Journal of Systems Architecture 97 (2019) 42–53

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

The power impact of hardware and software actuators on self-adaptable many-core systems

André Luís del Mestre Martins[a], Rafael Garibotti[b], Nikil Dutt[c], Fernando Gehm Moraes[b,1,*]

[a] Sul-Rio-Grandense Federal Institute, IFSul, Brazil
[b] School of Technology, PUCRS, Brazil
[c] Donald Bren School of Information and Computer Sciences, University of California, Irvine, UCI, USA

ARTICLE INFO

Keywords:
Hardware and software actuators
Power consumption
Self-Awareness
Many-Core
Resource management

ABSTRACT

Many-core systems rely on the advantages of the latest Complementary Metal Oxide Semiconductor (CMOS) technologies to increase the number of cores. However, this improvement comes at the cost of higher power dissipation, which prevents full use of the chip. To continue improving performance on future many-core systems, Resource Management (RM) becomes imperative to handle multi-objective and conflicting requirements such as power, performance, resilience, among others. In this task, RM can use both hardware (e.g., dynamic voltage and frequency scaling) and software actuators (e.g., task remapping). However, the complexity of synchronizing available actuators to follow a particular goal while avoiding actuation overlapping is a remaining challenge. This paper evaluates the power impact of each actuator and provides insights that will help engineers develop appropriate resource management heuristics to improve self-adaptable many-core systems. A state-of-the-art comparison shows that no related work provides or details the same comprehensiveness of actuation methods concerning power consumption. Our proposal is validated in a many-core system described in a true clock-cycle accurate model. Regarding hardware actuators, the results show the power profiling at the core level and detail the contribution of each hardware component. Furthermore, results of software actuators evidence that task events present a more significant power impact on the ratio of active and idle cores changes.

1. Introduction

Despite the countless advantages brought by many-core systems regarding performance increase, a certain number of cores must remain off (dark) or slowed (grey) during the applications' execution due to power constraints [1]. These restrictions, known as Dark Silicon [1] and utilization wall [2], respectively, are more pronounced on recent technology nodes due to the higher number of integrated cores. Without respecting the *power capping*, i.e., limiting how much electricity a system can consume, the system becomes vulnerable or unreliable to several problems such as cooling, faults from thermal issues, and fast aging effects [3]. This growing concern with power constraints motivates research in the area of adaptive embedded systems [4], i.e., systems endowed with decision-making capacity, capable of making online decisions for better energy efficiency. In this regard, Resource Management (RM) has gained popularity thanks to its flexibility to handle power, performance, and other conflicting parameters to achieve system and application requirements.

RM drives many-core systems to follow a given objective like energy and performance by configuring the available actuators set, also called as knobs. The cooperation of distinct actuators can bring advantages to the system management [5]. However, the amount of possible system settings increases exponentially as the number of actuators grows. To further complicate, each core usually allows dynamic settings of their control knobs. As a consequence, a comprehensive actuators set increases the complexity of any RM design and makes the management of a many-core system much more challenging due to the amount of possible operating points.

Many-core systems that support RM must be hierarchically organized to properly distribute the actuators and allow scalability [6]. A self-adaptive RM employs a general framework for self-aware computing paradigm, such as autonomic computing [7] and *observe, decide, act* (ODA) [8]. Concerning ODA paradigm, *observation* is essential to provide an adequate measurement infrastructure by monitoring computation and communication resources, *actuation* adopts hardware and software mechanisms to meet the resource management goals. Lastly, *de-*

* Corresponding author.
E-mail addresses: almmartins@charqueadas.ifsul.edu.br (A.L.d.M. Martins), rafael.garibotti@pucrs.br (R. Garibotti), dutt@uci.edu (N. Dutt), fernando.moraes@pucrs.br (F.G. Moraes).

[1] The Author Fernando Moraes is supported by FAPERGS (17/2551-0001196-1) and CNPq (302531/2016-5), Brazilian funding agencies.

https://doi.org/10.1016/j.sysarc.2019.05.006
Received 20 February 2019; Received in revised form 23 April 2019; Accepted 25 May 2019
Available online 28 May 2019

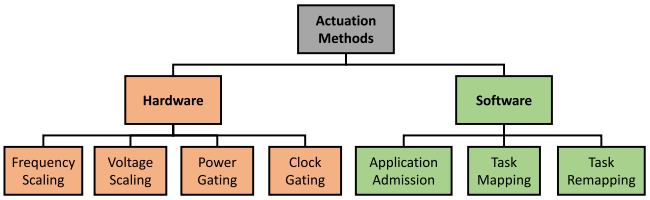

Fig. 1. Classification of actuation methods adopted in this work.

cisions adopt comprehensive algorithms to control the system actuators based on the observed data.

The ODA paradigm requires self-awareness and self-configuration from the system to enable the adaptation policies [9,10]. In this work, self-awareness means that the system can observe itself through virtual or physical sensors, and self-configuration is the ability to adjust the available system actuators to update system settings. The biggest challenge remains in how to control the actuators set of a many-core system in a coordinated way to provide the required adaptability to improve its energy efficiency. Note that energy-efficient self-adaptive techniques are also a concern in other domains, such as High-Performance Computing (HPC) [10,11].

To pave the way for designing an energy-efficient many-core system, the objective of this paper is to reveal which actuators best fit for a given scenario. This paper focuses on the *actuation* component of the ODA paradigm, and presents the following contributions: (*i*) provide a protocol for each actuation method; (*ii*) pinpoint the advantages and disadvantages of each actuation method; and (*iii*) provide recommendations for the use of each actuator according to its power impact. To validate our work, we model an actuation platform that includes several hardware and software actuators to support RM decisions. The proposed platform allows individual control of each layer by enabling the hierarchical approach under the ODA paradigm. It provides insights for future RM designs take smart decisions according to heuristics that meet the multi-objective purposes [12]. Further, no other work provides or details the same comprehensiveness of actuation methods concerning power consumption.

The rest of this paper is organized as follows. Section 2 introduces the actuation methods. Next, Section 3 presents the extended reference many-core platform with support for actuation methods. Section 4 shows the design and model of the hardware actuators. Following, Section 5 presents the protocol of the software actuators. Section 6 evaluates the power impact of the actuation methods, as well as revealing the benefits of combining some actuators. Section 7 overviews related work regarding the RM comprehensiveness. Finally, Section 8 points out conclusions.

2. Actuation methods

This section presents the actuation methods adopted in this work. In the literature, several specific actuators are presented, such as *pause/unpause app, kill app, increase/reduce power cap*, and *set power cap* [10]. However, our choice was based on their relevance and coverage. Among the chosen ones, the actuation methods were classified based on two properties: (*i*) the latency for enforcing an actuation; and (*ii*) the impact delivered by an actuator [5]. Further, actuators can fall into two implementation categories, as shown in Fig. 1. *Hardware actuators* are usually fast, although they usually have a limited overall impact (e.g., resources and power) at the system point of view due to inherent hardware limitations. On the other hand, *software actuators* are slower than the hardware ones to take effect. However, they usually are more flexible and enforce a higher impact on the system.

The hardware actuation methods include:

– *Frequency Scaling* (FS): technique used to increase or decrease the frequency of a hardware component;

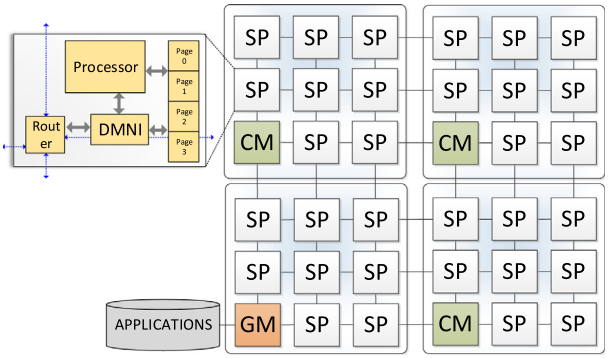

Fig. 2. A 6x6 instance of the reference many-core platform, organized in four 3x3 clusters.

– *Voltage Scaling* (VS): technique used to increase or decrease the voltage of a hardware component;
– *Power Gating* (PG): technique used to shutdown the power supply of an idle block of a circuit [13];
– *Clock Gating* (CG): technique used to disable the clock switching to stop unnecessary gate activity [14].

On the other hand, software actuators are modeled using communication protocols. Any software actuation method includes a set of messages, each one related to a specific service to model the behavior of the given actuator. The software actuation methods include:

– *Application Admission* (AA): protocol used to adapt the application parallelism to the available resources on the system;
– *Task Mapping* (TM): protocol used to assign a task of the incoming application to the system, usually based on a mapping heuristic [15];
– *Task Remapping* (TR): protocol used to employ task migration. It deals with the availability of resources dynamically.

A demonstration of how these actuators can be used in conjunction is as follows: at the application level, the manager evaluates if the system has enough power and resources for new incoming applications [16]. Once the manager allows the application to execute, TM finds the most suitable area to place application tasks. Further, TR [17], FS and VS may adapt the system and applications settings according to the system status [18] or goal switching. To support adaptability, the RM controls the knobs available in the system at runtime to meet the goals embedded in the management algorithms. Therefore, comprehensive and adaptive management for many-core systems should include not one or two, but several actuation methods. In this sense, this work seeks to cover this challenge by evaluating the power impact presented in such actuation methods and the best way to use them.

3. Many-core platform

This section describes our many-core platform, which is used to model the actuation methods presented in the previous section. The adopted reference many-core architecture is based on an open-source project, named HeMPS, which is available online [19]. The Processing Elements (PEs) are interconnected through a 2D-mesh Network-on-Chip (NoC). Each PE contains a 32-bit Reduced Instruction Set Computer (RISC) processor, a local scratchpad memory, a packet-switching router and a DMNI module [20], which combines the functions of a Network Interface (NI) with Direct Memory Access (DMA) capabilities.

Fig. 2 illustrates a 6x6 HeMPS instance. Despite the same hardware for all PEs, they have different roles in the system, i.e., a PE is either a manager or a slave. Further, the system is hierarchically organized into *clusters*. In this context, a cluster is a virtual region of the many-core platform that contains a manager PE and a set of slave PEs. The cluster size is a design-time parameter, although a cluster can borrow resources

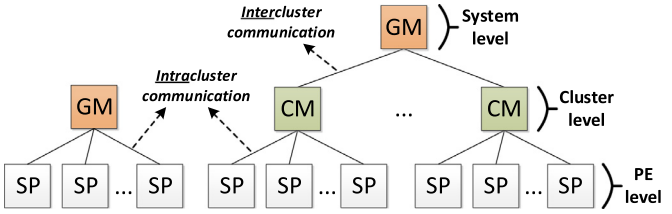

Fig. 3. Hierarchical organization, where the GM manages the system, the CMs manage a set of SPs and SPs execute application tasks.

from other clusters at runtime when there is no available slave PEs in the cluster to execute a given application.

This work *extends* the reference platform [19] by increasing the capabilities of the manager PE to handle multiple actuators concurrently. The Operating System (OS) running on top of each PE defines its role in the hierarchical organization. A PE can assume the following roles:

- *Cluster Manager* (CM): resource manager at the Cluster-level. It executes the decision management regarding hardware and software actuators, such as task mapping and dynamic voltage and frequency scaling (DVFS), i.e., a combination of VS and FS;
- *Global Manager* (GM): resource manager at the System-level. It receives application execution requests via the external interface (application repository) and decides which clusters execute the applications. The GM also acts as a CM, managing the PEs belonging to its cluster;
- *Slave* (SP): slave PEs that execute the applications tasks. Each SP executes a multi-task OS, enabling the concurrent execution of tasks.

Fig. 3 overviews the hierarchical organization of the many-core system, highlighting the different levels and the communication pattern between PEs. The exchanged messages related to the system management may be intra- or inter-cluster. SP belonging to a cluster communicates with the manager PE of its cluster, which defines an intra-cluster communication. Similarly, the inter-cluster communication occurs when CMs communicate with the GM.

The actuators set is modeled to fit in the hierarchical organization shown in Fig. 3. The higher the actuation impact and latency, the higher its hierarchical level. Thus, the actuators set is organized as follows:

- *System-level*: application admission;
- *Cluster-level*: task mapping and task remapping;
- *PE-level*: frequency scaling, voltage scaling, power gating and clock gating, i.e., all hardware actuators.

It is worth mentioning that the architecture and actuation methods are *orthogonal* concepts. Architectural features, like caches, out-of-order execution, or different hardware affect the communication at both processor and NoC levels [21]. The actuators capture the computation and communication loads, firing a given action according to the processor load and NoC traffic. Thus, the methodology presented in this work is *independent* of the reference many-core platform chosen. The reason for adopting this many-core platform is the easy reproducibility of our experiments by other research groups and the accurate validation of the power impact of the modeled actuators.

4. Hardware actuators

This section details four hardware actuation methods evaluated in this work, as shown in Fig. 1. Although these hardware actuators are not new, the purpose of this section is to demonstrate how these methods are implemented on a reference many-core platform, as well as to show its benefits and drawbacks.

The model of the frequency scaling (Section 4.1) requires modifications on the original PE structure to cope with different frequencies and keep the accuracy regarding the clock cycle. To guarantee realistic

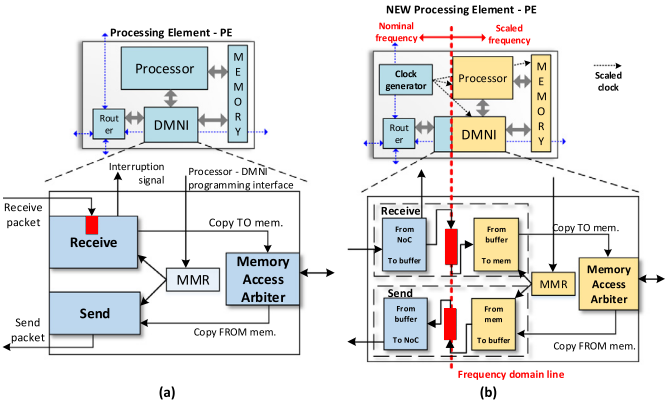

Fig. 4. (a) Original PE and (b) novel PE with FS support, which integrates a clock generator hardware and modifies the DMNI.

DVFS support, the model of the voltage scaling (Section 4.2) considers the hardware overheads (e.g., latency and energy), standard cells library characterized for distinct supply voltages provided by the foundry, and the delays inherent of the voltage scaling for establishing a correct DVFS protocol. The last two sections present actuators commonly found in RM designs to deal with power capping issues. While Section 4.3 describes the PG model that follows the same assumption used by some works [22,23]; Section 4.4 introduces the CG model, which is created through native features of the reference many-core platform.

4.1. Frequency scaling model

Fig. 4 illustrates the hardware modifications on the PE for frequency scaling support. The frequency scaling actuates only on the processor, memory, and DMNI module. The main goal is to enable processors to work at different frequencies while the NoC transmits the message by using the *nominal frequency*. The reason for transmitting messages at the nominal frequency is to avoid PEs running at higher frequencies stall due to PEs running at lower frequencies. The novel PE includes a clock generator that creates the *scaled frequency* from the nominal frequency. Fig. 4 shows the frequency domain line that separates the hardware blocks running at the nominal frequency (blue color) and the ones at a scaled frequency (gold color).

The DMNI synchronizes the hardware modules working at different frequencies. The original DMNI (Fig. 4a) has a `Send module` responsible for reading the data from memory and converting it to a message for sending to the network. The `Receive module` reads the message from the NoC and copies them to the memory. The new version of DMNI (Fig. 4b) has two bisynchronous buffers included in the `Send/Receive modules` to synchronize the DMNI. Both modules are divided into two regions, one running at the nominal frequency and the other one running at the scaled frequency. The `Receive module` reads the message from the NoC at the nominal frequency, and then writes the flits into the internal bisynchronous buffer. If this buffer is not empty and the memory is ready for writing, the receiver copies the message from the buffer to the memory by using the scaled frequency. The sending process is similar to the receiving process, the difference is in the opposite direction of the data flow, i.e., read from memory and send to the network.

Our many-core employs this FS model as one power knob to provide adaptability concerning power constraints. However, this actuator has a drawback, the inherent hardware overhead brought by the implementation of the clock generator and the bisynchronous buffer at the DMNI. Furthermore, the insertion of buffers in the DMNI penalizes the execution time of the applications on average at 6.55%.

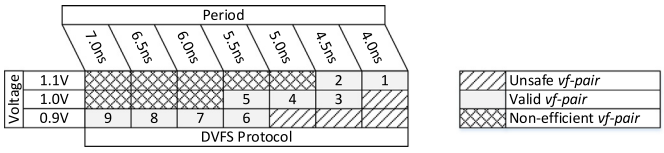

Fig. 5. The DVFS protocol shows the valid voltage and frequency pairs.

4.2. Voltage scaling model

A voltage regulator is an analog circuit that allows voltage scaling in a system. To model the voltage scaling, we characterized the reference many-core platform with supply voltages of 1.1V, 1.0V and 0.9V concerning the standard cells provided by the foundry (65nm CMOS technology). Note that this methodology is technology agnostic, i.e, other technology nodes can be used and our purpose is to validate the concept in order to produce meaningful results. First, the processor netlist obtained from a characterization flow (using the nominal *vf-pair*: 1.1 V–4 ns) is evaluated for 1.0V and 0.9V supply voltages. Then, it is verified the delay for reading the memory. The minimum period to obtain a zero or positive time slack concerning processor and memory is 4.479 ns and 5.229 ns, for 1.0 V and 0.9 V, respectively. Finally, we evaluate the router netlist for the same supply voltages. The goal is to ensure that the router can keep a positive slack at any supply voltage since the frequency scaling does not affect the router.

Fig. 5 defines the DVFS protocol by linking the minimum period for scaling the voltage safely with the frequency range generated by the clock generator. The numbers in the yellow boxes define valid *vf-pairs*. The ascending order establishes the protocol to scale the *vf-pair* down, while the descending order is the protocol to scale the *vf-pair* up. The system always starts at the nominal *vf-pair*.

In general, coarse-grain voltage regulators present latency in the order of milliseconds while fine-grain latency is lower than hundreds of nanoseconds [24]. On the other hand, the power overhead from on-chip voltage regulators to support fine-grain voltage scaling is non-negligible [25]. Due to the low latency of fine-grain voltage regulators and the feature of frequency scaling at the PE-level, we model a fine-grain (PE-level) voltage scaling which assumes that the latency of a voltage scaling (up or down) is 100 ns (i.e., 25 clock cycles at the nominal frequency), and the power overhead from on-chip voltage regulators increase the PE power in 10%, as indicated by Choi et al. [26].

4.3. Power gating

Power gating is a power actuator widely employed in many-core systems, mainly to mitigate Dark Silicon [1] and utilization wall [2] problems. PG provides a more significant power impact than DVFS because it is the only mechanism that eliminates leakage [13,27]. PG is usually deployed when no tasks are running in the PE. However, the PE needs to stay off long enough to compensate the time and power overheads to wake it up back. For example, on x86 processors, the reported wake up delay is in the order of a few microseconds [28]. As this proposal assumes applications can start at any moment, integrating the PG model would require predictions concerning how many time the PE will be on (active) or off (inactive). For this work, PG follows a standard assumption found in the literature [23,29,30]: if the PE is running no tasks, it is considered power-gated.

Although the PG is presented as a PE-level actuation, the process of shutting down a target PE may include another PE. In addition, the vf-pair settings can use another PE as well, so it is not wrong to see the management of this hardware actuator as a Cluster-level problem or even a System-level problem. For cases where two PEs are related to perform a hardware actuation, a source PE decides when a target PE is on and off, and its vf-pair. The source PE can be either a manager PE (i.e., GM or CM) or an SP running a task with power management capabilities. Therefore, a communication protocol between a source PE and a target PE [31] allows DVFS and PG decisions involving two or more PEs. In other words, the knob for performing a DVFS and a PG is at PE-level, however, the order to set the knob can come from another PE.

4.4. Clock gating

Clock gating is an actuator suitable for dealing with idle times when the PE is executing tasks due to the common short duration of idle times. Although the power impact is smaller than PG, it is significant compared to DVFS. The proposed CG model affects only the processor and the memory. The processor supports clock hold, i.e., when the processor is idle, the clock signal is disabled, saving dynamic power. Since the characterization computes dynamic power from the instructions counters, no changes are required to model CG for the processor. Similarly, the dynamic memory power comes only from load and store operations. Therefore, when memory operations are not occurring, the memory is considered in idle and no dynamic power is accumulated during idle periods. The router is continuously spending dynamic power, in the active or the idle state. The timing overhead from CG is considered negligible, and it is not inserted into the model.

Due to the hierarchical organization of the adopted many-core platform, the CG technique is more suitable to deal with the power consumption within a cluster, e.g., we would have a negligible timing overhead to map a new task. Instead, the PG technique is applied to clusters that are not running an application.

5. Software actuators

This section describes the communication protocol required to synchronize the software actuators. It is important because software actuators use message passing communication to invoke services. The communication protocols satisfy the hierarchy sorting for the actuators set, i.e., the application admission (Section 5.1) is a System-level actuator, while task mapping (Section 5.2) and task remapping (Section 5.3) are Cluster-level actuators. Note that communication protocols that allow fine-grain individual and independent hardware actuators [31] are required when a manager PE performs the decision on the PE-level settings.

In regard to software actuators, application admission and task mapping are correlated problems. Once the manager allows an application admission, the mapping of all tasks of this application is enabled as well. As a task graph composes one application, then the task mapping is considered as an inner problem of the application admission. This work distinguishes application admission and task mapping as two instances of the same problem, so the first is a System-level problem and the second is a Cluster-level problem. Also, we have chosen to further detail these software actuator protocols for two reasons. First, usually, these protocols are not properly discussed in the literature. Second, the primary concern of the available proposals are the heuristics and not the protocols. For example, Singh et al. [17] survey a broad set of task mapping heuristics for many-core systems, but not the protocols to deploy them.

Depending on the configuration chose (e.g., the number of PEs and clusters), the hierarchical organization of the many-core platform (as shown in Fig. 2) may result in clusters without enough resources to receive an entire application (as will be seen in Section 5.2) or the need to move an application to a free processor (as will be discussed in Section 5.3). In both cases, the absence of resources in a given cluster causes the OS of the cluster manager to have to request to the neighboring managers a processing element, and this process is called *reclustering* [32]. The reclustering process implies a set of messages exchanged between the manager processors in order to lend resources, allowing the mapping/remapping procedures. This process, if necessary, has a negligible impact on the mapping/remapping procedures because it consists of few messages exchanged between managers.

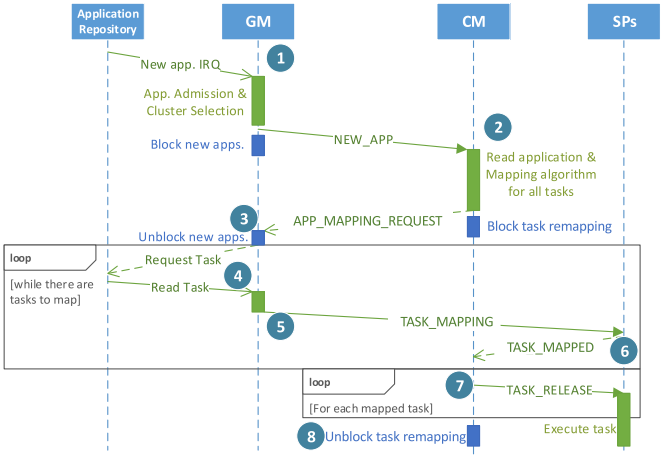

Fig. 6. Diagram of the application admission and task mapping protocols.

5.1. Application admission

The Application Admission assigns an application task graph to a cluster. The services to synchronize the AA are as follows:

- `NEW_APP`: message sent from the GM to a CM for notifying that a new application was assigned to the cluster. This message carries the application task graph and design-time data. The CM maps all applications' tasks after receiving this message;
- `APP_MAPPING_REQUEST`: message sent from a CM to the GM for notifying that the cluster is ready to receive the tasks from the incoming application and requires the TM protocol. This message carries the position of each task of the incoming application.

Fig. 6 presents a sequence diagram for admitting one application. An application from the Application Repository can enter at any time into the system by setting a hardware interruption to indicate to the GM the admission request (step 1). Next, the GM runs an algorithm to *decide* if the system can admit the application and, in case of success, selects the cluster to allocate the application. At step 2 the GM sends a `NEW_APP` message to the chosen CM and blocks the interruption for new applications. After receiving the `NEW_APP` message, the CM triggers the task mapping algorithm for the incoming application. Once the CM *decides* the mapping of all tasks of the application, the CM sends `APP_MAPPING_REQUEST` message to the GM (step 3). After processing the `APP_MAPPING_REQUEST` service, the GM unblocks the interruption for new applications and starts the task mapping protocol (steps from 4 to 8 are presented in Section 5.2).

5.2. Task Mapping

The Task Mapping protocol follows the Application Admission protocol. This protocol coordinates the assignment of all tasks to their SPs, once the mapping decision for all tasks was already defined. Because the TM transmits the object code of all application tasks to the SPs of a cluster (i.e., a considerable communication load is about to start), any task remapping is temporarily blocked. The services to synchronize the TM are as follows:

- `TASK_MAPPING`: it loads the task object code into the memory of the SP (message direction: GM to SP);
- `TASK_MAPPED`: it notifies the CM that a task was successfully loaded into an SP (message direction: SP to CM);
- `TASK_RELEASE`: it releases the mapped task (message direction: CM to SP).

Once the Application Admission finishes, a loop to map the tasks begins (Fig. 6). The GM reads a task from the Application Repository (step 4) and sends a `TASK_MAPPING` message to an SP (step 5). The GM is aware of the task mapping because this information was embodied in the `APP_MAPPING_REQUEST` message. Next, when the SP receives the task to execute, it sends a `TASK_MAPPED` message to the CM to notify that the task was successfully mapped (step 6). When the CM receives all `TASK_MAPPED` messages, it sends `TASK_RELEASE` messages to release the mapped tasks (step 7), i.e., the new application starts. Finally, after the last received `TASK_RELEASE` message, the CM unblocks any task remapping to finish the TM protocol (step 8).

Although the Task Mapping protocol is classified as a Cluster-level actuation, the GM plays a relevant role in the protocol due to the exclusive access to the Application Repository. Furthermore, the GM sends `TASK_MAPPING` messages directly to the SPs of any cluster. This design choice comes from two reasons: (*i*) to avoid the transmission of the object code first to the CM and then to the SPs; and (*ii*) due to the adoption of XY routing, e.g., if all object codes were transmitted to the CM, network congestion (hotspots) would occur.

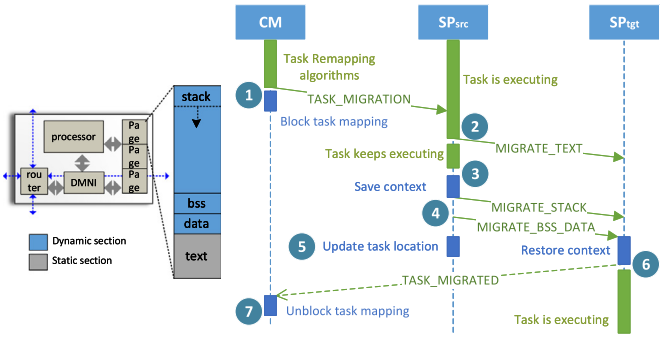

Fig. 7. Diagram of the task remapping protocol.

5.3. Task Remapping

The Task Remapping protocol, also know as task migration, is an essential feature to support adaptability because it allows remapping of the application at runtime. The TR goal is to move a task from a source SP (SP_{src}) to a target SP (SP_{tgt}). TR employs a low latency protocol for many-core systems with distributed memory, because it requires neither checkpoints nor task code replication as well as allows task migrations in parallel [33].

The typical layout of a memory page loaded with a task contains read-only and read-write sections. The read-only section is the task object code (`text` in Fig. 7). The read-write sections are global variables (`data` and `bss`) and the `stack` area (Fig. 7). The TR protocol migrates all sections of the memory page. The services to synchronize the TR are as follows:

- `TASK_MIGRATION`: it notifies the SP_{src} which task should migrate to SP_{tgt} to initialize task remapping (message direction: CM to SP_{src});
- `MIGRATE_TEXT`: it migrates the `text` section (message direction: SP_{src} to SP_{tgt});
- `MIGRATE_STACK`: it migrates the `stack` section (message direction: SP_{src} to SP_{tgt});
- `MIGRATE_BSS_DATA`: it migrates the `bss` and `data` sections (SP_{src} to SP_{tgt});
- `TASK_MIGRATED`: it notifies the CM that a task was successfully remapped from SP_{src} to SP_{tgt} (message direction: SP_{tgt} to CM).

A *decision* algorithm starts the TR protocol at a CM. The CM sends the `TASK_MIGRATION` message to SP_{src} and blocks both application admission and task mapping protocols while there are ongoing migrations (step 1). Once the SP_{src} is aware of the remapping request, SP_{src} can send a `MIGRATE_TEXT` message to migrate the read-only part of the task code, without blocking its execution (step 2). Next, SP_{src} chooses

Table 1
Summary of the many-core evaluated in this work.

Cluster sizes	3x3; 4x4; 6x6
NoC	2D mesh topology, input buffering, round-robin arbitration, XY routing
Communication	Bidirectional 32-bit links
CPU core	32 bit, 3-stage pipeline, MIPS ISA
Operating Systen (OS)	in-house OS, LST scheduling (soft RT), API for message exchange
Tile local private RAM	scratchpad memory model, 128kB – OS and tasks, true dual-port
Memory characterization	CACTI tool [34]
Target technology	65 nm CMOS bulk

an appropriate moment to stop the task and save the context (step 3) to enable the migration of the read-write memory segments (step 4). After sending the `MIGRATE_STACK` and `MIGRATE_BSS_DATA` messages, SP_{src} updates the new task location for all communicating tasks if they exist (step 5). In parallel, SP_{tgt} restores the context and can proceed the task execution after reporting to the CM the successful finish of task remapping (step 6). After receiving the `TASK_MIGRATED` message, the CM unblocks all task operations (step 7).

The reason to block task mapping while remapping a task is the same: avoid unnecessary network congestion and ensure synchronization. Although a concurrency between TR and TM requests is unlikely to happen, the CM guarantees only one at a time running on the cluster.

6. Results

This section evaluates the power impact of hardware and software actuators on self-adaptable many-core systems. The purpose of the experiments is to reveal which actuators best fit for a given scenario. For example, hardware actuators, such as *Clock Gating*, take advantage of the idle times of processors when they have tasks that are waiting for data from other PEs to reduce power consumption. On the other hand, software actuators have a systemic action, e.g., consider an application that if admitted would violate the power cap, the *Task Remapping* can be used to reduce the system power in order to accept the incoming application by joining tasks in the same processor. In addition, we also provide recommendations for the use of each actuator according to its power impact, bringing insights to guide new heuristics that assist in the adaptive decisions of RMs. In this regard, Section 6.1 summarizes the many-core architecture set evaluated throughout all experiments. Section 6.2 shows the first power results obtained from hardware actuators. Next, Section 6.3 highlights the energy improvements brought by software actuators. Finally, Section 6.4 provides insights into the best fit of each actuator and shows the results from combinations of some actuators.

6.1. Experimental setup

Table 1 gives the details of the architecture set used throughout these experiments. These configurations were used in the extended reference platform with support to the actuation methods. Further, we use conventional technology (65 nm) to facilitate the replication of the experiments by other researchers.

For the power impact exploration, a critical step is the communication within a cluster in which the network topology selection has a direct impact on the overall system power consumption. However, the NoC topology exploration results are out of the scope of this paper. In this sense, we adopted 2D-mesh topology. Besides being one of the most used topologies in literature, routing in a 2D-mesh is easy, resulting in potentially small area footprint routers, short clock cycle, and overall scalability [35]. Furthermore, 2D-mesh well matches the planar, regular layout of a cluster-based design, which can increase the scalability of self-adaptable many-core systems.

In this work, we consider both synthetic applications and a set of workloads from scientific to multimedia computing domain. These

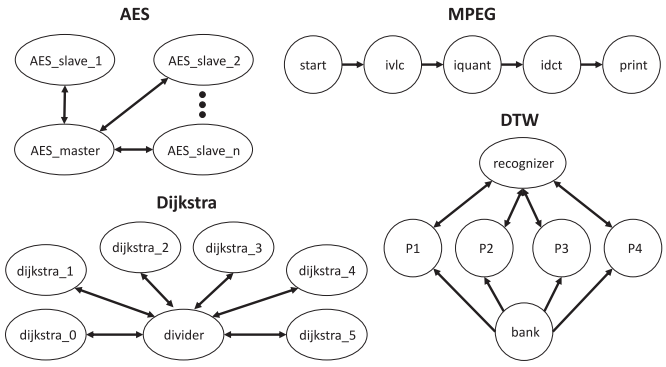

Fig. 8. Communication graphs of the chosen workloads.

workloads were selected according to their profiles to deeply evaluate the power impact of chosen actuators, e.g., AES, MPEG, Dijkstra, and DTW.

Fig. 8 shows the communication graphs of the chosen workloads, highlighting the different characteristics of each task. While MPEG tasks are interdependent, AES and Dijkstra workloads present bottlenecks as they are parallelized by centralizing responses into a single PE. Finally, DTW is dependent on 2 tasks, which work concomitantly.

6.2. Power impact of hardware actuators

This section evaluates the power impact of the four designed hardware actuation methods. As all have been implemented at the PE-level, a correct fine-grain tuning is essential to have an effective power consumption reduction.

6.2.1. DVFS actuator

The first experiment illustrates the maximum and minimum power consumption at the PE-level using the DVFS actuator. Once the DVFS protocol is defined (Fig. 5), the PE is characterized by each supply voltage, considering the smallest periods (*vf-pairs* 1, 3, 6). The router always works at the nominal frequency (i.e., 4 ns). The processor and the memory consider the power per instruction and per operation (read/write), respectively. Therefore, it is not necessary to characterize the modules for each frequency. As a result, each PE component has three look-up tables (i.e., 1.1 V, 1.0 V, 0.9 V), obtained from the characterization flow.

Fig. 9 presents the experimental setup to determine upper and lower bounds values related to the power consumption of the SPs. This experiment uses a synthetic application. While the central SP (SP_4) executes a CPU-bound task, with a uniform distribution of instructions classes. The SP_4 neighbors execute communication-bound tasks, generating traffic traversing SP_4 router. With this scenario, the consumption of SP_4 defines the *maximum power* (p_{max}) value that an SP can consume in an epoch.[2] On the contrary, SPs at the corners (2, 6, 8) spend mostly leakage power

[2] *Epoch*: corresponds to a monitoring period, defined as a hardware interruption.

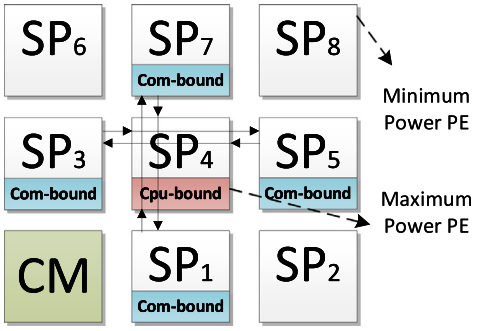

Fig. 9. A 3x3 many-core system executing synthetic tasks mapped to maximize the average power dissipation of the central SP, and minimize the power dissipation of the SPs at the corners.

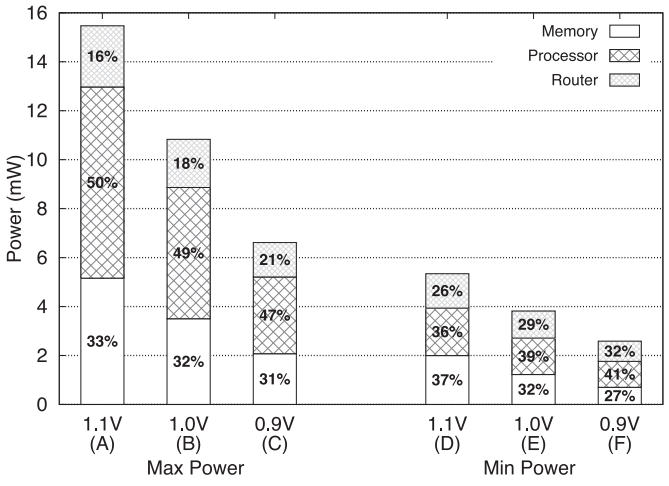

Fig. 10. Power profiling of the SP for all supply voltages. The total power (y-axis) corresponds to the power consumption in an epoch of 1 ms.

because these PEs have no tasks to execute (processors in hold state), and there is no traffic traversing the routers of these SPs (routers in idle mode). Note that these routers have only three ports so that the router consumption reduces. Therefore, the power consumption at these SPs defines the *minimum power* (p_{min}) value that an SP can consume in one epoch.

Fig. 10 details the power consumption for each SP component considering the scenario presented in Fig. 9. The histograms assume three supply voltages for an epoch equal to 1 ms. The histograms show the contribution of the three modules in the power consumption, considering *vf-pairs* as 1, 3 and 6. The comparison of p_{max} and p_{min} highlights the effect of the leakage power.

The p_{max} histograms (left part of the Figure – *max power*) show the contribution of the three modules in the power consumption: 50% processor, 30% memory, and 20% router. As the voltage reduces, the portion due to router power increases because only the processor and the memory works on the scaled frequency. Therefore, in the epoch, the number of executed instructions and memory accesses reduces when the frequency is scaled down. The p_{min} histograms present a distinct behavior, with an increased consumption by the routers due to the lack of clock hold. p_{min} is approximately one-third of p_{max} when comparing the bars of the same voltage. The comparison of p_{min} at 0.9 V (processor executing CPU-bound tasks) and p_{max} at 1.1 V (processor in hold state) evidences the effect of the leakage power. The p_{max} at 0.9 V is only 18% higher than the p_{min} at 1.1 V.

The evaluation of the PE power dissipation at Fig. 10 points out to the developer of the adaptation heuristics how much a given PE can benefit from the DVFS technique. For example, if the application running on

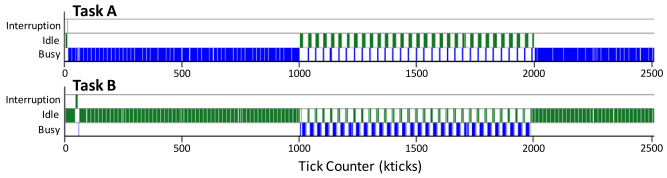

Fig. 11. Processor scheduling zooms in the task phases of two tasks running into two different PEs, highlighting the communication between tasks.

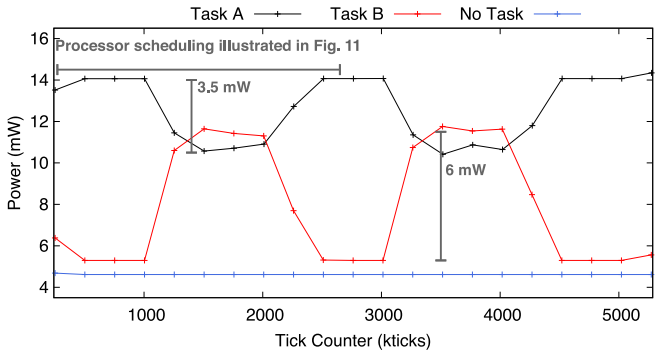

Fig. 12. The power curves show the impact of CG relies on task phases, where "no task" curve presents the power of an idle SP.

a particular PE can have its performance reduced, the dissipated power can reduce by up to 50% using DVFS (the relationship between columns C and A in Fig. 10). By placing a PE in an idle state, the consumption can reduce by up to 5 times (relation between columns A and F of Fig. 10).

6.2.2. Power and clock gating actuators

This section shows the power consumption variation regarding the task phases at the PE-level when CG is applied. Further, it reveals the opportunities that PG actuators can bring for self-adaptable many-core systems.

This experiment is also based on a synthetic application and consists of running two tasks. Both tasks have distinct moments of computation and communication. In the communication period, the SP checks if the processor is in idle mode and applies the clock gating actuator. Fig. 11 depicts the scheduling of these two tasks running at two distinct SPs. The blue bars are the moments when the processor is busy, and the green bars are idle periods. Task A corresponds to a computation-intensive task. At the end of an iteration, the results are sent to task B (between 1000 and 2000 kticks in Fig. 11), where there is an alternation between active and idle states.

To analyze the effectiveness of the RM, Fig. 12 shows the power profile of tasks A and B including the CG model. At the beginning of the execution, from 0 to 1000 kticks, Task A is in a busy phase spending around 14 mW and Task B stays mostly in idle state consuming around 5.5 mW (this phase corresponds to the first one third of Fig. 11). After this period, from 1000 to 2000 kticks, both tasks alternated busy and idle states, exchanging messages (second third of Fig. 11). Note that the idle phase of both tasks generates lower power due to the CG actuator, while the busy phase creates peaks of power consumption.

The third curve (blue line points) at Fig. 12 illustrates the power of an SP when no tasks are running (i.e., the minimum power of an idle SP, 4.9 mW). If PG would be applied, this consumption could be eliminated, at the cost of the latency to start the PE.

This experiment shows that CG reduced the dynamic power consumption by 3.5 mW and 6 mW for tasks A and B, respectively. Thus, CG is a fine-grain actuator, which can act during the different phases of a task execution by taking advantage of the idle periods of the processor.

The results so far presented by the hardware actuators are as expected, according to the literature [23,29,36]. However, the purpose

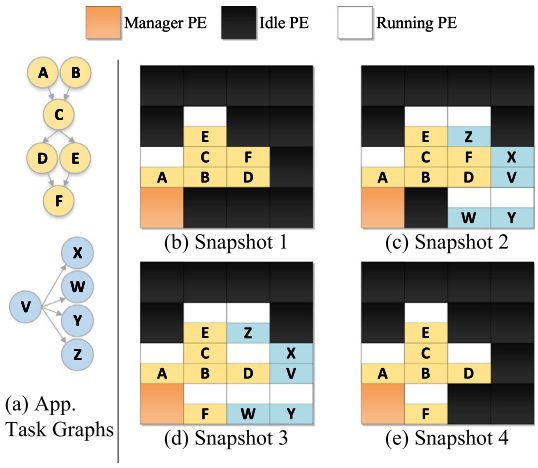

Fig. 13. Snapshots taken to show the application mappings at important moments explained by Fig. 14.

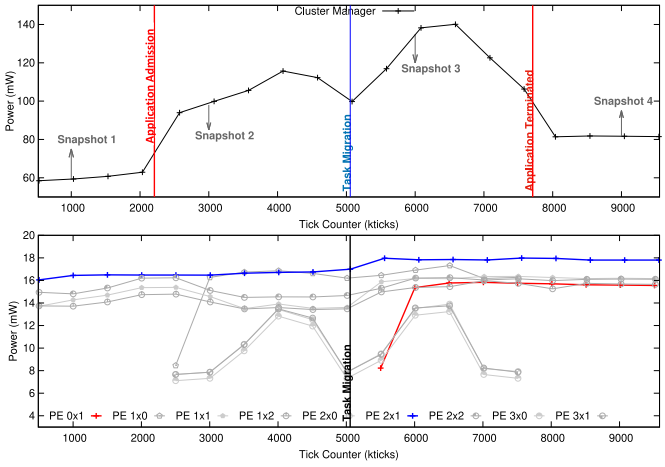

Fig. 14. Example of the power impact of software actuators in a 4x4 cluster. The top graph shows the power at the cluster level, and the bottom one illustrates the power in each PE.

of these experiments was to illustrate upper and lower consumption bounds for a processing element, for a given technology. Thus, decision algorithms, using these values obtained from the characterization process, can modify DVFS parameters so that the system reacts proactively, respecting the power cap as a function of future actions, such as the admission of a new application.

6.3. Power impact of software actuators

As stated at the beginning of this work, software actuators usually have a larger impact than the hardware ones. Fig. 13 presents the system occupation for this experiment, using a 4x4 cluster, where each PE may execute up to two tasks. Fig. 14 illustrates the variation of power and resources for each software actuator. The bottom graph presents the individual power consumption of each PE, and the top graph shows the cluster power. The simulation aims to highlight task events disturbing the power and resources. For the sake of readability, the PEs running no tasks are considered clock-gated in this example, because they are all part of an active cluster. The choice of the CG over the PG actuator is motivated by the latency brought by the latter, as discussed in Section 4.4.

The simulation starts with six tasks running in four PEs (snapshot 1). Due to an Application Admission, five new tasks occupy additional four PEs and make the power increases around 80% (snapshot 2). In this sense, the bottom graph in Fig. 14 illustrates the power variation felt by each PE due to the AA actuator. A TR is the second software actuator triggered by the RM. The bottom graph in Fig. 14 also show the power impact caused by a task that migrates from a PE running two tasks (PE 2x1 - blue curve) to an idle PE (PE 0x1 - red curve) – snapshot 3. Although the number of tasks and resources is still the same after the task remapping, the power in PE 2x1 increases because its utilization increases. Finally, the end of an application execution decreases the PE utilization and consequently the power as well (snapshot 4).

In general, Application Admission and Task Mapping increase the power. However, the amount of this growth relies on the number of tasks, the number of PEs used to map the tasks and the task characteristics (e.g., if the task is communication-bound or CPU-bound). Although both applications use the same number of PEs (four), Fig. 14 shows that the first admitted application requires more power than the second one because the former has more tasks (six versus five) and the average power of its tasks is higher, according to the power curves shown in the bottom graph. Concerning task remapping, the power increases when occurring task remapping from a PE running multiple tasks to an idle PE, similar to the presented in Fig. 14. As the opposite, the power decreases when a task remapping releases a PE and migrates to another busy PE. Similarly to task mapping, the amount of the power variation due to a task remapping is variable and relies on the task characteristics. If the number of idle and busy PEs does not change, the impact on the power consumption of a task remapping is small. Thus, task remapping can be best used as a power knob in the following situations: (*i*) when tasks are divided to run in distinct PEs to make the application achieve better performance; or (*ii*) tasks are joined to share PEs while releasing other PEs to reduce power.

Besides the power variation due to different software actuators, Fig. 14 also illustrates how the task phases affect the power consumption. In the bottom graph, the power of some tasks exhibits a constant behavior while other tasks have a period behavior with peaks and valleys of power. As a consequence, the overall power (top graph) is not constant even though no actuation occurs.

Given the impact on the power due to the admission of new applications, we suggest that RM policies should receive a design-time estimation of the applications' consumption, in such a way to avoid power capping violations. Knowing the impacts of the hardware actuators (DVFS, CG, PG), it is possible to admit a new application allowing a higher power than the cap. Thus, it is clear that hardware actuators should work together with software actuators, to manage the system power dissipation.

6.4. Combined actuator analysis

This section discusses the advantages and disadvantages of each actuation method, as well as providing an example of how to use the combined actuators.

Table 2 summarizes the best way to use each actuator. The hardware actuators act at the PE-level, with a local impact, i.e., restricted to the PE where the actuation method is applied. On the other hand, the software actuators have a systemic action, i.e., at the application level. For example, consider an application that if admitted would violate the power cap. The decision heuristic may use TR to reduce the system power in such a way to accept the incoming application by joining tasks in the same processor.

To evaluate the energy efficiency brought by the coordinated use of hardware and software actuators, we created a test case consisting of a 6x6 many-core system running four applications: MPEG (A_1 - 5 tasks), Dijkstra (A_2 - 7 tasks), DTW (A_3 - 6 tasks), and AES (A_4 - 5 tasks). Fig. 15 shows the overall power consumption in different scenarios, and Fig. 16 shows the snapshots taken from the application mappings at crucial moments shown in Fig. 15.

First, we present our baseline scenario with no actuator in Fig. 15a. Each application starts its execution at a given moment, highlighted by

Table 2
Advantages and disadvantages of each actuation method.

Actuation Method	*Advantages*	*Disadvantages*	*Best way to use the actuator*
DVFS	Fine control at the PE level	Latency to switch the *vf*-pair; execution time overhead due to the synchronization process	During the task mapping select the *vf*-pair to respect the power cap; change the *vf*-pair dynamically to follow the power cap
Power Gating (PG)	Leakage reduction	Wake-up latency	Apply to processors not executing tasks and even to entire clusters
Clock Gating (CG)	Fine control at the task level	–	Take advantage of the idle times of processors when they have tasks that are waiting for data from other PEs
Application Admission (AA)	Power coarse grain actuator	New applications may be delayed to start	Considering that applications are pre-characterized in terms of power, the AA should be used to allow the execution of new applications *iff* they do not exceed the power cap
Task Mapping (TM)	Workload distribution	Complexity of the heuristics	Evenly distribution of the workload using power-aware heuristics to select processors with smaller load or temperature
Task Remapping (TR)	Mechanism to tweak the power consumption	Complexity of the heuristics and the communication synchronization	To reduce power: migrate tasks freeing processors; to increase power (respecting the power cap): split the tasks in free processors

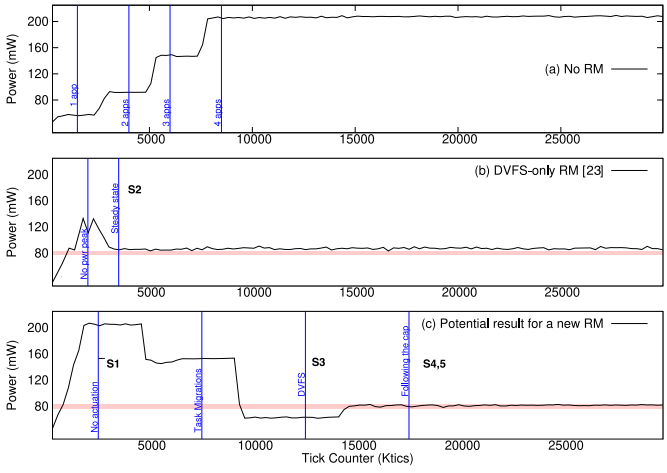

Fig. 15. Overall power consumption for a 6x6 many-core system considering memory, router, and processor of all PEs. The three scenarios for the test case are as follows: (a) No actuation - open loop; (b) A PID control use DVFS as power knob to follow the cap; and (c) An example of how the same test case can follow the cap by using a combination of actuators. Note that S1 to S5 correspond to the moment the snapshots were taken to evaluate the system usage.

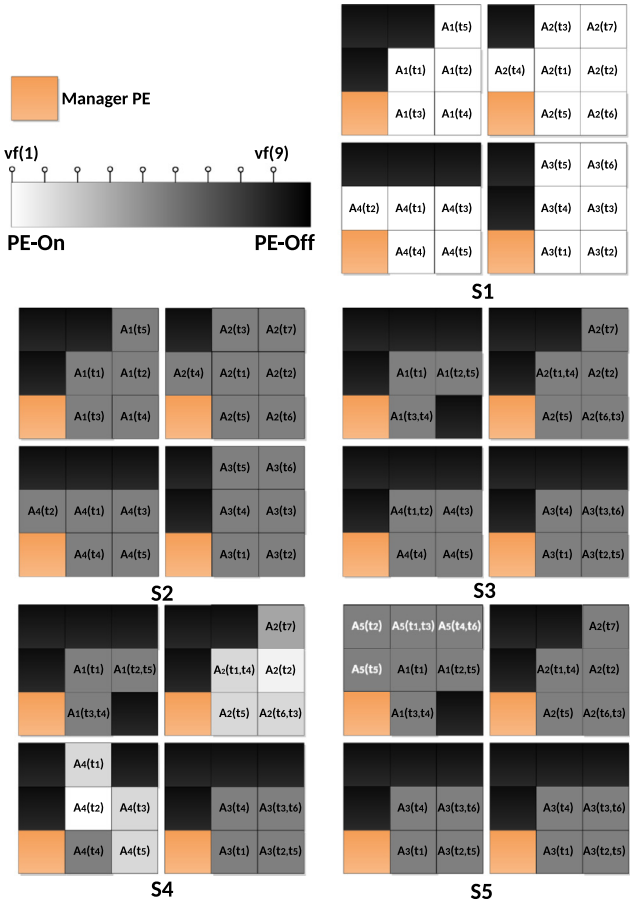

Fig. 16. Snapshots taken to show the application mappings at moments highlighted in Fig. 15.

the vertical lines. When the four applications are running in the many-core system, the power reaches 200mW, since in this scenario the system has no power management and all PEs are using the nominal *vf-pair* (vf1), as shown in Fig. 16 (Snapshot 1).

Our second scenario is based on Rahmani et al. [30] work. They propose a multi-objective control approach managed by a Proportional-Integral-Derivative (PID) controller. In Fig. 15b, we reproduce their approach using DVFS as the power knob - *DVFS-only*. In this experiment, all four applications start simultaneously. To keep power at moderate levels, the power cap is set to 80 mW, roughly 40% of the applications' power running without actuation. The first vertical mark corresponds to the actuation of the PID controller to avoid a power overshoot. The second vertical mark highlights the beginning of the steady state. Note that *DVFS-only* cannot reach 80 mW (the power always stays above the cap) even when setting minimum *vf-pair* for all SPs, as shown in Fig. 16 (Snapshot 2). Also, due to the adoption of only one actuator (i.e., DVFS), it is no possible to admit new applications into the system neither to improve the performance of a given application by using, e.g., task migration. Therefore, if on the one hand, it is possible to manage the system power

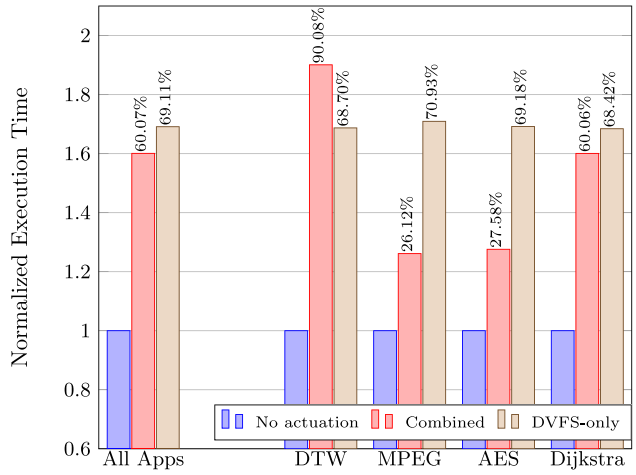

Fig. 17. Comparative performance results for different actuators.

using a single actuator. On the other hand, this approach does not have the flexibility to modify an application behavior or open room to admit new applications into the system.

To overcome the previous limitations, we propose to combine several actuators. Fig. 15c starts with no actuation, with the four applications starting simultaneously (as in Fig. 15b), and the power reaching 200 mW. Next, the actuation of task migration releases some SPs to become idle by joining communicating tasks in the same SP, reducing the system power consumption. The third mark represents the system power after setting all PEs executing tasks to the minimum *vf-pair*. Note that the system power is below the cap by using task migration with DVFS. This resulting scenario can be seen in Snapshot S3 in Fig. 16. Note that some PEs are executing simultaneously two tasks, there are more power-gated PEs than snapshot S2, and the power is below the cap. Due to this power slack, two actuation scenarios can be proposed, as shown in Fig. 16. It is possible to speed up some applications (snapshot S4, observe the clearer PEs), or it is possible to admit a fifth application (snapshot S5). Such flexibility on using the power slack is only possible using different actuators acting conjointly. In this example, we have chosen to relax the constraint and apply DVFS to an intermediate *vf-pair* (*vf(3)* at Fig. 5) for two applications, still respecting the power cap (forth mark) and improving the performance of these applications. This demonstrates that combining several actuators in a proper sequence allows the system to follow the power cap with flexibility.

In order to assess the performance overhead arising from the use of combined actuators, we performed a test case with the three scenarios presented in Fig. 15. To create a baseline, applications have been calibrated to finish at approximately the same time when running at peak performance, i.e., *vf(1)* with one task per PE. In addition, the applications reach the system in the following order: DTW, MPEG, AES and Dijkstra. Note that due to calibration, applications end in the same order of arrival.

Fig. 17 shows the overall performance results (left bars) and performance results per application (right bars). Results are normalized by the *no actuation* scenario. The other actuators are *DVFS-only* and *combined actuators*.

In regard to *DVFS-only*, as far as applications finish running, the next application has its *vf-pair* increased. In the end, all applications, as well as overall applications, finished their execution with similar performance losses (i.e., around 69%). Further, the *DVFS-only* runs above the power cap (Fig. 15b), which in addition to worsening performance, can not meet the imposed power cap.

On the other hand, *combined actuations* (i.e., DVFS + task migrations) commonly perform better than *DVFS-only* because it allows applications to speed up while respecting the power cap (similar to presented in Fig. 16 - S4). However, PE sharing leads to performance losses (only communicating tasks are allocated together at the same PE) and therefore applications with more parallel tasks are more likely to have relevant performance losses.

Concerning the *combined actuation* scenario, MPEG and AES are chosen to speed up by running at higher *vf-pairs* than the minimum, which causes them to finish their execution only 26% after the ideal situation. Next, Dijkstra is chosen to speed up its last iterations, so its performance loss is mitigated to 60%. Meanwhile, DTW runs at a minimum *vf-pair* and with a minimum number of PEs at all times and has the higher performance loss among all applications (i.e., 90%).

To conclude, regarding performance, overall results show the *combined actuation* scenario performs better than *DVFS-only*. However, applications may have bigger performance losses with *combined actuation* if they are not able to speed up at the right times throughout their execution.

7. Related work

This section discusses the related work, following two major themes. Namely "*Available Actuators*" and "*Actuators Improvements*". The Available Actuators theme addresses both the software and hardware actuators commonly found in many-core systems and how they are being used by different works. The Actuators Improvements theme describes how this work can be used to guide and improve the adoption of actuators. Regarding software actuators, most of the works found in the literature consider Application Admission as a problem prior to the TM, which makes AA the least employed. For example, authors [16,37–41] present frameworks that focus directly on deciding the best number of tasks for a given application before the TM. In Rahmani et al. [30], the application enters the system only if there are available processors, although applications can also be killed suddenly if the power overcomes the capping. In regard to TM, some works [16,37,38,42,43] assume one task per PE, considering the PE as one resource. It leads to an under-utilization of the system. On the other hand, other works admit to having two or more tasks per PE [5,22,39,44], like this work. The advantage of addressing multiple tasks in a single PE is to evaluate more realistic systems, acting at the early appearance of bottlenecks. To conclude the discussion on software actuators, TR enables runtime adaptability for task mapping into RM. Although some authors decided to turn the migration off due to the overhead costs and prefer to deal with the issues inherent to dynamic workloads using only DVFS [30], or change the application parallelism level [16,37] instead of employing TR. However, we have shown that TR is a powerful mechanism to tweak the power consumption in order to respect the power cap.

Once the tasks are mapped, hardware actuators become relevant, and DVFS has become the *de facto* mechanism to deploy power management in many-core systems. The granularity of DVFS varies between PE-level [5,16,23,29,36,43–45] and Cluster-level [22,46]. In this work, we suggest DVFS at the PE-level because it provides a fine-tuning of power management. When no tasks are running, some works [22,29,30] assume PG to avoid energy wasting and reduce the average power. However, we have seen that processors take a few microseconds to wake up, which is inappropriate for safety-critical applications. CG disables the clock switching off at parts of the circuit to save dynamic power. Since CG is a widely used and can employ this power actuator, few works mention CG explicitly [36]. In this context, we suggest that PG is the power actuator suitable for clusters with no tasks due to the latency to wake-up the circuit, while CG is ideal for small periods of idle times throughout the execution of tasks. Further, the manager can use DVFS at minimum levels plus CG when no tasks are running in case of absence of PG, as stated in [36,44,46].

Usually, authors abstract the discussion and implementation of hardware and software actuators because the *decision* (i.e., mapping heuristics) is the central focus of their work. However, some works consider previous design-time steps to capture actuators properties, such as la-

tency and power, to improve control and make more accurate decisions based on these properties [22,44,46]. Instead of focusing only on decision heuristics, our work details how actuators should be used, and their consequences. For example, a hierarchical adaptive multi-objective RM for many-core systems is presented by [47]. This proposed work can help them by disclosing the benefits and overheads of each actuator, making better heuristics to be employed in the future.

To the best of authors' knowledge, this is *the first* work that implements and provide insights that will help engineers develop appropriate resource management heuristics to improve self-adaptable many-core systems.

8. Final remarks

Future many-core systems will be closely related to effective resource management that addresses multi-objective and conflicting requirements such as power, performance, resilience, among others. In this work, we have extended a reference many-core platform to include several hardware and software actuators to support RM decisions and assessed the power impact brought by each of them. Further, a state-of-the-art comparison shows that no related work provides or details the same comprehensiveness of actuation methods concerning power consumption.

Regarding hardware actuators, we show how to implement DVFS, PG, and CG in a reference many-core system. Further, communication protocols illustrate how the hierarchical RM can coordinate software actuators to avoid resources conflicts and network congestion. In this regard, analyzing each knob individually allowed us to measure the power impact of each actuator.

Results show that hardware actuators have a more significant impact at PE-level because it provides a fine-tuning of power management. However, software actuators help in the adaptive decisions of RMs. Therefore, the use of both hardware and software actuators together brings the ability to better manage the power consumption of many-core systems.

Conflict of interest

None.

References

[1] H. Esmaeilzadeh, E. Blem, R.S. Amant, K. Sankaralingam, D. Burger, Dark silicon and the end of multicore scaling, IEEE Micro 32 (3) (2012) 122–134, doi:10.1109/MM.2012.17.

[2] N. Goulding, J. Sampson, G. Venkatesh, S. Garcia, J. Auricchio, J. Babb, M. B. Taylor, S. Swanson, GreenDroid: a mobile application processor for a future of dark silicon, in: HCS, 2010, pp. 1–39, doi:10.1109/HOTCHIPS.2010.7480072.

[3] A.M. Rahmani, P. Liljeberg, A. Hemani, A. Jantsch, H. Tenhunen, The Dark Side of Silicon: Energy Efficient Computing in the Dark Silicon Era, first ed., Springer, 2017.

[4] L. Ost, R. Garibotti, G. Sassatelli, G. Almeida, R. Busseuil, A. Butko, M. Robert, J. Becker, Novel techniques for smart adaptive multiprocessor socs, IEEE Trans. Comput. (2013) 1–14, doi:10.1109/TC.2013.57.

[5] H. Zhang, H. Hoffmann, Maximizing performance under a power cap: a comparison of hardware, software, and hybrid techniques, ACM SIGPLAN Not. 51 (4) (2016) 545–559, doi:10.1145/2954679.2872375.

[6] W. Quan, A.D. Pimentel, A hierarchical run-time adaptive resource allocation framework for large-scale MPSoC systems, Des. Autom. Embed. Syst. 20 (4) (2016) 311–339, doi:10.1007/s10617-016-9179-z.

[7] IBM, An Architectural Blueprint for Autonomic Computing, 2005,

[8] N. Dutt, A. Jantsch, S. Sarma, Toward smart embedded systems: a self-aware system-on-chip (SoC) perspective, ACM Trans. Embed. Comput. Syst. 15 (2) (2016) 22:1–22:27, doi:10.1145/2872936.

[9] N. Dutt, A. Jantsch, S. Sarma, Self-aware cyber-physical systems-on-chip, in: ICCAD, 2015, pp. 46–50, doi:10.1109/ICCAD.2015.7372548.

[10] R. Kavanagh, K. Djemame, Energy-aware self-adaptive middleware for heterogeneous parallel architectures, in: ISIICT, 2018, pp. 1–8, doi:10.1109/ISIICT.2018.8613291.

[11] K. Djemame, R. Kavanagh, V. Kelefouras, A. Aguilà, J. Ejarque, R.M. Badia, D.G. Pérez, C. Pezuela, J.-C. Deprez, L. Guedria, et al., Towards an energy-aware framework for application development and execution in heterogeneous parallel architectures, in: Hardware Accelerators in Data Centers, Springer, 2019, pp. 129–148, doi:10.1007/978-3-319-92792-3_7.

[12] A.K. Singh, C. Leech, B.K. Reddy, B.M. Al-Hashimi, G.V. Merrett, Learning-based run-time power and energy management of multi/many-core systems: current and future trends, J. Low Power Electron. 13 (3) (2017) 310–325, doi:10.1166/jolpe.2017.1492.

[13] H. Jiang, M. Marek-Sadowska, S.R. Nassif, Benefits and costs of power-gating technique, in: ICCD, 2005, pp. 559–566, doi:10.1109/ICCD.2005.34.

[14] Q. Wu, M. Pedram, X. Wu, Clock-gating and its application to low power design of sequential circuits, IEEE Trans. Circuits Syst. I 47 (3) (2000) 415–420, doi:10.1109/81.841927.

[15] L. Ost, M. Mandelli, G.M. Almeida, L. Moller, L.S. Indrusiak, G. Sassatelli, P. Benoit, M. Glesner, M. Robert, F. Moraes, Power-aware dynamic mapping heuristics for NoC-based MPSocs using a unified model-based approach, ACM Trans. Embed. Comput. Syst. 12 (3) (2013) 75:1–75:22, doi:10.1145/2442116.2442125.

[16] N. Kapadia, S. Pasricha, VARSHA: variation and reliability-aware application scheduling with adaptive parallelism in the dark-silicon era, in: DATE, 2015, pp. 1060–1065, doi:10.1145/1993498.1993502.

[17] A.K. Singh, M. Shafique, A. Kumar, J. Henkel, Mapping on multi/many-core systems: survey of current and emerging trends, in: DAC, 2013, pp. 1–10, doi:10.1145/2463209.2488734.

[18] A. Pathania, H. Khdr, M. Shafique, T. Mitra, J. Henkel, Scalable probabilistic power budgeting for many-cores, in: DATE, 2017, pp. 864–869, doi:10.23919/DATE.2017.7927108.

[19] G. PUCRS, Hermes Multiprocessor System on Chip, 2018,

[20] M. Ruaro, F.B. Lazzarotto, C.A. Marcon, F.G. Moraes, DMNI: a specialized network interface for NoC-based MPSoCs, in: ISCAS, 2016, pp. 1202–1205, doi:10.1109/ISCAS.2016.7527462.

[21] E. Carara, N. Calazans, F. Moraes, Router architecture for high-performance NoCs, in: SBCCI, 2007, pp. 111–116, doi:10.1145/1284480.1284515.

[22] T.S. Muthukaruppan, M. Pricopi, V. Venkataramani, T. Mitra, S. Vishin, Hierarchical power management for asymmetric multi-core in dark silicon era, in: DAC, 2013, pp. 1–9, doi:10.1145/2463209.2488949.

[23] P. Bogdan, R. Marculescu, S. Jain, Dynamic power management for multidomain system-on-chip platforms: an optimal control approach, ACM Trans. Des. Autom. Electron. Syst. 18 (4) (2013) 46:1–46:20, doi:10.1145/2504904.

[24] W. Kim, D. Brooks, G.-Y. Wei, A fully-integrated 3-level DC-DC converter for nanosecond-scale DVFS, IEEE J. Solid-State Circuits 47 (1) (2012) 206–219, doi:10.1109/JSSC.2011.2169309.

[25] W. Kim, M.S. Gupta, G.-Y. Wei, D. Brooks, System level analysis of fast, per-core DVFS using on-chip switching regulators, in: HPCA, 2008, pp. 123–134, doi:10.1109/HPCA.2008.4658633.

[26] Y. Choi, N. Chang, T. Kim, DC-DC converter-aware power management for low-power embedded systems, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 26 (8) (2007) 1367–1381, doi:10.1109/TCAD.2007.890837.

[27] M. Arora, S. Manne, I. Paul, N. Jayasena, D.M. Tullsen, Understanding idle behavior and power gating mechanisms in the context of modern benchmarks on CPU-GPU integrated systems, in: HPCA, 2015, pp. 366–377, doi:10.1109/HPCA.2015.7056047.

[28] R. Schöne, D. Molka, M. Werner, Wake-up latencies for processor idle states on current x86 processors, Comput. Sci. - R&D 30 (2) (2015) 219–227, doi:10.1007/s00450-014-0270-z.

[29] B. Raghunathan, Y. Turakhia, S. Garg, D. Marculescu, Cherry-picking: exploiting process variations in dark-silicon homogeneous chip multi-processors, in: DATE, 2013, pp. 39–44, doi:10.7873/DATE.2013.023.

[30] A.-M. Rahmani, M.-H. Haghbayan, A. Kanduri, A.Y. Weldezion, P. Liljeberg, J. Plosila, A. Jantsch, H. Tenhunen, Dynamic power management for manycore platforms in the dark silicon era: a multiobjective control approach, in: ISLPED, 2015, pp. 219–224, doi:10.1109/ISLPED.2015.7273517.

[31] J.-M. Chabloz, A. Hemani, Power management architecture in McNoC, in: Scalable Multi-core Architectures, Springer, 2012, pp. 55–80, doi:10.1007/978-1-4419-6778-7_3.

[32] G. Castilhos, M. Mandelli, G. Madalozzo, F. Moraes, Distributed resource management in NoC-based MPSoCs with dynamic cluster sizes, in: ISVLSI, 2013, pp. 153–158, doi:10.1109/ISVLSI.2013.6654651.

[33] M. Ruaro, F.G. Moraes, Demystifying the cost of task migration in distributed memory many-core systems, in: ISCAS, 2017, pp. 148–151, doi:10.1109/ISCAS.2017.8050257.

[34] S. Li, K. Chen, J.H. Ahn, J.B. Brockman, N.P. Jouppi, CACTI-P: architecture-level modeling for SRAM-based structures with advanced leakage reduction techniques, in: ICCAD, 2011, pp. 694–701, doi:10.1109/ICCAD.2011.6105405.

[35] S. Kumar, A. Jantsch, J.-P. Soininen, M. Forsell, M. Millberg, J. Öberg, K. Tiensyrjä, A. Hemani, A network on chip architecture and design methodology, in: ISVLSI, 2002, pp. 117–124, doi:10.1109/ISVLSI.2002.1016885.

[36] S. Maiti, N. Kapadia, S. Pasricha, Process variation aware dynamic power management in multicore systems with extended range voltage/frequency scaling, in: MWSCAS, 2015, pp. 1–4, doi:10.1109/MWSCAS.2015.7282119.

[37] D. Olsen, I. Anagnostopoulos, Performance-Aware resource management of multi-threaded applications on many-core systems, in: GLSVLSI, 2017, pp. 119–124, doi:10.1145/3060403.3060426.

[38] S. Kobbe, L. Bauer, D. Lohmann, W. Schrøder-Preikschat, J. Henkel, DistRM: distributed resource management for on-chip many-core systems, in: CODES+ISSS, 2011, pp. 119–128, doi:10.1145/2039370.2039392.

[39] V. Tsoutsouras, I. Anagnostopoulos, D. Masouros, D. Soudris, A hierarchical distributed runtime resource management scheme for NoC-Based many-Cores, ACM Trans. Embed. Comput. Syst. (TECS) 17 (3) (2018) 65:1–65:26, doi:10.1145/3182173.

[40] V. Tsoutsouras, S. Xydis, D. Soudris, Application-arrival rate aware distributed run-Time resource management for many-Core computing platforms, IEEE Trans. Multi-Scale Comput. Syst. 4 (3) (2018) 285–298, doi:10.1109/TMSCS.2018.2793189.
[41] A. Pathania, V. Venkataramani, M. Shafique, T. Mitra, J. Henkel, Defragmentation of tasks in many-core architecture, ACM Trans. Architect. Code Optim. 14 (1) (2017) 2:1–2:21, doi:10.1145/3050437.
[42] R. Garibotti, A. Butko, L. Ost, A. Gamatié, G. Sassatelli, C. Adeniyi-Jones, Efficient embedded software migration towards clusterized distributed-Memory architectures, IEEE Trans. Comput. 65 (8) (2016) 2645–2651, doi:10.1109/TC.2015.2485202.
[43] A.M. Rahmani, A. Jantsch, N. Dutt, HDGM: Hierarchical dynamic goal management for many-Core resource allocation, Embed. Syst. Lett. 10 (3) (2018) 61–64, doi:10.1109/LES.2017.2751522.
[44] V. Hanumaiah, S. Vrudhula, Energy-efficient operation of multicore processors by DVFS, task migration, and active cooling, IEEE Trans. Comput. 63 (2) (2014) 349–360, doi:10.1109/TC.2012.213.
[45] H. Yu, R. Syed, Y. Ha, Thermal-aware frequency scaling for adaptive workloads on heterogeneous MPSoCs, in: DATE, 2014, pp. 1–6, doi:10.7873/DATE.2014.304.
[46] Z. Lai, K.T. Lam, C.-L. Wang, J. Su, Y. Yan, W. Zhu, Latency-aware dynamic voltage and frequency scaling on many-core architectures for data-intensive applications, in: CloudCom-Asia, 2013, pp. 78–83, doi:10.1109/CLOUDCOM-ASIA.2013.68.
[47] A.L.d.M. Martins, A.H.L. da Silva, A.M. Rahmani, N. Dutt, F.G. Moraes, Hierarchical adaptive multi-objective resource management for many-core systems, J. Syst. Architect. (2019) 1–12, doi:10.1016/j.sysarc.2019.01.006.

André L. M. Martins received the M.Sc. degree in 2011 from the Federal University of Rio Grande do Sul (UFRGS), and the Ph.D in 2017 from the Pontifical Catholic University of Rio Grande do Sul (PUCRS). He is Associate Professor at Sul-riograndense Federal Institute of Education (IFSul). His main research interest includes multiprocessor systems on chip (MPSoCs) and networks on chip (NoCs).

Rafael Fraga Garibotti. Is currently an Associate Professor at PUCRS University. Former Postdoctoral Fellow at Harvard University. He received his Ph.D. and MSc. Degree in Microelectronics, respectively from University of Montpellier II and EMSE, France and his BSc. Degree in Computer Engineering from PUCRS, Brazil. His research activity focuses on programmability of multi/many-core systems and design of parallel architectures, such as accelerators. He is a member of the IEEE.

Nikil Dutt received a Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign in 1989, and is currently a Distinguished Professor at the University of California, Irvine, with academic appointments in the CS, EECS, and Cognitive Sciences departments. His research interests are in embedded systems, EDA, computer systems architecture and software, and brain-inspired architectures and computing. He received numerous best paper awards at conferences. Dutt previously served as Editor-in-Chief of ACM TODAES and as Associate Editor for ACM TECS and IEEE TVLSI. He has served on several premier EDA and Embedded System Design conferences and workshops, and serves or has served on the advisory boards of ACM SIGBED, ACM SIGDA, ACM TECS and IEEE Embedded Systems Letters (ESL). He is a Fellow of the ACM, Fellow of the IEEE, and recipient of the IFIP Silver Core Award.

Fernando G. Moraes (SM2) received the Electrical Engineering and M.Sc. degrees from the Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, Brazil, in 1987 and 1990, respectively. In 1994 he received the Ph.D. degree from the Laboratoire dnformatique, Robotique et Microélectronique de Montpellier), France. He is currently Professor at PUCRS. He has authored and co-authored 29 peer refereed journal articles in the field of VLSI design. His primary research interests include Microelectronics, FPGAs, reconfigurable architectures, NoCs and MPSoCs.