.g., data-parallel algorithms. For instance, the Llano processor [15] proposed by AMD and the Jetson TX2 board [10] from Nvidia follow this idea by combining multicore CPUs with a GPU. Other heterogeneous multicore platforms rather combine DSPs with CPUs, as in the KeyStone DSP+ARM SoC[1] from Texas Instruments.

The ARM big.LITTLE technology [16] considers two different clusters: a big cluster composed of high-performance application processors used to execute heavy workloads, and a LITTLE cluster composed of low power application processors that are used for a lightweight workload to save energy. By exploiting this feature, a suitable runtime can provide workloads with required performance while reducing the power consumption whenever possible.

Despite the attractive features of the above heterogeneous multicore chips, they do not offer a power reduction below a watt for aggressive energy sustainability in battery-powered edge nodes. This is hardly achievable with platforms such as the Jetson TX board and ARM big.LITTLE Exynos chip families, which even consume a few watts in idle status. Moreover, platforms that combine processing elements supporting different instruction set architectures, such as the aforementioned Jetson TX2, Llano and KeyStone DSP+ARM SoCs, do not facilitate a uniform and simple programming of applications. Note that some of these chips are not mature and robust enough in real-world commercial solutions [17]. Therefore, exploring complementary opportunities is very relevant.

B. PROBLEM FORMULATION

Our study aims to devise heterogeneous compute node designs [18], [19], which have sub-watt power consumption and can fill the current gap observed in the implementation of edge devices. It is expressed through the following problem.

DEFINITION 1 (DESIGN PROBLEM): *Starting from a family of low power processors, supporting the same instruction set architecture (ISA) and their complementary System-on-Chip (SoC) blocks, we build and evaluate heterogeneous systems. The main requirements taken into account are:*

1) ***cores heterogeneity****: the target architectures rely on the combination of cores with different features resulting from graceful customization, which could be leveraged as much as possible in order to provide the best trade-offs in terms of performance and power;*
2) ***low power hardware architecture****: the cores and SoC blocks used to build the target heterogeneous architectures inherently dissipate low power, which contributes to minimizing the energy consumption of the target architecture;*
3) ***application characteristics-aware execution****: the considered programming model favors workload management in such a way that application programs execute on the most energy-efficient hardware configurations with respect to their characteristics, e.g., compute-intensiveness versus synchronization-intensiveness.*

The work carried out in this paper is based on FPGA prototyping so as to derive performance and power measurements

[1] https://training.ti.com/keystone-ii-dsparm-soc-architecture-overview

with the highest possible confidence.

C. OUR CONTRIBUTION

In light of the issues raised above, this paper advocates a novel asymmetric multicore architecture, together with an associated programming model and workload management. This architecture includes ultra-low power cores devoted to parallel workloads for high throughput, and a high-performance core that copes with weakly-parallel workloads. The covered parallel workloads can be either regular and irregular. Even though the above design *a priori* resembles a CPU/GPU heterogeneous combination, it proves far more flexible in the sense that GPU is only practical for rather regular parallel workloads. In addition, GPUs require specific APIs such as OpenCL and CUDA, which are not necessarily supported by CPUs, requiring extensive software support. Our proposal exploits a unique programming model, thus facilitating the programmer's job.

A salient feature of our proposal is the usage of the cost-effective and inherently low power core technology provided by Cortus SA [20], one of the world-leading semiconductor IP companies in the embedded domain. These cores are highly energy (MIPS/μW) and silicon efficient (MIPS/mm^2) compared to existing technologies. We believe the massive usage of such embedded cores deserves attention to achieve the energy-efficient architectures required for high-performance embedded computing. Compared to ARM big.LITTLE, which considers only application processors, our approach combines a high-frequency core and several microcontrollers (not intended to support a full OS), which are key for aggressive energy optimization.

Another trade-off considered in our solution is the support of floating point arithmetic, which is important for a range of applications executed in edge computing nodes: matrix inversion required for Multiple Input / Multiple Output (MIMO); Fast Fourier Transforms (FFT) which often suffer from scaling problems in fixed point; and Machine Learning tasks (e.g., [21]) through the weights neural networks, etc. As floating point units (FPUs) can be expensive in terms of area and power in the very low power cores being considered, it is considered as a customization parameter.

Finally, a tailored lightweight and flexible multi-task programming model is defined in order to describe and manage application programs on the multicore architectures. By taking different programs characteristics into account during workload allocation, we show 22% energy-efficiency improvement on average (up to 45%) while executing multi-benchmark programs, compared to a reference design, measured on FPGA prototypes.

D. OUTLINE OF THE PAPER

The remainder of this paper is organized as follows: Section II discusses some related studies on the design of computing solutions for edge computing; then Section III introduces the architecture building blocks selected for our proposal; Section IV presents the programming model devised for application workload management on top of designed architectures; Section V describes a comprehensive evaluation of different architecture variants, in terms of energy gain; Section VI shows how the insights gained from the previous evaluation can be exploited for improving the energy-efficiency through a better workload scheduling; finally, Section VII gives concluding remarks and perspectives.

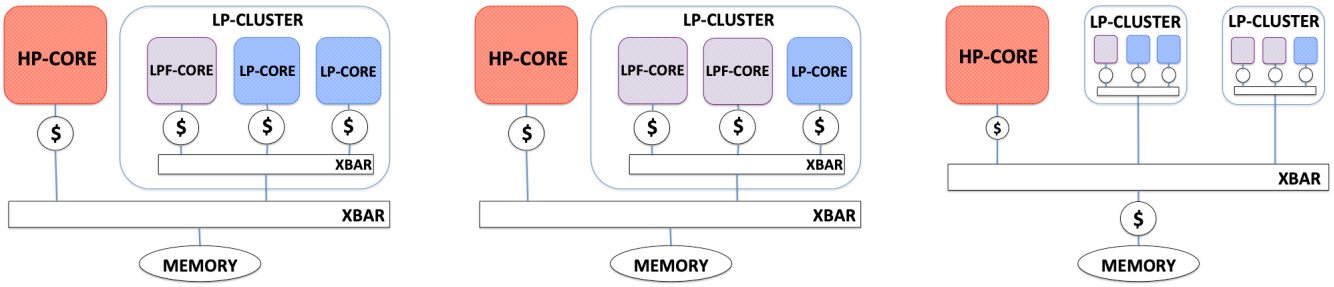
II. RELATED WORK

The need of well-suited heterogeneous architectures for IoT devices has been already motivated [18]. Similarly, the edge computing applications also require such architectures for energy-efficient execution on their compute nodes. A recent survey [19] presents the main microprocessor technologies and computing paradigms that are under consideration for addressing the IoT compute node requirements, i.e., intelligence, heterogeneity, real-time constraints, spatial constraints, inter-node support, etc. A certain number of computing paradigms are distinguished, as follows:

- *configurable architectures*, which support configurable components such as caches [22], reorder buffer [23] or pipeline [24]: they are efficient w.r.t. energy, performance, cost, and area; configurable, i.e., specialized to different applications for better energy-efficiency; and profitable for future applications without being over-provisioned for current applications.
- *distributed heterogeneous architectures* [25], which equip a microprocessor with other core types or configurations, such as CPUs, DSPs or GPUs: they are efficient, profitable for future applications, and extensible, i.e., further microprocessors could be derived from the current ones by extending them with additional functionalities (e.g., specialized instructions).
- *approximate computing* [26] [27], *energy harvesting* [28] [29] and *non volatile processors* [30] [31], which are mainly efficient. The approximate computing paradigm tolerates less accurate results while preserving acceptable output quality. This concession comes with notable performance and energy gains. Energy harvesting for available sources such as solar or radio frequency radiation enables to supplement batteries in ultra-low power nodes. Non-volatile processors integrate NVM to save processor state and quickly restore it later on wake up (after a power disruption). They have been leveraged for energy harvesting systems [32].
- *in-memory processing* [33] [34], which reduces off-chip communications and favors the local processing of the data collected on a node: this computing paradigm is efficient and extensible.
- *secure microarchitectures* [35] [36]: they provide security guarantees since IoT compute node are potentially subject to attacks.

We adopt an approach based on heterogeneous architectures as a design solution in this work. As pointed out in [19], the major part of research efforts on heterogeneous cores has

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2910932, IEEE Access



(a) Quadcore template **(b)**Another quadcore template **(c)**Heptacore template

FIGURE 1: Various templates of the proposed asymmetric architecture

been conducted in general-purpose computers and embedded systems, without explicit application to IoT microprocessors.

Several studies have been carried out in academia on asymmetric architecture design. Hill et al. [37] applied Amdahl's Law to explore different multicore chip architecture designs, namely symmetric, asymmetric and dynamic multicore (which enables multiple cores to work together for sequential execution). They combined an Amdahl's software model with a simple hardware model based on fixed chip resources. They observed that asymmetric and dynamic multicore chips offer the highest speedups. Morad et al. [38] evaluated asymmetric cluster chip multiprocessors for maximizing performance within a given power budget. Here, serial regions of multi-task programs are executed on high-performance cores while parallel regions are executed on both large and small cores. A theoretical analysis, validated by emulation, has been applied to make a comparison with symmetric clusters. The authors observed that asymmetric design can provide a reduction of more than two thirds in power for similar performance while enabling more than 70% higher performance for the same power budget. Both [37] and [38] concluded their study by pointing out the fact that asymmetric architecture design exploration deserves much more attention for improved performance and power-efficiency in modern multicore systems. Reaching this goal obviously requires suitable program execution models capable of exploiting this asymmetric feature [39].

The two major challenges raised by authors in [19] regarding heterogeneous microprocessors for the IoT concern the core configuration (i.e., number and type of cores) and the scheduling of applications to the appropriate cores. This requires a careful analysis of the execution requirements of a wide variety of application characteristics, w.r.t. considered cores. Only a few existing works partially addressed this problem in the literature. In [40], authors tried to understand the interaction between execution characteristics of IoT applications (such as compute or memory intensity) and the architectural features of edge nodes (such as clock frequency, memory capacity) designed with ARM and Intel CPUs. In [41], authors described a design space exploration methodology that focuses on the combination of different CPU microarchitectures to design energy-efficient processors for IoT applications. In both studies, authors mainly focused on the impact of CPU frequencies and cache sizes on the performance and energy when executing the considered benchmarks. They used existing architecture simulators (i.e., gem5 and ESESC) combined with power estimation tools (e.g., McPAT) to perform their respective analyses. While such tools enable reasonable virtual prototyping, they can lead to biased evaluations. For instance, the average error of the used CPU models in such tools is rarely low, e.g., below 20% [42].

The current paper deals with similar issues as in [40], [41]. It relies on a novel asymmetric single-ISA architecture built with cost-effective and very low power core technology. Unlike the aforementioned studies, it considers microarchitecture design trade-offs targeting advanced mechanisms such as out-of-order, in-order, float-point unit execution supports. It adopts an FPGA-based prototyping to avoid unbiased evaluation. Special attention is given to the application workload management on such an architecture in order to optimize both performance and power consumption. We also show that the design trade-off of floating point support plays an important role in performance improvement while benefiting the inherently low-power nature of the cores. Finally, we demonstrate that leveraging some knowledge of application characteristics contributes to reaching this goal.

III. HETEROGENEOUS ARCHITECTURE DESIGN

The design approach adopted for the considered heterogeneous multicore architectures relies on different core customization degrees: i) a basic ultra-low-power and high code density CPU microprocessor without a floating point unit (FPU), ii) a low-power microprocessor having a FPU, and iii) a high-performance application processor based on a fully-out-of-order multiple issue architecture, FPU and full MMU support. This offers more opportunities in terms of performance and power tradeoffs.

A. GENERAL PRINCIPLE

Fig. 1 shows three templates of candidate designs. These templates are arbitrary designs, which, however, aim at providing a trade-off regarding the core diversity requirements, e.g., out-of-order *versus* in-order cores, cores with *versus* without

FPU, for global energy-efficiency. The quadcore architecture depicted in Fig. 1a comprises one high-performance core, referred to as HP-Core[2]; and three low power cores, i.e., micro-controllers, with various features: one core with FPU, referred to as LPF-Core[3] and two cores without FPU referred to LP-Cores[4]. Assuming the Amdahl's law, considering only one single high-performance core for fast execution of serial regions combined with several power-efficient cores appears relevant [37].

In the quadcore architecture depicted in Fig. 1b, the low power cores configuration is different: two LPF-Cores are combined with a single LP-Core.

Core count can be increased as shown in Fig. 1c, through an heptacore system representing a superset of the previous two quadcore templates. This provides a diversity of microarchitecture features that meets the requirements of applications. Indeed, floating point operations are not always present in embedded workloads. In all templates, the cores are connected to the shared memory via a hierarchy of crossbars. The cache memory hierarchy is organized in such a way that every core has its private L1 cache. On the contrary, a unique L2 cache is shared by these cores. This ensures cache coherence by construction for the considered multi-programmed workload setup in this paper. Fig. 2 illustrates a synthesizable implementation of this heptacore template.

B. DESIGN INSTANTIATION

The generic HP-Core, LPF-Core and LP-Core cores referenced in Fig. 1 are respectively implemented with the APSX2, FPS26 and APS25 core technologies[5], developed by the Cortus company.

The APSX2[6] is a recent high-end multiple-issue, out-of-order CPU supporting floating point computation. It was designed as an application processor with features such as precise exceptions handling, branch prediction and multiple threads of execution. Compared to other cores from Cortus, it provides a higher memory bandwidth thanks to wider memory buses.

The FPS26 is an extensible 32-bit core featuring single precision floating point combined with excellent code density. As most Cortus cores, it relies on Harvard architecture with 2×4 GByte address space. It is suitable for creating complex embedded systems with caches, co-processors, and multiple cores, e.g., in audio, vision, advanced control and communication applications. Floating point arithmetic benefits a number of algorithms in those domains.

The APS25 is similar to the FPS26, but has no FPU as a major part of embedded applications do not require floating point calculations. This reduces its complexity in terms of area as well as decreases power consumption. The execution of floating point computations on this core is achieved via a software emulation mechanism. There is a strict inclusion between the above Cortus cores in terms of instruction sets: APS25 is included in FPS26, itself included in APSX2.

TABLE 1: Design elements assessment

	FPGA metrics	ASIC metrics		
	Slices	Gates	Area (μm^2)	Power (mW)
HP-Core	122941	1471462	1341624	4
LPF-Core	7919	93083	134039	0.86
LP-Core	3981	47648	68613	0.42
Intercon. (Quadcore)	7359	24736	119733	0.78
Quadcore (Fig. 1a)	164653	1551905	4431142	6.48
Quadcore (Fig. 1b)	168591	1599038	4499015	6.92
Heptacore	205135	1809319	6160819	8.62

Table 1 provides an assessment of different design elements in terms of the number of FPGA slices. Furthermore, number of gates, area and power figures resulting for a synthesis targeting a UMC 55nm ULP ASIC technology are indicated. This assessment relies on two prototypes: a Kintex-7 FPGA embedded in the Genesys 2 board of Diligent [43], and a Virtex Ultrascale VCU108 FPGA evaluation kit. The power consumption indicated in Table 1 has been estimated while assuming a toggling activity of 50% of the synthesized logics every clock tick. Cache memory is covered in these design estimations, but not the external memory. Through the reported numbers, we can observe the higher complexity of the HP-Core compared to LPF-Core and LP-Core, due to its advanced features. In addition, the presence of a floating-point unit in LPF-Core makes this core twice costly than the LP-Core.

In order to meet the energy-efficiency requirements of the overall target system, the design of the communication infrastructure should consider a trade-off between complexity (required die area and the corresponding dissipated power), transfer speed, latency, and throughput. In general, very simple systems composed of a few cores can use a shared bus. When the core count increases, a crossbar becomes more attractive, allowing multiple accesses between cores via high-speed paths. As the number of potential paths between cores increases the complexity of the crossbar increases to a point that a large portion of the die is reserved for the crossbar and timing closure becomes increasingly difficult. At this point a network-on-chip (NoC) is desirable. Nevertheless, the point at which a NoC becomes necessary can be postponed by using a multi-level crossbar system where the number of communicating cores per crossbar is reduced. Our design templates adopt this last approach (see Fig. 1).

For instance, Table 1 shows that the cost of the crossbar interconnect is reasonable compared to that of cores. An estimate for the three design templates (memory is not included as assumed to be off-chip). An interesting observation is that the scalability of these templates does not dramatically

[2]Acronym for the high-performance core.

[3]Acronym for low-power core **with** floating point unit.

[4]Acronym for low-power core **without** floating point unit.

[5]Note that in the Cortus hardware platform, RISC-V cores and their software tools have been already adapted by the company to provide a leading RISC-V based solution.

[6]For reasons of confidentiality, some details are omitted.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2910932, IEEE Access

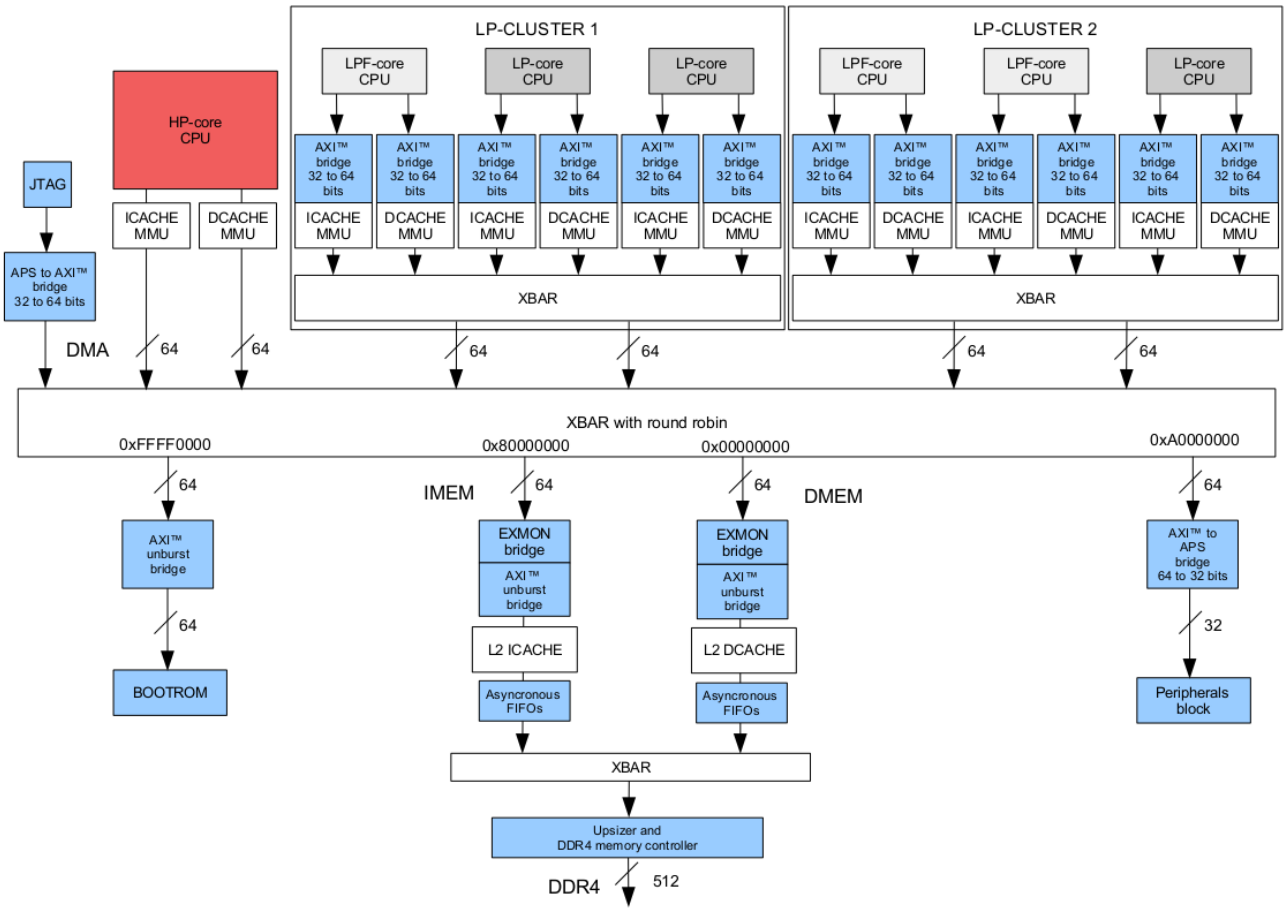


FIGURE 2: Design of the asymmetric heptacore architecture shown in Fig. 1c.

degrade their cost in area and power, while performance improvement is expected thanks to more parallelism.

Compared to ARM microarchitectures, which are the biggest competitors, the aforementioned cores provide more attractive performance scores as illustrated in Table 2. Here, a comparison of considered Cortus cores with relevant ARM microarchitectures [44] is given in terms of *Dhrystone Million Instruction per Second (DMIPS) per MHz*, which is a representative metric for processor performance evaluation.

TABLE 2: Microarchitecture comparison: Cortus versus ARM.

Performance values (DMIPS/MHz)			
APS25	2.51 DMIPS / MHz	Cortex-M0	0.93 DMIPS/MHz
FPS26	2.51 DMIPS / MHz	Cortex-M3	1.25 DMIPS/MHz
		Cortex-M4	1.25 DMIPS/MHz
APSX2	4 DMIPS / MHz	Cortex-A7	1.9 DMIPS/MHz
		Cortex-A15	3.5 DMIPS/MHz

IV. TAILORED MULTITASK PROGRAMMING

Having a suitable programming model is crucial for adequate exploitation of the proposed asymmetric system design. Here, a task data-flow programming models similar to OpenMP 4.0 [45] or OmpSs [46] is considered. It allows one to define the job of each task and how to execute it on the available cores.

A. PROGRAMMING MODEL

From a syntactic point of view, the programming model considered in this work is close to POSIX Threads programming [47]. Fig. 3 illustrates the correspondence between the two programming styles. One can distinguish the declaration and definition of the functions that are performed by created threads or tasks depending on the programming model. In particular, when focusing on our task-oriented programming model, the specified parameters include the input arguments taken of the functions realized by every task, and the dependency information between tasks. The programming model considered in this work is adequately tailored for Cortus technology-based architecture. Nevertheless, automatic code generation from existing code, e.g. written in POSIX Threads could be envisioned due to their high syntactical similarity.

B. DATA MANAGEMENT

We separate program and data memories for each core. Two additional memory zones are reserved for shared memory and for the memory management unit (MMU) configuration. This makes it possible to compile the same program for

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2910932, IEEE Access

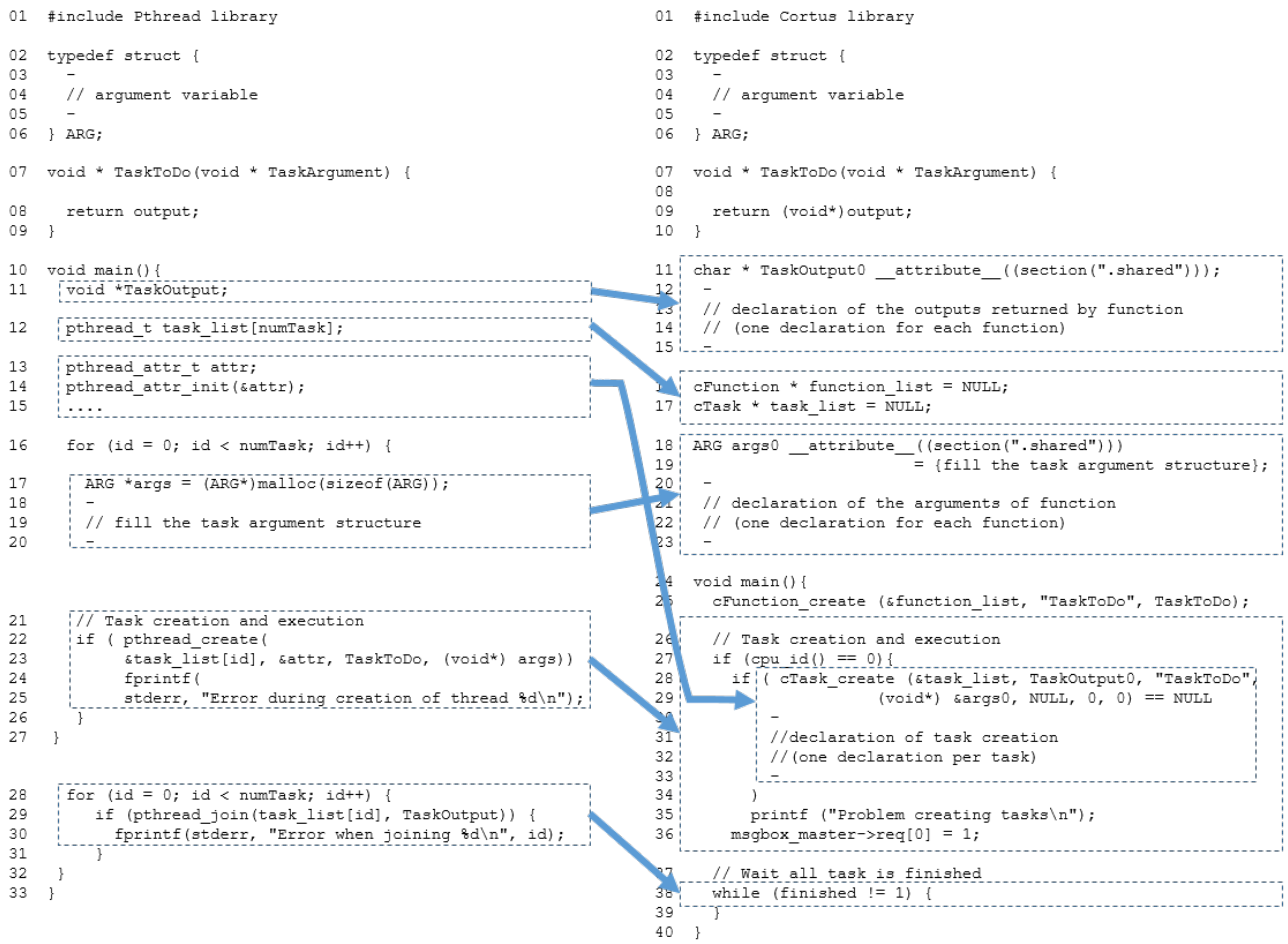


FIGURE 3: Pthread (left) versus our proposed programming model (right)

cores implementing different instruction sets. In Fig. 4a, three different functions `a()`, `b()` and `c()` can be compiled differently, resulting in different machine codes, sizes and memory placements, but unchanged functionality. If a specific code fragment has to be executed by a specific core, the `cpu_id()` run-time function is used to indicate this core.

At the data level, each core has its own data, stack, and heap. To share data, a shared memory section is available, including a shared heap. For dynamic memory allocation, the `smalloc()` and `sfree()` functions are available. A basic lock mechanism for exclusive access is implemented in those functions. The MMU configuration for all cores is stored in a dedicated memory section. Since direct memory mapping is used and is same for every core, this allows for memory saving, hence reducing information replication.

In the shared data, three status vector are provided:

- `cpu_ready[4]` to indicate if a core is ready,
- `cpu_valid[4]` to indicate to a core if data is valid and execution can be started,
- `cpu_assigned_task[4]` to store the address of the task to execute.

Furthermore, a `cpu_lock` variable is available to implement exclusive access to shared resources.

C. TASK SCHEDULING

A cooperative task scheduling is adopted, i.e., a task completes before switching to another task on a given core (meaning no context switch). While this approach is less flexible for real-time workloads, it is simple and more effective for computation-intensive workloads. Multi-tasked execution is eased here by giving tasks a list of dependencies to be met by the scheduler. The scheduling is also dynamic, i.e., tasks can be executed by any available core in any order when allowed. Task declaration is static, thus fixed at compile time. A library of user-level functions and data structures is provided for task scheduling.

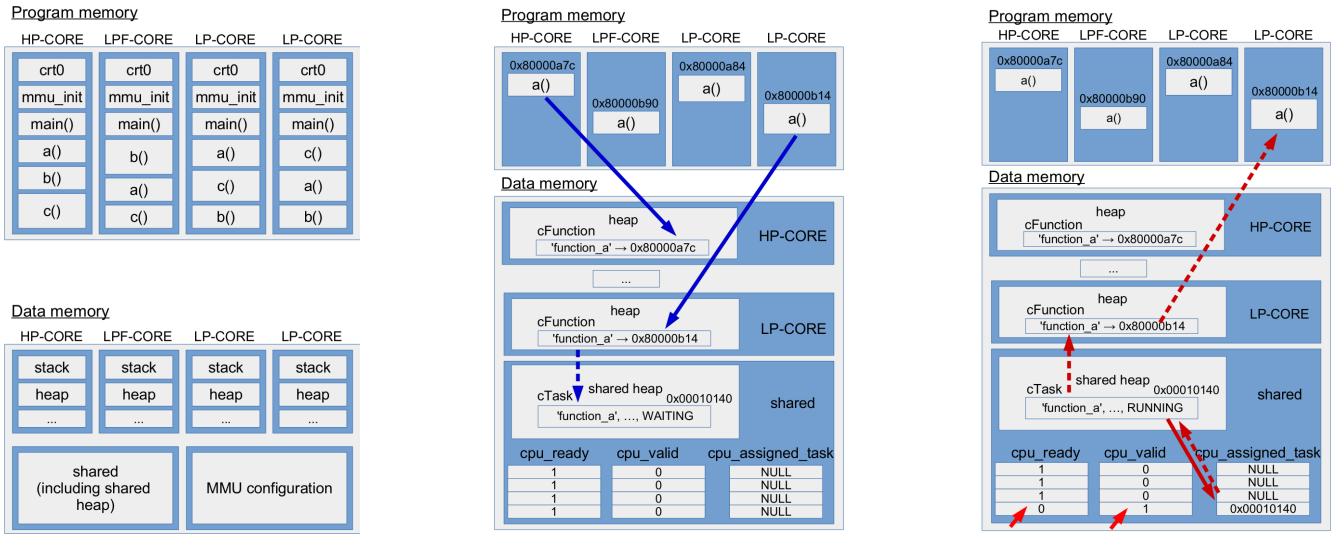
Here, the scheduler is executed on the HP-Core, which plays the role of "master" core that assigns tasks to "slave" cores (i.e., LPF-Core and LP-Cores) and itself.

1) Task creation

Tasks and functions are represented by data structures:

- `cFunction` makes the link between a function and its physical address in memory. This is very important to

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2910932, IEEE Access



(a)memory organization **(b)**task creation **(c)**task execution

FIGURE 4: Multi-threaded management approach

support different instruction sets.

- `cTask` contains the reference to the function with its parameters, the return value, dependencies, status, execution time and further information.

Two creation functions populate these structures and link them in lists:

- `cFunction_create()`: takes a function pointer and a string tag, and associates them in a linked-list working as a look-up table in the heap memory. Each core must perform this creation to have its LUT in its heap.
- `cTask_create()`: takes a function string tag, the parameters, return value, dependencies and information on the presence of floating point computation; then, puts them in a linked list in the shared heap memory. The information about the presence of floating point computation allows the scheduler to execute the tasks on the appropriate CPU configurations, i.e. HP-core and LPF-cores when floating point computation is involved. Parameters and return values are always cast `void*`. This information is available to every core in the shared heap.

In Fig. 4b, `cFunction_create` takes the address of a function and links it with the string tag `function_a` (blue solid arrows). Then, `cTask_create` takes the string tag to create a task in the shared memory (blue dashed arrow).

2) Task execution

When the two steps of creation are performed, the "master" core can start task scheduling, and finally all cores can start execution. For this purpose, two methods are provided:

- `cScheduler_execute()`: launches one iteration of the scheduler, which checks whether:
  - there are tasks available in the task list;
  - dependencies are fulfilled;
  - a core is available (`cpu_ready[i] == 1`);

  If all conditions are satisfied, then a task is assigned to a core. Its address is copied in `cpu_assigned_task[i]` where `i` is the identifier of the target core. The core is signaled via an interrupt or by setting `cpu_valid[i]` to 1. `cScheduler_execute` returns the identifier of the task to be executed.
- `cTask_execute()`: executes a task after retrieving the function to be executed from the function list. When it has finished, it notifies the "master" core.

In Fig. 4c, `cScheduler_executes` (on HP-Core) assigns the task to LP-Core: `cpu_ready[3]` goes 0, `cpu_valid[3]` is set to 1 while the task address is stored in `cpu_assigned_task[3]` (red solid arrows). Then, `cTask_execute` (on LP-Core) takes this address and the control of the task. It uses the function string tag to retrieve the address of the function and executes it (red dashed arrows).

In the above scheduler description, we mentioned a "polling" approach which uses `cpu_valid[]` for signaling and an "interrupt" approach which relies on interrupt routines. The polling approach consists of continuously looking at a memory location waiting for some value (in this case, `cpu_valid[i] == 1`). During idle phase, "slave" cores do nothing and "master" core runs the scheduler. All cores can run a task. A corresponding pseudo-code is as follows:

```
00  if (cpu_id() == 0) {
01    do {
02      remaining = cScheduler_execute();
03      if (cpu_valid[0]) cTask_execute();
04    } while (remaining != 0);
05  } else {
06    do {
07      if (cpu_valid[cpu_id()]) cTask_execute();
08    } while (1);
```

```
09  }
```

This approach is much simpler but less effective. In fact, the "master" core assigns a task to itself and re-runs the scheduler only once this task is completed.

The interrupt approach consists of notifying the core whenever an interrupt signals the start of execution. An interrupt is also sent to signal the end of execution; launching the scheduler. The interrupt routines then call the execution functions. Corresponding pseudo-code is as follows:

```
00 void interrupt_handler(IRQ_MSGBOX_0Mto1S) {
01   msgbox[1]->req[0] = 0;
02   cTaskExecute();
03 }
04 void interrupt_handler(IRQ_MSGBOX_1Sto0M) {
05   msgbox[0]->req[1] = 0;
06   cSchedulerExecute();
07 }
```

Here the routines, shown for HP-Core and LPF-Core, exist for all four cores. This approach is more difficult to handle but, if nesting interrupts is enabled, is more effective. In fact, the "master core" can interrupt its assigned task execution to run the scheduler and assign a new task to a free core as soon as possible.

V. EVALUATION OF THE ASYMMETRIC ARCHITECTURE

A. BENCHMARKING APPROACH

The explored architecture designs will be evaluated by using selected benchmarks. We re-encoded these benchmarks in the task-based programming model presented previously. Table 3 summarizes the entire set of benchmarks. Some characteristics of interest are specified for each program: parallelism (i.e., multi-task), floating point manipulation alongside the major algorithmic features: compute-intensive, many branching instructions, synchronization-intensive, high instruction parallelism and memory-boundedness. This enables to study the tradeoff between the possible architecture configurations w.r.t. the workload characteristics.

TABLE 3: Selected benchmarks

Benchmarks	Parallel workload	Float	Intensity
I-Factorial	No	No	Compute-intensive
F-Factorial	No	Yes	Compute-intensive
FFT	Yes	Yes	Compute-intensive
I-Matmul	Yes	No	Compute-intensive
F-Matmul	Yes	Yes	Compute-intensive
Mpeg	Yes	Yes	Compute-intensive
RandNumCmp	Yes	No	Branch instructions
HashSync	Yes	No	Sync.-intensive
InstPar	Yes	No	Instr. parallelism
Bitonic	Yes	No	Memory-bound
KNN	Yes	Yes	Memory-bound
Stencil	Yes	Yes	Memory-bound

1) Considered benchmarks

The I-Factorial and F-Factorial benchmarks implement algorithms that compute the factorial of integer and floating-point numbers respectively. They are the only sequential programs considered in our experiments. Henceforth, they will be only executed on single-core configurations. This enables to compare the three core types part of the Cortus IP portfolio.

On the other hand, since multicore heterogeneous designs are the main focus of our study, all remaining benchmarks have been re-encoded as parallel multi-task programs. Most of them consist of a set of identical tasks, i.e. each task realizes the same function. FFT [48] is a benchmark where each task executes the same Fast Fourier Transform. The aim is to devise a typical embarrassingly parallel workload that is compute-intensive. In the Mpeg benchmark[7], each task executes an MPEG algorithm.

The RandNumCmp benchmark encodes an algorithm consisting of a loop that iterates five successive if-condition statements. All Boolean conditions in these statements depend on a random integer value. The aim of this benchmark is to make the branch prediction difficult to the processor. Hence, this will result in a high number of branch mispredictions, with variable impact on processor microarchitecture.

The HashSync benchmark implements an algorithm that triggers frequent accesses to a shared and synchronized hash-table. Each task calculates a key corresponding to a row where to insert some elements in the hash-table. By specifying a high number of tasks, this benchmark allows to reproduce the behavior of synchronization-intensive workloads.

InstPar is a simple benchmark that contains a sequence of independent operations that can be executed in parallel. Processors with deeper instruction pipelines efficiently execute such a benchmark.

The last three benchmarks, Bitonic [49], K-Nearest Neighbours (KNN) [49] and Stencil [50], have in common are memory-bound. They can handle large array data structures that lead to many cache misses. Bitonic is an algorithm that sorts the elements of an array in ascending order. KNN implements a classification algorithm commonly used in machine learning. It relies on the calculation of distances between the points within a bi-dimensional space. Finally, the Stencil benchmark often used in image processing consists of matrix cell averaging algorithms. Given a cell, it computes the average of the values in the current cell and its four adjacent cells.

Finally, I-Matmul and F-Matmul encode a matrix multiplication, respectively on integer matrices and floating-point value matrices. Unlike the other parallel benchmarks, these two benchmarks are encoded in such a way that each task computes a different column of the resulting matrix. Note that the following benchmarks are in-house programs that capture well-known algorithms: I-Matmul, F-Matmul, InstPar, HashSync, RandNumCmp, I-Factorial and F-Factorial.

2) Multi-benchmark programs

Based on above benchmarks, we define multi-benchmark programs to reflect realistic application workloads. Indeed, such workloads generally combine more than one

[7]Adapted from http://www.jonolick.com/uploads/7/9/2/1/7921194/jo_mpeg.cpp.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2910932, IEEE Access

of the separate characteristics found in a given benchmark. These considered multi-benchmark programs are described in Table 4. We arbitrarily selected five benchmarks, reflecting different characteristics, which are combined in different ways: I-Factorial (compute-intensive), RandNumCmp (branching), Bitonic (memory-bound), HashSync (synchronization-intensive) and InstPar (instruction parallelism). The idea is to have five different phases in terms of algorithm characteristics. Each phase consists of several similar tasks.

Six variants of multi-benchmark programs are defined (see Table 4). In the variant referred to as Multi-B, each phase has 10 tasks with the same characteristic, i.e., a total of 50 tasks in this multi-benchmark program. Then, in the other program variants, we increase the number of tasks for each phase. This moves the nature of a program towards the algorithmic characteristics of the increased task phase. For instance, the program Multi-CI, which is composed of 50 compute-intensive tasks and 10 tasks for each of the four remaining characteristics will tend to have a compute-intensive dominant characteristic. In total, it has 90 tasks.

TABLE 4: Multi-benchmark program variants

Variants	Dominant character.	Composition
Multi-B	Balanced	10 tasks per characteristic
Multi-BI	Branch instructions	50 tasks with dominant charac. & 40 tasks with the remaining characteristics by groups of 10 tasks
Multi-CI	Compute-intensive	
Multi-MB	Memory-bound	
Multi-IP	Instr. parallelism	
Multi-SI	Sync.-intensive	

B. EXPERIMENTAL SETUP

A major part of the experimentation reported in the sequel is performed on the two quadcore configurations mentioned in Fig. 1. They are implemented separately on two Genesys 2 Kintex-7 FPGA boards of Diligent [43]. The template presented in Fig. 1a is considered for benchmarks without floating-point computations, while Fig. 1b is preferably used for benchmarks with floating-point computations since it includes more cores supporting an FPU.

Both the execution time and power consumption are measured to compare the energy variations according to benchmarks and explored architecture configurations. To enable a high accurate power measurement of the architectures synthesized on the FPGAs, we implemented the apparatus illustrated in Fig. 5, inspired by the JetsonLeap approach [51]. An interesting feature of the considered FPGA board is that it offers the possibility of setting a targeted power supply of the board. Indeed, without using the default 12V power supply, the board can be powered based on the voltage values shown in Table 5. This allows us to directly measure the consumption of the FPGA chip itself (covering all design components, including the external memory used in the FPGA). It is illustrated in Fig. 5. A shunt resistance is used between a power supply (i.e., a current generator) and the power supply of the FPGA board. It enables to measure

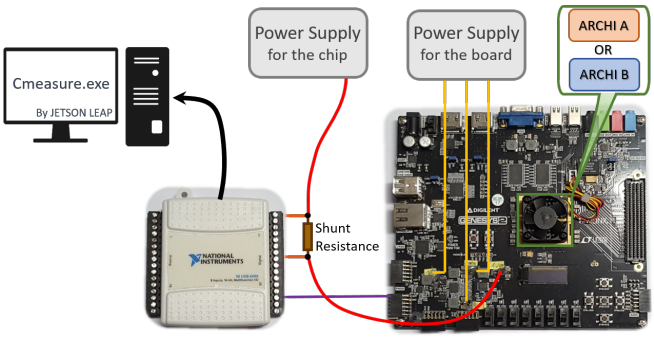


FIGURE 5: Energy measurement setup

the voltage at its boundaries. This voltage is used afterward to compute the instantaneous power consumption and the resulting energy consumption.

In addition to the aforementioned experiments, some complementary evaluations are conducted on an implementation of the heptacore architecture version (see Fig. 1c) to analyze the performance tendency observed on the quadcore architecture prototypes. For this purpose, the Virtex Ultrascale VCU108 FPGA evaluation kit is used to synthesize the heptacore architecture.

TABLE 5: Genesys 2 power supplies

Supplied voltage	Covered circuit components
1.0 V	FPGA cores
1.8 V	FPGA auxiliary
3.3 V	FPGA peripheral & ect
5 V	USB Host & HDMI & DDR3 & ect

C. BENCHMARK EVALUATION

We execute the benchmarks presented in Table 3 on the core configuration space corresponding to the proposed quadcore architectures. The obtained energy consumption values are summarized in Fig. 6. The corresponding execution time and measured power consumption are given in Fig. 7 for a fine-grain analysis.

In the sequel, to enable a convenient comprehensive comparison of all system execution scenarios, we consider two different reference scenarios according to which all the remaining ones are normalized. In the assessment of the quadcore architecture templates (Section V-C1), the reference design consists of a single HP-Core execution. For the heptacore template (Section V-C2), the reference design is the quadcore architecture shown in Fig. 1a.

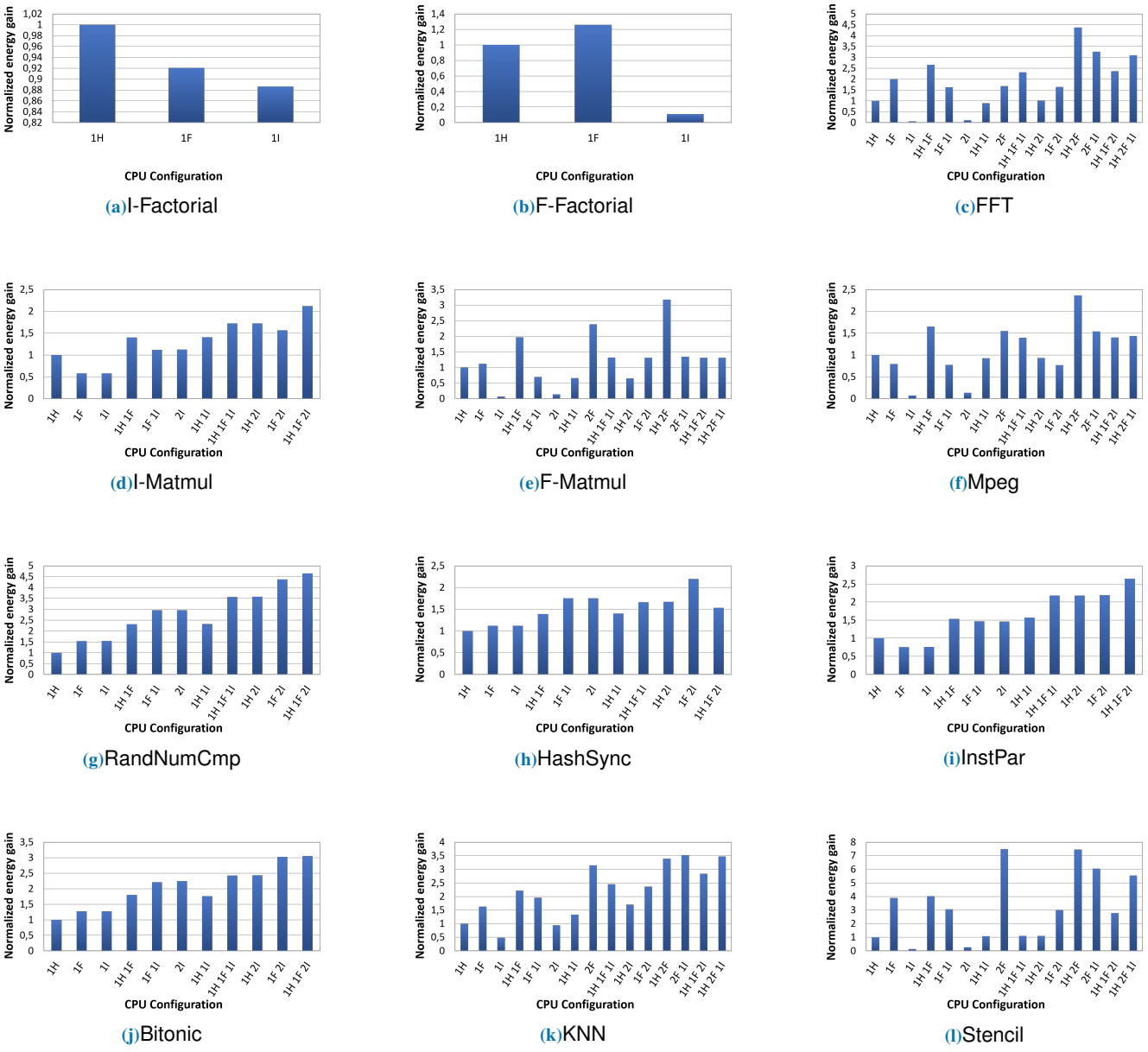
1) Quadcore architecture assessment

For the sake of simplicity, the following notations[8] are adopted to encode the different architecture configurations:

- single core: 1X
- two cores: 1X 1Y, and 2X

[8]Note that only the benchmarks with floating point computations include configuration abbreviations that contain "2F", which refers to the two LPF-Cores available in the template of Fig. 1b.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2910932, IEEE Access



(a)I-Factorial (b)F-Factorial (c)FFT

(d)I-Matmul (e)F-Matmul (f)Mpeg

(g)RandNumCmp (h)HashSync (i)InstPar

(j)Bitonic (k)KNN (l)Stencil

FIGURE 6: Normalized energy consumption comparison for evaluated benchmarks

- three cores: 1X 1Y 1Z, and 1X 2Y
- four cores: 1X 1Y 2Z

where X, Y and Z denote either HP-Core (abbreviated as `H`), LPF-Core (abbreviated as `F`) and LP-Core (abbreviated as `I`). For instance, the configuration `1H 1F 2I` denotes the full quadcore configuration depicted in Fig. 1a.

Compute-intensive workloads. Overall, the obtained results show that for single core execution of compute-intensive workloads, the HP-Core is more energy-efficient than the others. For I-Factorial benchmark, we observe that the HP-core is 8% more efficient than LPF-Core, as reported in Fig. 6a. Since this benchmark has no floating point computation, one would expect the same efficiency for both LPF-Core and LP-Core, but the former is slightly better, i.e., by 4%. When considering the floating point version of the benchmark, i.e., F-Factorial, the benefit of the FPU in the LPF-Core becomes clearly visible through its higher energy efficiency compared to LP-Cores, estimated around 91% as shown in Fig. 6b. Most importantly, for benchmarks performing intensive floating point operations such as F-Factorial, FFT and F-Matmul, the LPF-Core can be even more efficient than HP-core by 26%, 96% and 11.5% respectively, due to the aggressive FPU optimization in LPF-Core.

Similar single-core energy tendencies are observed for all compute-intensive parallel benchmarks, i.e., FFT, I-Matmul, F-Matmul and Mpeg, respectively shown in Figs. 6c, 6d,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2910932, IEEE Access

6e, and 6f. The multicore configurations show that only two LPF-Cores combined with HP-Core provide the best energy consumption for benchmarks containing intensive floating point computations (FFT and F-Matmul). Without floating point computation, as in I-Matmul benchmark, the full quadcore configuration shown in Fig. 1a is the best. Nevertheless, this configuration does not bring a significant gain compared to configurations with three cores. Finally, in presence of floating point computations, both full quadcore architecture depicted in Fig. 1a and Fig. 1b yield a similar energy consumption for the parallel F-Matmul and Mpeg benchmarks. This pertains to the significant execution time overhead induced by LP-Cores in both architecture configurations, which hides any improvement enabled by HP-Core and LPF-Cores. For the FFT benchmark, the full quadcore including two LPF-Cores is slightly better than the other quadcore configuration. More generally, we observe that the best energy gain for parallel floating-point compute-intensive benchmarks is obtained with the configuration `1h 2F`. It means that the combination of HP-Core and LPF-Cores provides the most efficient floating-point execution. When adding LP-Cores, which do not include any FPU in their microarchitecture, the overall performance becomes worse despite a higher execution parallelism due to more cores.

Branching, instruction parallelism and synchronization intensive workloads. Unlike the above observations, the RandNumCmp benchmark containing a high number of branching instructions exhibits similar energy consumption for both LPF-Core and LP-Core in single-core executions (see Fig. 6g). The HP-Core is the least efficient with an additional energy consumption, the low power cores have a better energy efficiency of almost 50%. This is explained by the detrimental impact of frequent branch mispredictions on the microarchitecture of the HP-Core. As a matter of fact, the processor often needs to revert all intermediate results whenever the prediction turns out to be wrong: this implies emptying the instruction pipeline of the core, which further requires to save and restore structures such as renaming tables. In multicore configurations, even though the full quadcore configuration is the most energy-efficient, its gain is very marginal compared to a configuration with only three low power cores. This minimum energy gap is induced by the branch misprediction penalty on HP-Core with quadcore.

For the InstPar benchmark, which is characterized by high instruction parallelism, the HP-Core is more energy-efficient than the LP-Cores by 33% (see Fig. 6i). This is favored by the advanced microarchitecture of HP-Core, e.g., out-of-order execution, deeper pipeline stages. The parallel execution of this benchmark improves the energy by 50% compared to a single core. Most of the configurations with equivalent core count have comparable energy consumption, while those including the HP-Core run 25% faster compared to configurations using only low power cores (see Fig. 7i).

While the above observations were expected for InstPar, a different outcome is obtained for the HashSync benchmark, which is synchronization-intensive. On a single-core, the execution of this benchmark shows low power cores are slightly more energy-efficient than HP-Core, as illustrated in Fig. 6h. In configurations with an equivalent number of cores, those including the HP-Core are less efficient, by 25% compared to low power cores only. This is explained by the overhead induced by the HP-Core microarchitecture management in presence of task synchronizations. More precisely, this overhead comes from the costly context switches occurring in the complex microarchitecture of this specific core (contrarily to low power cores). The HashSync algorithm involves frequent task suspension and resumes. In the end, there is an important execution time overhead due to pipeline stages emptying.

Memory-bound workloads. The Bitonic sorting algorithm shows that execution on two cores makes it possible to obtain an improvement of the efficiency of the order 40% with HP-core and 70% with LP-core compared to a single core (see Fig. 6j). Multicore configurations with the same number of cores have comparable execution time as shown in Fig. 7j, while those using only low power cores are more energy-efficient. The only exception concerns the 3-core configuration 1F 2I and the full quadcore, which have equivalent energy consumption. The latter configuration has a gain of 24% in execution time compared to the former.

In the case of KNN benchmark, an energy gap of 70% between an LP-Core and an LPF-Core is observed, due to the presence of floating-point operations in this program (see Fig. 6k). The HP-Core is also 40% less energy-efficient than the LPF-core. Actually, the quadcore template shown in Fig. 1b, which contains two LFP-Cores is the best choice for the KNN benchmark.

For the Stencil benchmark, which also contains floating-point computations, the LP-Core is, of course, the least energy-efficient. Similarly to the KNN benchmark, the LPF-Core is the best, being nearly 280% better than the HP-core (see Fig. 6l). This huge difference reduces the benefits of the program parallelization. In general, the configurations including one or two LPF-Cores are the most energy-efficient. The LP-core is extremely penalizing, while the HP-core enables only a limited improvement, about 3%, as shown in Fig. 6l when comparing the configurations `2F` and `1H 2F`.

Finally, we generally observe that thanks to the adopted data management approach, the memory bandwidth is never saturated during the execution of the above memory-bound benchmarks. This also confirms an adequate dimensioning of the system.

2) Heptacore architecture assessment

We extend the previous assessment to the heptacore template through a brief comparison of the speedup obtained with the above two quadcore designs (Fig. 1a and Fig. 1b) and the heptacore design (Fig. 1c). For brevity, in the following these three designs are referred to as Archi_A, Archi_B and Archi_C respectively.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2910932, IEEE Access

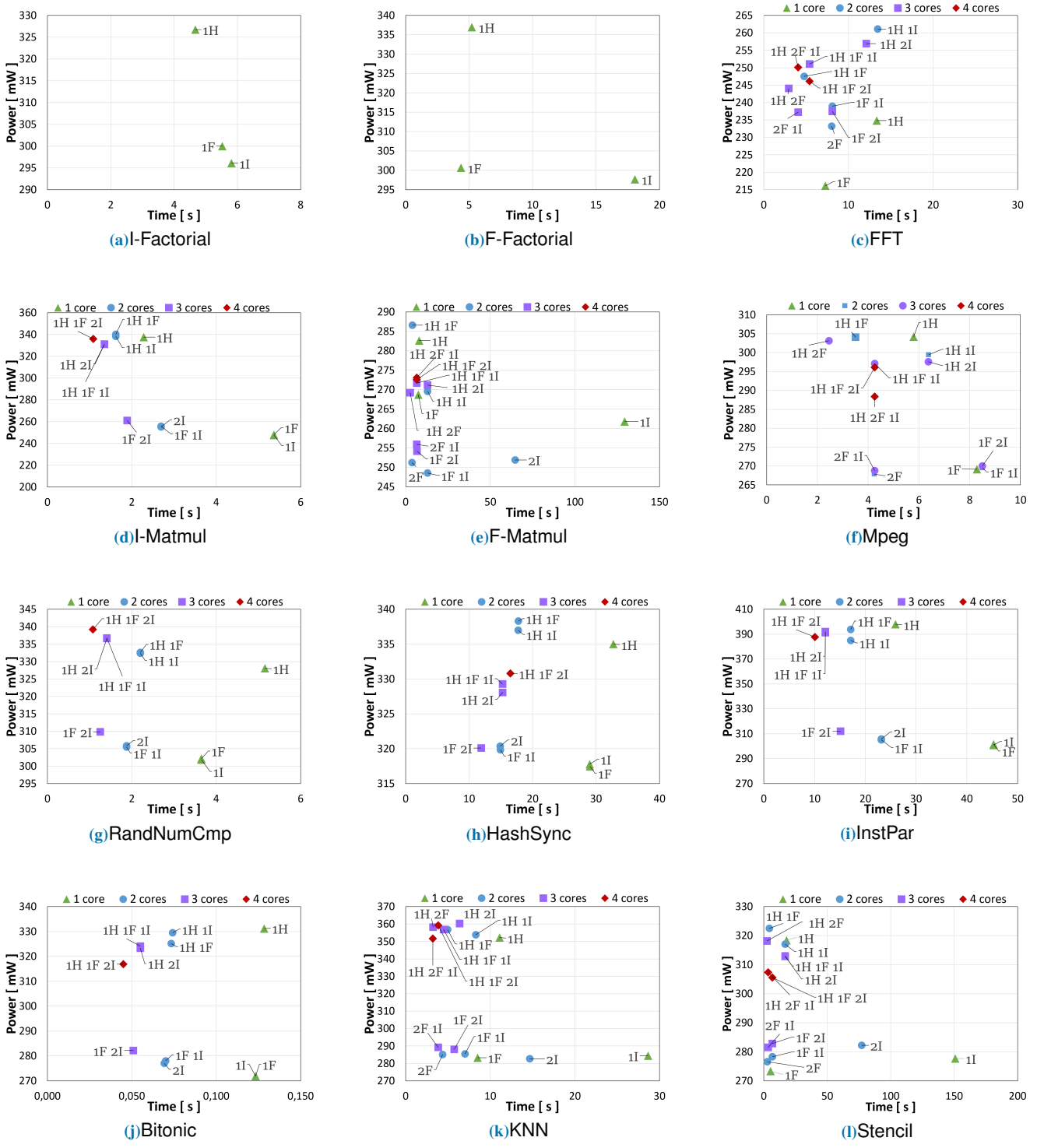


FIGURE 7: Power consumption and execution time for evaluated benchmarks on different core configurations

Overall, we observe in Fig. 8[9] that the heptacore architecture C provides a speedup improvement of 2.4x on average. It is worth mentioning that in the F-Matmul benchmark, architecture C does not bring any speedup improvement compared to the full quadcore configurations. As discussed earlier, this comes from the very low performance of LP-Cores, which always makes the execution time higher: tasks assigned to HP-Core and LPF-Cores terminate earlier, while those executed by LP-Cores complete later. So, the execution

[9]Only, parallel benchmarks are considered in this comparison. In other words, the two sequential variants of the factorial algorithm are ignored.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2910932, IEEE Access

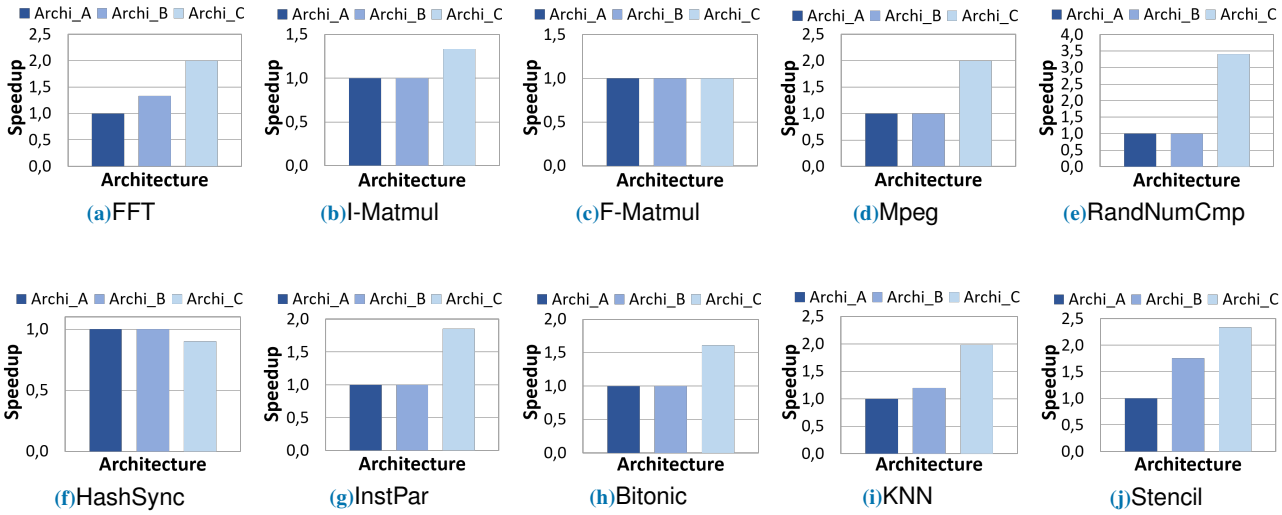


FIGURE 8: Speedup comparison for quadcore and heptacore architecture configurations

acceleration enabled by HP-Core and LPF-Cores is still hidden by the penalty induced by LP-Cores. It is not the case of the Mpeg, which is similar to F-Matmul on the two full quadcore configurations. The negative impact of LP-Cores on the speedup obtained with heptacore is mitigated.

VI. TOWARDS BETTER ENERGY-EFFICIENCY

We first summarize the main insights gained from the above comprehensive architecture evaluation. Then, we show how these insights can be exploited for additional energy gains.

A. SUMMARY OF GAINED INSIGHTS

First of all, even though the covered benchmarks are relatively modest application workloads, the maximum power consumption threshold reached in our experiments is always below 0.4 W. This is favored by the inherently low power SoC blocks integrated in our designs. Note that complementary well-known power saving techniques such as power or clock gating [52], [53] could further contribute to reduce the power consumption measured in our current experiments.

On the other hand, the different architecture evaluations show that the expected energy gains often depend on the workload nature. Fig. 9 summarizes for each benchmark, its best configuration. Fortunately, the heterogeneity of the proposed architecture enables to run workloads on the most favorable configurations.

For benchmarks that scale with several cores, such as I-Matmul or InstPar, noticeable energy gains are observed while selecting configurations with higher core count. As presumed, the architecture configurations including LPF-Cores provide the lowest energy consumption in presence of floating-point computations, e.g., see the F-Matmul, FFT and Mpeg benchmarks. An interesting insight is that the FPU customization implemented in the LPF-Core by the Cortus company is powerful enough to become an alternative choice compared to the HP-Core. Using the latter, which

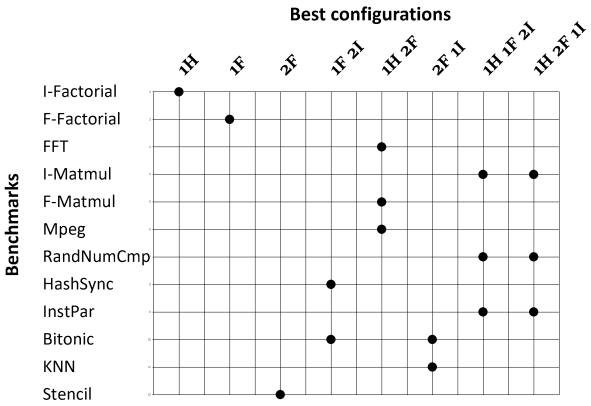


FIGURE 9: Best configuration for each benchmark

dissipates much more power, can be even notably worse for the memory-bound KNN and Stencil benchmarks. However, in the case of the Bitonic benchmark, which is also memory-bound but without floating point computations, increasing the number of LP-Cores contributes to energy minimization, even without the help of the HP-Core.

A non-trivial insight concerns synchronization handling in the considered heterogeneous multicore architecture. The usage of the HP-Core to execute synchronization-intensive algorithms reveals penalizing. A similar remark concerns workloads with high branch misprediction rates as illustrated by the RandNumCmp benchmark. The HP-Core becomes less energy-efficient because of the costly operations occurring in its microarchitecture upon branch mispredictions. More generally, low power cores turn out to be better in the above two situations.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2910932, IEEE Access

B. LEVERAGING THE GAINED INSIGHTS

The above insights result from experiments while considering the task scheduling (described in Section IV-C), which assigns any ready task to a CPU as soon as it becomes available. Let us refer to this scheduler as the Default-sched scheduler.

The affinity between workload nature and architecture configurations, depicted in Fig. 9 can be leveraged to refine the task scheduling policy on cores. More precisely, the ready tasks of a program should be assigned to their most favorable core configurations. To show the potential benefit of such a scheduling strategy, we consider the multi-benchmark programs already presented in Table 4. Then, a typical application example is evaluated to further confirm the relevance of our gained insights. We only present execution scenarios on quadcore architectures.

1) Multi-benchmark evaluation

Each multi-benchmark program is executed according to three scenarios: i) on the full quadcore using Default-sched, ii) on low power cores only, using Default-sched, and iii) on the full quadcore using a new scheduler Opt-sched, which assigns tasks to their best matching configuration in terms of workload nature. The main idea of this experiments is to show that when taking algorithmic characteristics of tasks into account, further energy-efficiency improvements become reachable.

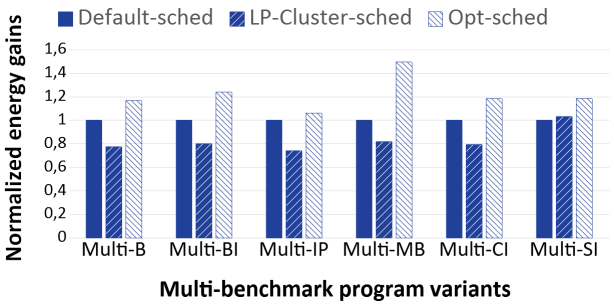
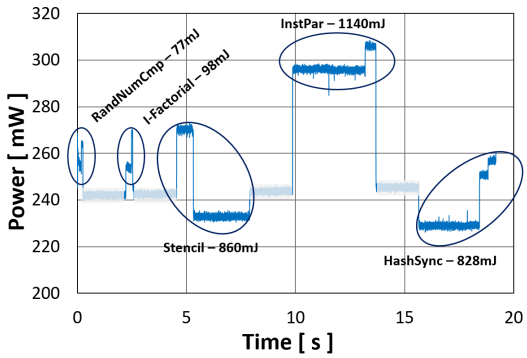


FIGURE 10: Normalized energy gain

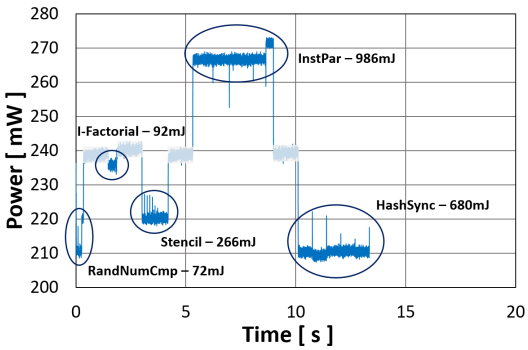
Fig. 10 summarizes the normalized energy gain corresponding to the above three scheduling scenarios for each multi-benchmark program. Globally, we observe that using only low power cores with Default-sched (i.e., referred to as "LP-Cluster-sched" in the figure) is less efficient than the Default-sched scenario, by about 20%. The only exception appears for the Multi-SI multi-benchmark where the poor performances of the HP-core negatively influence the execution of the benchmark with Default-sched.

For all multi-benchmarks, the Opt-sched provides the best energy improvements. The energy gains are about 22% on average (and vary between 5% and 45%) compared to Default-sched in the reported case studies. This confirms the benefits of adaptive workload assignments on the heterogeneous architecture. Here, the task allocation decisions rely on the insights obtained from the comprehensive evaluation presented in Section V-C. There are advanced well-known approaches to deal with such decisions, which are under consideration in our future work. Typically, we can mention dynamic information monitoring and workload mapping techniques [54] [55] and adaptive mapping techniques based on static program analysis [56].

EXAMPLE 1 (FOCUS ON THE EXECUTION OF MULTI-B): *To give a more precise idea of the improvements enabled by the gained insights, let us focus on the executions of the Multi-B multi-benchmark when using the Default-sched and Opt-sched.*



(a) Default-sched



(b) Opt-sched

FIGURE 11: Execution of Multi-B with different schedulers

Fig. 11 depicts the respective energy consumptions for each sub-part of Multi-B, corresponding to a benchmark with a specific algorithmic characteristic. To improve the readability of the produced results, we inserted a dummy task (represented by the light-blue portions of the graphs in Fig. 11a and Fig. 11b) between the different sub-parts of Multi-B. This dummy task has a constant power consumption within each execution scenario.

We observe that the overall execution time of Multi-B is reduced with Opt-sched compared to Default-sched. In addition, for each sub-part (excluding the dummy task portions), the corresponding energy consumption is annotated in the figures. This reflects the obtained improvements.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2910932, IEEE Access

2) Typical data analytics application evaluation

We consider an artificial, yet typical example of application workload that is representative of functions executed on edge nodes. The application features data analytics and mainly performs a linear regression on a set of data collected from some sensors (see Fig. 12). Its result is subsequently used to make predictions. The advantage of deporting such a regression task on the edge node is the reduction of the costly frequent transfer of raw data from sensors to centralized cloud servers.

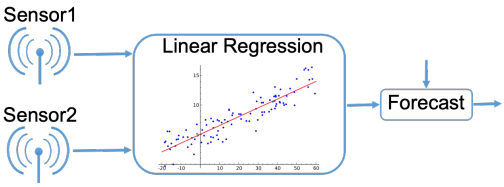


FIGURE 12: Data analytics application example

The executed application is composed of four tasks: the tasks `Sensor1` and `Sensor2` collect data aggregated and exploited in a linear regression process realized by the task `LinReg`. Concretely, the tasks `Sensor1` and `Sensor2` are implemented by random number generation functions. The `LinReg` task implements a linear regression algorithm that takes as inputs two vectors of data values. Finally, a task named `Forecast` takes the regression coefficients computed by `LinReg` for prediction.

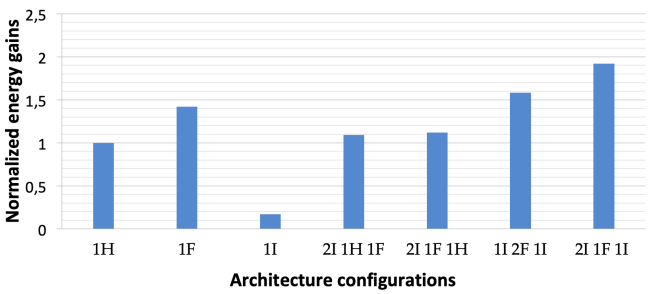


FIGURE 13: Energy comparison for data analytics application

Fig. 13 describes the energy consumption comparison of a few execution scenarios of the data analytics application. This comprises three monocore ("`1H`", "`1F`" and "`1I`") and four quadcore scenarios ("`2I 1H 1F`", "`2I 1F 1H`", "`1I 2F 1I`" and "`2I 1F 1I`"). These scenarios are normalized w.r.t. the HP-Core monocore execution, denoted by `1H`.

According to the tasks' characteristics, their mapping to the most suitable cores enables an energy-efficient execution. Here, this is obtained by executing `Sensor1`, `Sensor2`, `LinReg` and `Forecast` respectively on LP-Core, LP-Core, LPF-Core and LP-Core, corresponding to the architecture configuration `2I 1F 1I`. Indeed, `Sensor1`, `Sensor2` and `Forecast` are not compute-intensive, contrarily to `LinReg`. From our gained insights, the latter might be executed on either the HP-Core or LPF-Core due to the presence of floating point computations in the corresponding algorithm. Here, the LPF-Core reveals more energy-efficient than the HP-Core. The other three tasks can be executed on low power cores. This makes configuration "`2I 1F 1I`" a better candidate than all the others as confirmed in Fig. 13.

VII. CONCLUSION AND PERSPECTIVES

In this paper, we presented the design of heterogeneous multicore architecture templates based on cost-effective and very low power core technology targeting the embedded domain. Our solution combines a high-performance core suitable for sequential execution, and several lightweight low power cores devoted to parallel execution. Prototypes of designed architectures have been implemented on FPGA and reported performance and power consumption figures were measured rather than estimated as in existing works [40], [41]. Further, a tailored and flexible multi-task execution model / API is proposed for efficiently leveraging the flexibility offered by the template in selecting at run-time target cores for processing. This is the first attempt to develop an asymmetric multicore architecture based on the low power core technology of Cortus company. The opportunity of customizing certain low power cores, w.r.t. floating point processing makes it possible to provide a tradeoff in terms of performance, area and energy efficiency. Based on a comprehensive evaluation of the proposed architecture designs, we showed that an adequate multi-benchmark workload management on the heterogeneous cores can provide about 22% energy gain on average, compared to a reference design. This makes our solution a very promising candidate for edge compute nodes where energy efficiency is key.

The core customization exploited in this paper is not limited to FPU or out-of-order execution supports. It can be also extended to other features such as cryptographic primitives or pattern-oriented computations, particularly useful for security or channel coding in edge computing devices. Actually, the Cortus company already provides a range of SoC IPs for addressing security issue. Integrating such IPs in our architectures is one relevant perspective to the present study. Another important perspective concerns the integration of more advanced workload management techniques to increase the overall energy-efficiency of the designed systems. One possible direction may rely on compiler-based static analysis on programs before execution, to infer their features, e.g. see [56]. Then, these features could be exploited for efficient workload mapping and scheduling on the heterogeneous architectures.

ACKNOWLEDGEMENTS

We thank the referees of IEEE Access for their insightful comments and suggestions that contributed to improve this work. We also thank Junio Cezar Ribeiro da Silva, Fernando Pereira, Kais Belwafi and Florent Bruguier for their help and suggestions about the experimental setup used in this paper.

REFERENCES

[1] F. Ganz, D. Puschmann, P. Barnaghi, and F. Carrez, "A practical evaluation of information processing and abstraction techniques for the internet of things," IEEE Internet of Things Journal, vol. 2, no. 4, pp. 340–354, Aug 2015.

[2] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, "Edge-centric computing: Vision and challenges," SIGCOMM Comput. Commun. Rev., vol. 45, no. 5, pp. 37–42, Sep. 2015. [Online]. Available: http://doi.acm.org/10.1145/2831347.2831354

[3] Y. Ai, M. Peng, and K. Zhang, "Edge computing technologies for internet of things: a primer," Digital Communications and Networks, vol. 4, no. 2, pp. 77 – 86, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2352864817301335

[4] M. Satyanarayanan, "The emergence of edge computing," Computer, vol. 50, no. 1, pp. 30–39, Jan 2017.

[5] D. Smith, "Computing at the Edge of IoT," https://medium.com/google-developers/computing-at-the-edge-of-iot-140a888007bd, 2018.

[6] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "A survey on mobile edge networks: Convergence of computing, caching and communications," IEEE Access, vol. 5, pp. 6757–6779, 2017.

[7] A. M. Khan, I. Umar, and P. H. Ha, "Efficient compute at the edge: Optimizing energy aware data structures for emerging edge hardware," in 2018 International Conference on High Performance Computing Simulation (HPCS), July 2018, pp. 314–321.

[8] M. H. Ionica and D. Gregg, "The movidius myriad architecture's potential for scientific computing," IEEE Micro, vol. 35, no. 1, pp. 6–14, Jan 2015.

[9] "Samsung Optimizes Premium Exynos 9 Series 9810 for AI Applications and Richer Multimedia Content," https://news.samsung.com/global/samsung-optimizes-premium-exynos-9-series-9810-for\-ai-applications-and-richer-multimedia-content, 2018.

[10] D. Franklin, "NVIDIA Jetson TX2 Delivers Twice the Intelligence to the Edge," https://devblogs.nvidia.com/jetson-tx2-delivers-twice-intelligence-edge/, 2017.

[11] J. Davies, "Arm is changing machine learning experiences: Project Trillium," https://community.arm.com/processors/b/blog/posts/ai-project-trillium, 2018.

[12] J. Boukhobza, S. Rubini, R. Chen, and Z. Shao, "Emerging NVM: A survey on architectural integration and research challenges," ACM Trans. Design Autom. Electr. Syst., vol. 23, no. 2, pp. 14:1–14:32, 2018. [Online]. Available: https://doi.org/10.1145/3131848

[13] W. Chen, K. Li, W. Lin, K. Hsu, P. Li, C. Yang, C. Xue, E. Yang, Y. Chen, Y. Chang, T. Hsu, Y. King, C. Lin, R. Liu, C. Hsieh, K. Tang, and M. Chang, "A 65nm 1mb nonvolatile computing-in-memory reram macro with sub-16ns multiply-and-accumulate for binary dnn ai edge processors," in 2018 IEEE International Solid - State Circuits Conference - (ISSCC), Feb 2018, pp. 494–496.

[14] B. Varghese, N. Wang, S. Barbhuiya, P. Kilpatrick, and D. S. Nikolopoulos, "Challenges and opportunities in edge computing," in 2016 IEEE International Conference on Smart Cloud (SmartCloud), Nov 2016, pp. 20–26.

[15] A. Branover, D. Foley, and M. Steinman, "Amd fusion apu: Llano," IEEE Micro, vol. 32, no. 2, pp. 28–37, Mar. 2012.

[16] P. Greenhalgh, "Big.Little processing with ARM cortex-A15 & cortex-A7 - ARM White paper," ARM, Tech. Rep., 2011.

[17] S. Mittal, "A survey of techniques for architecting and managing asymmetric multicore processors," ACM Comput. Surv., vol. 48, no. 3, pp. 45:1–45:38, Feb. 2016.

[18] D. Miorandi, S. Sicari, F. D. Pellegrini, and I. Chlamtac, "Internet of things: Vision, applications and research challenges," Ad Hoc Networks, vol. 10, no. 7, pp. 1497 – 1516, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570870512000674

[19] T. Adegbija, A. Rogacs, C. Patel, and A. Gordon-Ross, "Microprocessor optimizations for the internet of things: A survey," IEEE Trans. on CAD of Integrated Circuits and Systems, vol. 37, no. 1, pp. 7–20, 2018. [Online]. Available: https://doi.org/10.1109/TCAD.2017.2717782

[20] "Cortus SAS – Advanced Processing Solutions," http://www.cortus.com, July 2017.

[21] A. Sehgal and N. Kehtarnavaz, "A convolutional neural network smartphone app for real-time voice activity detection," IEEE Access, vol. 6, pp. 9017–9026, 2018.

[22] A. Gordon-Ross, F. Vahid, and N. D. Dutt, "Fast configurable-cache tuning with a unified second-level cache," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 17, no. 1, pp. 80–91, Jan 2009.

[23] Y. Kora, K. Yamaguchi, and H. Ando, "Mlp-aware dynamic instruction window resizing for adaptively exploiting both ilp and mlp," in Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture, ser. MICRO-46. New York, NY, USA: ACM, 2013, pp. 37–48. [Online]. Available: http://doi.acm.org/10.1145/2540708.2540713

[24] A. Efthymiou and J. D. Garside, "Adaptive pipeline structures fo speculation control," in Proceedings of the 9th International Symposium on Asynchronous Circuits and Systems, ser. ASYNC '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 46–. [Online]. Available: http://dl.acm.org/citation.cfm?id=785169.785390

[25] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen, "Single-isa heterogeneous multi-core architectures: The potential for processor power reduction," in Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture, ser. MICRO 36. Washington, DC, USA: IEEE Computer Society, 2003, pp. 81–. [Online]. Available: http://dl.acm.org/citation.cfm?id=956417.956569

[26] V. K. Chippa, S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Approximate computing: An integrated hardware approach," in 2013 Asilomar Conference on Signals, Systems and Computers, Nov 2013, pp. 111–117.

[27] F. Samie, L. Bauer, and J. Henkel, "An approximate compressor for wearable biomedical healthcare monitoring systems," in Proceedings of the 10th International Conference on Hardware/Software Codesign and System Synthesis, ser. CODES '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 133–142. [Online]. Available: http://dl.acm.org/citation.cfm?id=2830840.2830855

[28] A. P. Chandrakasan, D. C. Daly, J. Kwong, and Y. K. Ramadass, "Next generation micro-power systems," in 2008 IEEE Symposium on VLSI Circuits, June 2008, pp. 2–5.

[29] S. Gollakota, M. S. Reynolds, J. R. Smith, and D. J. Wetherall, "The emergence of rf-powered computing," Computer, vol. 47, no. 1, pp. 32–39, Jan 2014.

[30] Y. Liu, Z. Li, H. Li, Y. Wang, X. Li, K. Ma, S. Li, M. Chang, S. John, Y. Xie, J. Shu, and H. Yang, "Ambient energy harvesting nonvolatile processors: From circuit to system," in 2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), June 2015, pp. 1–6.

[31] S. Senni, L. Torres, G. Sassatelli, and A. Gamatié, "Non-volatile processor based on MRAM for ultra-low-power iot devices," JETC, vol. 13, no. 2, pp. 17:1–17:23, 2016. [Online]. Available: https://doi.org/10.1145/3001936

[32] J.-M. Choi, C.-M. Jung, and K.-S. Min, "Pcram flip-flop circuits with sequential sleep-in control scheme and selective write latch," JSTS:Journal of Semiconductor Technology and Science, vol. 13, 02 2013.

[33] J. Ahn, S. Yoo, O. Mutlu, and K. Choi, "Pim-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture," in 2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA), June 2015, pp. 336–348.

[34] M. Kang, S. K. Gonugondla, M. Keel, and N. R. Shanbhag, "An energy-efficient memory-based high-throughput vlsi architecture for convolutional networks," in 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), April 2015, pp. 1037–1041.

[35] J. Crenne, R. Vaslin, G. Gogniat, J.-P. Diguet, R. Tessier, and D. Unnikrishnan, "Configurable memory security in embedded systems," ACM Trans. Embed. Comput. Syst., vol. 12, no. 3, pp. 71:1–71:23, Apr. 2013. [Online]. Available: http://doi.acm.org/10.1145/2442116.2442121

[36] P. Koeberl, S. Schulz, A.-R. Sadeghi, and V. Varadharajan, "Trustlite: A security architecture for tiny embedded devices," in Proceedings of the Ninth European Conference on Computer Systems, ser. EuroSys '14. New York, NY, USA: ACM, 2014, pp. 10:1–10:14. [Online]. Available: http://doi.acm.org/10.1145/2592798.2592824

[37] M. D. Hill and M. R. Marty, "Amdahl's law in the multicore era," Computer, vol. 41, no. 7, pp. 33–38, Jul. 2008.

[38] T. Y. Morad, U. C. Weiser, A. Kolodny, M. Valero, and E. Ayguade, "Performance, power efficiency and scalability of asymmetric cluster chip multiprocessors," IEEE Comput. Archit. Lett., vol. 5, no. 1, pp. 4–17, Jan. 2006.

[39] R. Kumar, D. M. Tullsen, N. P. Jouppi, and P. Ranganathan, "Heterogeneous chip multiprocessors," Computer, vol. 38, no. 11, pp. 32–38, Nov. 2005.

[40] T. Adegbija, A. Rogacs, C. Patel, and A. Gordon-Ross, "Enabling Right-Provisioned Microprocessor Architectures for the Internet of Things," in ASME International Mechanical Engineering Congress and Exposition, 2015.

[41] P. Kansakar and A. Munir, "Selecting Microarchitecture Configuration of Processors for Internet of Things (IoT)," IEEE Transactions on Emerging Topics in Computing, pp. 1–1, 2018.

[42] A. Butko, F. Bruguier, A. Gamatié, G. Sassatelli, D. Novo, L. Torres, and M. Robert, "Full-system simulation of big.little multicore architecture for performance and energy exploration," in 10th IEEE International Symposium on Embedded Multicore/Many-core Systems-on-Chip, MCSOC 2016, Lyon, France, September 21-23, 2016. IEEE Computer Society, 2016, pp. 201–208. [Online]. Available: https://doi.org/10.1109/MCSoC.2016.20

[43] "Genesys 2 Kintex-7 FPGA Development Board," https://www.xilinx.com/products/boards-and-kits/1-cfdwjq.html, July 2017.

[44] Wikipedia, "List of ARM microarchitectures," https://en.wikipedia.org/wiki/List_of_ARM_microarchitectures, March 2019.

[45] "OpenMP Application Program Interface - Version 4.0," https://www.openmp.org/wp-content/uploads/OpenMP4.0.0.pdf, 2013.

[46] A. Duran, E. Ayguadé, R. M. Badia, J. Labarta, L. Martinell, X. Martorell, and J. Planas, "Ompss: a proposal for programming heterogeneous multi-core architectures." Parallel Processing Letters, vol. 21, no. 2, pp. 173–193, 2011. [Online]. Available: http://dblp.uni-trier.de/db/journals/ppl/ppl21.html#DuranABLMMP11

[47] B. Nichols, D. Buttlar, and J. P. Farrell, Pthreads Programming. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 1996.

[48] J. Gustafsson, A. Betts, A. Ermedahl, and B. Lisper, "The Mälardalen WCET benchmarks – past, present and future," in International Workshop on Worst-Case Execution Time Analysis (WCET'2010), B. Lisper, Ed. Brussels, Belgium: OCG, Jul. 2010, pp. 137–147.

[49] L. Ma, L. Lavagno, M. T. Lazarescu, and A. Arif, "Acceleration by inline cache for memory-intensive algorithms on fpga via high-level synthesis," IEEE Access, vol. 5, pp. 18 953–18 974, 2017.

[50] N. S. Mokhtari, "Performance optimization of memory-bound programs on data parallel accelerators," Ph.D. dissertation, The Ohio State University, 2016.

[51] J. C. R. da Silva, F. M. Q. Pereira, M. Frank, and A. Gamatié, "A compiler-centric infra-structure for whole-board energy measurement on heterogeneous android systems," in 13th International Symposium on Reconfigurable Communication-centric Systems-on-Chip, ReCoSoC 2018, Lille, France, July 9-11, 2018, S. Niar and M. A. R. Saghir, Eds. IEEE, 2018, pp. 1–8. [Online]. Available: https://doi.org/10.1109/ReCoSoC.2018.8449378

[52] Z. Hu, A. Buyuktosunoglu, V. Srinivasan, V. Zyuban, H. Jacobson, and P. Bose, "Microarchitectural techniques for power gating of execution units," in Proceedings of the 2004 International Symposium on Low Power Electronics and Design, ser. ISLPED '04. New York, NY, USA: ACM, 2004, pp. 32–37. [Online]. Available: http://doi.acm.org/10.1145/1013235.1013249

[53] Q. Wu, M. Pedram, and X. Wu, "Clock-gating and its application to low power design of sequential circuits," IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications, vol. 47, no. 3, pp. 415–420, March 2000.

[54] J. Cong and B. Yuan, "Energy-efficient scheduling on heterogeneous multi-core architectures," in Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design, ser. ISLPED '12. New York, NY, USA: ACM, 2012, pp. 345–350. [Online]. Available: http://doi.acm.org/10.1145/2333660.2333737

[55] R. Nishtala, P. Carpenter, V. Petrucci, and X. Martorell, "Hipster: Hybrid task manager for latency-critical cloud workloads," in 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA), Feb 2017, pp. 409–420.

[56] J. K. V. Sreelatha, S. Balachandran, and R. Nasre, "CHOAMP: Cost based hardware optimization for asymmetric multicore processors," IEEE Transactions on Multi-Scale Computing Systems, vol. 4, no. 2, pp. 163–176, April 2018.