Journal of Systems Architecture 97 (2019) 416–427

Contents lists available at ScienceDirect

Journal of Systems Architecture

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

Hierarchical adaptive Multi-objective resource management for many-core systems

André Luís del Mestre Martins[a], Alzemiro Henrique Lucas da Silva[b], Amir M. Rahmani[c], 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, UCI, Irvine, USA

ARTICLE INFO

Keywords:
Self-Awareness
Adaptability
Multi-Objective
Many-Core
Resource management

ABSTRACT

The typical workload of many-core systems produces peaks and valleys of resources utilization throughout the time. The power capping limits the full system utilization in a workload peak, but also creates a power slack to apply different resource management (RM) policy in a valley phase. Related works do not consider this workload behavior, by proposing RMs with fixed goals. This work proposes a hierarchical adaptive Multi-Objective Resource Management (MORM) for many-core systems under a power cap. MORM works with dynamic workloads, which presents peaks and valleys of utilization. The hierarchical approach allows clusters of processing elements (PEs) to execute applications according to different objectives simultaneously. A cluster can drive the PEs to optimize either performance or energy. MORM can dynamically shift the goals of a cluster according to the workload behavior. Comparison with a *state-of-the-art* RM optimized for single objective shows that MORM achieves equivalent results in a workload valley while outperforming up to 37.19–49.03% the performance in a workload peak regardless of the power cap. The comparison reveals relevant features to be considered in large many-core systems: hierarchical organization, multi-task mapping, and joint adaptability between software (remapping) and hardware (DVFS) actuation.

1. Introduction

Many-core systems provide high-performance computing for distinct market segments, such as embedded systems, desktop PCs, and servers. Due to the high power dissipation inherent in recent technology nodes, power capping constrains the full utilization of many-core systems [1–3]. A resource management (RM) unit dynamically allocates tasks from arriving applications for execution on PEs. To meet the system goals, RM employs distributed or hierarchical approaches to ensure scalability in large systems [4].

The challenge of executing dynamic workloads on many-core systems under specific constraints motivated researchers to employ RMs to meet one or more specific objectives. At the application level, the RM evaluates if the system has enough power and resources for new incoming applications [5,6]. Once the RM allows the application to execute, task mapping algorithms find the most suitable area to place the application' tasks. At runtime, task remapping [7] and Dynamic Voltage and Frequency Scaling (DVFS) may modify the system to adjust energy and performance of the applications according to the system status [8–10]. Thus, comprehensive management for many-core systems includes application admission, task mapping and remapping, and DVFS under a power cap using a hierarchical scheme.

Fig. 1 shows a typical workload behavior of a computational system, with peaks and valleys of system utilization [11] and, consequently, power. The figure highlights the number of executing tasks and allocated processors, in a peak and a valley of system utilization. The horizontal line corresponds to the power cap. As shown, in the utilization peak, the dissipated power is higher than the cap, justifying the adoption of an RM. For example, mobile systems reproduce similar utilization behavior by presenting low workload when in standby mode and high workload when the user is handling it [12]. Even in active mode, the system utilization is frequently varying according to the number of executing applications [13]. Servers for cloud computing are another example of a variation in the workload demand, but the time scale of the variation can be hours or days [14].

* Corresponding author.
E-mail addresses: almmartins@charqueadas.ifsul.edu.br (A.L.d.M. Martins), alzemiro.silva@acad.pucrs.br (A.H.L. da Silva), amirr1@uci.edu (A.M. Rahmani), 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.2019.01.006
Received 28 September 2018; Received in revised form 16 January 2019; Accepted 22 January 2019
Available online 23 January 2019

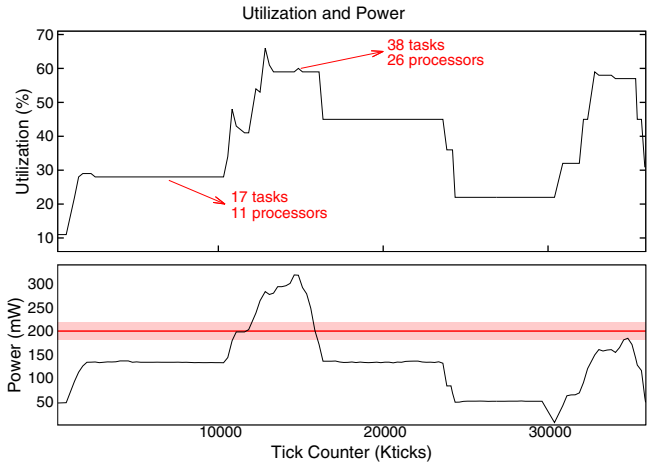


Fig. 1. System utilization and power for a 6 × 6 many-core system running a dynamic workload.

Due to the power capping, the system may not admit power peaks on a high utilization phase. For instance, the RM decides which applications should speed up and down, creating a resource sprinting situation. Meanwhile, the low workload may be an opportunity for boosting applications and perhaps finish some of them faster before the next peak when there is a power slack available. As another strategy, the RM can also activate a low power mode for the low workload to save energy. Therefore, an efficient RM requires adaptive mechanisms to make the system follow different goals throughout the time and benefit from the current workload status.

On the other hand, it has been shown in the literature that due to several constraints related to technology, environment, and workload variation (e.g., thermal issues, security, performance, and energy efficiency), the current many-core systems demand multi-objective RMs [15]. These objectives can be partially/fully overlapping, contradicting, and conflicting with each other. In case of conflicting objectives, the RM needs to opportunistically select a subset of goals to track over time and dynamically switch between the goals according to certain conditions to satisfy multiple dynamic objectives. Thus, fixed-objective RM covers only specific cases chosen according to narrow contexts.

Accordingly, the development of an RM able to select different objectives at runtime is challenging due to the following reasons: (*i*) the adaptability required for shifting the goal of the system (or a portion of it) at runtime according to the dynamic workload, (*ii*) the wide actuators set that the RM needs to manage and coordinate to support runtime changes of system goals, and (*iii*) the scalability issue inherent to the many-core systems.

This paper presents an RM called Multi-Objective Resource Management – MORM. MORM targets large many-core systems running dynamic workloads under restricted power capping. MORM controls at runtime multiple actuators in a coordinated way to provide the required adaptability to the system and enable multi-objective capability. Multi-objective in this context indicates that MORM addresses power, energy, and performance concomitantly, considering communication, computation and scalability issues. MORM is also adaptive as it can switch goals at runtime and optimize either performance or energy. This work is aligned with important trends for resource allocation [4]: (*i*) multi-objective resource allocation, (*ii*) consideration of communication and computation loads together, and (*iii*) addressing scalability in large-scale architectures.

The main *contributions* of this work are as follows:

- An approach that can dynamically prioritize different objectives by changing the system settings according to the workload variation throughout the time;
- A holistic approach including techniques for application admission, task mapping, task remapping, and DVFS, to make a trade-off between conflicting objectives: performance or energy in a coordinated way;
- A hierarchical organization which distributes the workload in clusters for allowing applications to run according to different goals in different regions simultaneously.

This paper is organized as follows. Section 2 reviews related works. Section 3 presents the many-core architectural features and the application model. Sections 4 overviews MORM. Sections 5 and 6 detail the MORM decisions at the system and cluster levels, respectively. Section 7 presents the experimental results, and Section 8 concludes this paper.

2. Related works

Table 1 summarizes related works concerning the features required by RMs: (*i*) management of the applications admission (AA); (*ii*) task mapping/remapping (TM and TR); (*iii*) DVFS control; (*iv*) power cap; (*v*) hierarchical management; and (*vi*) adaptive goals, i.e., the RM can manage the applications to meet distinct goals dynamically.

Regarding application admission (second column), works [6,18] present frameworks for deciding the best number of tasks by adapting the application parallelism to the available resources on the system or the power capping. In the work by Rahmani et al. [16], the application enters the system if there are available processors, but can also be killed suddenly if the power overcomes the capping. An alternative approach for AA assumes that application tasks can have approximated versions to trade-off power and performance [20]. MORM can remap running tasks to share PEs and map incoming applications in a reduced number of PEs to open power and resources room.

Applications need to be mapped once they enter into the system. Mapping heuristics have inherent challenges such as disturbances in other applications, traffic, and scalability [7]. In general, for homogeneous many-core systems, runtime mapping heuristics assume one task per PE and assign tasks in a continuous shape to avoid network congestion and optimize performance [22]. On the other hand, the assignment of more than one task per PE minimizes the hop number and the number of active PEs and leads to energy savings [23]. Some proposals [6,16,20,21] deploy distinct algorithms to map one task per processor in square shapes (third column). Another mapping approach to meet power constraints is the pattern mapping [24]. MORM takes advantage of the hierarchical system organization to propose two lightweight mapping heuristics that enable a fast adaptation between energy and performance goals.

Task remapping employs task migration to deal with the availability of resources dynamically. When a task arrives to execute, remapping the running tasks before mapping the incoming task delivers a better result than mapping straight the incoming task on the available resources. The overhead of a task migration is low for homogeneous many-cores without shared memory [25]. As some works [6,16,20] do not support multi-tasking mapping, task remapping brings no significant advantages concerning power and resources, and then it is not employed (fourth column). Pathania et al. [21] benefit from an exponential property of contiguous square shapes to optimally remap running tasks and avoid resources fragmentation due to dynamic workload. MORM uses two task remapping approaches, *join* and *split*, to perform adaptability for optimizing the system or part of the system to a new goal according to the workload. The *join* remapping stimulates the PE sharing to save energy by reducing the communication between tasks and creating more idle PEs for power gating. The *split* remapping spreads the tasks in more PEs to optimize performance.

As soon as tasks are running, DVFS is the power actuator to trade-off energy and performance at the task level. Besides that, some works

Table 1
Features of comprehensive RM for many-core systems.

Proposal	AA	TM	TR	DVFS	Pwr cap	Scalability	Adaptive Goal
Kapadia et al. [6]	√	√		√	√		
Rahmani et al. [16]		√		√	√		
Zhang et al. [17]		√	√	√	√		
Olsen et al. [18]	√	√			√		
Pathania et al. [8]		√		√	√	√	
Kim et al. [19]				√	√		
Kanduri et al. [20]	√	√		√	√		√
Pathania et al. [21]		√	√		√		
This work	√	√	√	√	√	√	√

AA: Application Admission; TM: Task Mapping; TR: Task Remapping

[6,16,20] employ DVFS (fifth column) at the application level to optimize the power with PID controllers. Alternative approaches for power capping propose DVFS assignment through reinforcement learning [19] and probabilistic [8] techniques. However, a recent work [17] shows that a joint actuation between DVFS and resources allocation boost the performance under a power cap. The DVFS approach employed in this proposal works with task mapping/remapping heuristics to maximize the adaptive goal as well as can also identify opportunities to save energy according to the task phase.

Regarding scalability (sixth column), distributed and hierarchical RM can guarantee scalability for current many-core systems [26]. In such complex systems, a centralized RM compromises the system performance by inducing network congestion and hotspots. The probabilistic RM of Pathania et al. [8] require no management messages to observe the system and avoid most of the overheads from management to provide scalability up to thousands of PEs. This work employs a hierarchical organization by grouping PEs in clusters to guarantee scalability.

Finally, the main feature and contribution of MORM is the adaptability according to the workload. The RM proposed in Kanduri et al. [20] can dynamically choose between an accurate version or an approximated version of the same task to make the application follow different goals. The remaining works do not assume dynamic changes in the workload, i.e., the RM strategy follows the same goal regardless the system utilization. While the workload is low, the power slack allows the RM to decide an optimization for saving energy or a boosting on performance. At peaks of workload, the cluster hierarchy allows the RM to set some clusters to boost some applications while still keeping the power capping by slowing others clusters down.

Overall, Table 1 highlights the comprehensiveness of related RMs. Besides, the discussion of the motivational example (Fig. 1) evidences that RMs for many-cores definitely require multi-objective purpose functions. Moreover, power-cap and scalability are primaries concerns and include additional complexity in the RM. Therefore, the *key issue* is the coordination of the wide actuation set (AA, TM, TR, and DVFS) to adapt the system for meeting a new goal when an event (e.g. the system utilization due to the workload) leverage to a goal shifting, while keeping scalability and power capping.

3. Background

Fig. 2 presents the main architectural features of the many-core system to support the techniques herein proposed. The system adopts a hierarchical organization, with virtual regions, named clusters. Each PE may assume distinct roles, defined by software:

- *Slave PE – SP*: execute applications' tasks using a multi-task scheduler (round-robin scheduler) and send observed data to a manager PE.
- *Cluster Manager PE – CM*: manage the SPs of a given cluster, executing functions such as mapping, remapping, DVFS control, and sends the cluster information to the global manager. CMs only execute management functions.

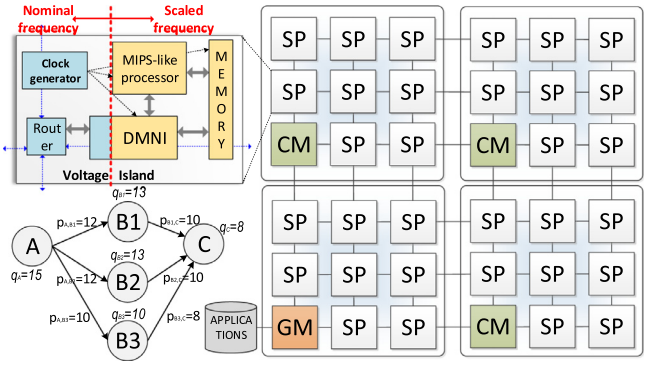


Fig. 2. NoC-based homogeneous many-core system.

- *Global Manager PE – GM*: Execute all functions of the CMs and also selects the cluster to execute incoming applications from the external world (application repository).

Fig. 2 has four 3 × 3 clusters. Each cluster has one manager PE (*CM* or *GM*) and a set of *SP*s. The reason to adopt this hierarchical organization is to ensure scalability, by distributing the management actions at different manager PEs. The cluster size is defined at design time. At execution time, when the cluster has all its resources in use, its manager PE may borrow resources from neighbour clusters. According to Castilhos et al. [27], a cluster size with 16 (4 × 4) PEs represents a good trade-off between execution time optimization and resources reserved for management. Moazzemi et al. [28] survey RMs for many-core systems and conclude that the Supervisory Control Theory (an approach similar to ours) is the most suitable approach regarding scalability.

All PEs have the same hardware, with a router, a clock generator, a private memory, a processor and a network interface (DMNI). Relevant features of the PEs related to this work include:

- *DVFS at PE level*: the router always works at the nominal frequency to avoid network stalls, while other modules may work at different frequencies according to the workload.
- *Observing*: SPs have *virtual sensors*, able to count the number of packets traversing the router, the number of memory accesses, and the number of executed instructions periodically, according to a sampling window. The data from these counters enables the CM to estimate the consumed energy [29].

Applications are modeled as directed acyclic task graphs, $A = (T, E)$, where:

- the vertex $t_i \in T$ is a task, with a weight q_i corresponding to its power consumption when t_i executes in a PE with no resources sharing;
- the directed edge $e_{ij} \in E$, with a weight p_{ij}, corresponds to the power consumption when t_i and t_j share the same PE.

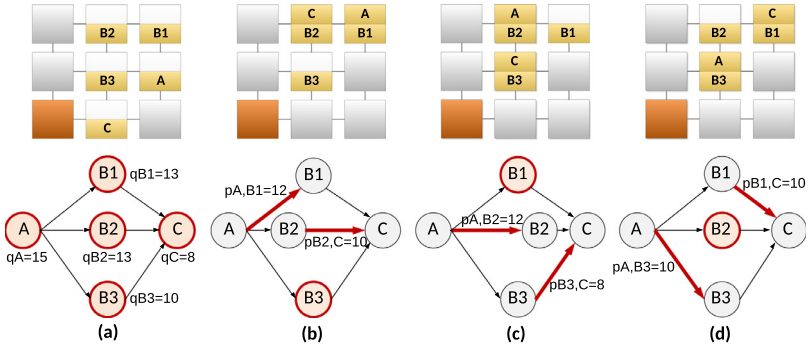


Fig. 3. Steps to obtain q_i and p_{ij} weights for a given application.

Obtaining q_i and p_{ij} weights require a design-time evaluation of the application set. Applications are simulated individually in the many-core to avoid disturbances from other sources. The number of PEs is set according to the number of the tasks. All SPs are set to the nominal *vf-pair* (voltage-frequency pair) to get q_i, and to the most energy efficient *vf-pair* to obtain p_{ij}.

Fig. 3 shows the steps to generate q_i and p_{ij} weights for a given application. To obtain q_i, the tasks are manually mapped so that an SP executes one task in a contiguous shape. At the end of the simulation, q_i corresponds to the worst-case value among all power sampling reported by the SP where t_i was allocated.

Next, to derive a p_{ij}, the communicating tasks t_i and t_j share the same SP while the remaining tasks run individually in another SPs. Similarly to q_i, p_{ij} is the worst-case value among all power samplings reported by the SP where t_i and t_j are allocated. Differentely from other works that assign to the edges the communication power [30,31], p_{ij} corresponds to the power measure when the communicating task t_i and t_j share the same PE. Note that, generating all p_{ij} values require multiple simulations to test all e_{ij} values assigned exclusively to an SP. For instance, the application from Fig. 3 has six edges and needs three simulations to extract all p_{ij} values. For this example, the first task mapping could allocate t_A and t_{B1}, and t_{B2} and t_C to share SPs for deriving p_{A-B1} and p_{B2-C}. Second simulation reports p_{A-B2} and p_{B3-C} and the last one generates p_{A-B3} and p_{B1-C}.

Once q_i and p_{ij} weights are computed, the manager PEs can derive the application power from any mapping combination. In Fig. 3(a) the application power is the sum of all q_i weights ($A_{pwr} = \sum q_i$) while in Fig. 3(b) the application power corresponds to $A_{pwr} = q_{B3} * k + p_{A,B1} + p_{B2,C}$ where k is a constant to obtain q_i in *energy mode* (Section 5).

4. Multi-objective resource management – MORM

Fig. 4 overviews MORM concerning the hierarchical organization, which adopts the observe-decide-act paradigm [32] to manage the system. Section 5 details the system level decisions, related to the admission of new applications. Next, Section 6 presents the adaptability at the cluster level.

Observing data follows a bottom-up direction. SPs send data (e.g., energy, CPU utilization, NoC congestion) to their CMs. Each CM transmits to the GM the current power consumption of its cluster. Manager PEs (CMs and GM) take *decisions* at cluster and system levels respectively. At the cluster level, a given CM may decide to modify, for example, the voltage-frequency pair of a set of SPs. At the system level, the GM may change the operation mode of a given cluster. Thus, *actuation* follows a top-down direction, with actions send from the GM to CMs and from CMs to SPs.

Definition 1. *Operation modes* – clusters may operate in one of two operation modes: (*i*) *performance mode* - CM optimizes the resources to minimize the execution time of the running applications; (*ii*) *energy mode* - CM optimizes the resources to improve the energy efficiency.

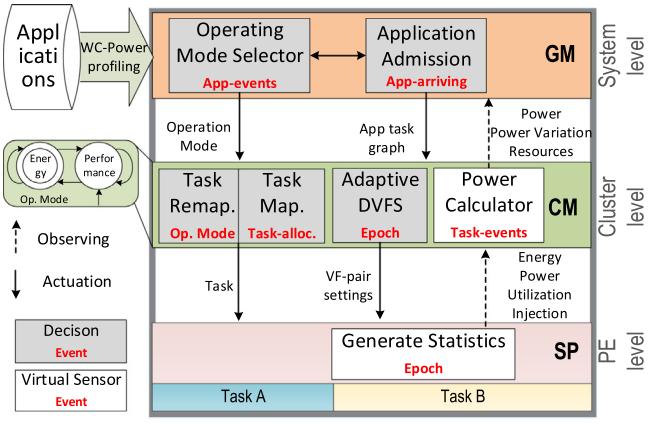


Fig. 4. Overview of the multi-objective resource management for many-core systems.

In Fig. 4, gray and white boxes correspond to *decision* algorithms and *virtual sensors*, respectively. Labels inside each box correspond to the event responsible for firing both decisions algorithms and observing from sensors. Events are classified into four classes:

Definition 2. *Application events* - correspond to external notifications to the GM that an application is ready for admission or a CM reports the end of an application to the GM.

Definition 3. *Task events* - correspond to the moment that a CM maps or remaps a task to an SP or the moment an SP reports to a CM that a task finished its execution.

Definition 4. *Operation Mode events* - correspond to the moment that GM changes the operation mode of a given cluster.

Definition 5. *Epoch events* - correspond to a periodical hardware interruption, where SPs report the observing data to its CM.

The system level management is in charge to take decisions at the application level, to maintain the power cap, and to choose the operation modes (Section 5). The cluster level management controls the DVFS, and task mapping and remapping (Section 6).

5. MORM System level decisions

MORM allows a new arriving application to execute if the application does not exceed the power cap, and the system has available resources. First, the GM verifies the additional power required by the incoming application (Section 5.1). After, the GM verifies the resources availability (Section 5.2). If none of the conditions are satisfied, MORM can modify the operation mode of the clusters to find room for the incoming application.

Algorithm 1 MORM Operating Mode Selector.

1: **Inputs**: app, $energyCl_{set}$, $perfCl_{set}$
2: **if** app is arriving **then**
3: $newPwr \leftarrow sys.pwr + app.pwrPerformance$
4: **if** $newPwr < sys.pwrCap$ **then**
5: *Allows the admission of the app. in performance mode*
6: **else**
7: $newPwr \leftarrow sys.pwr + app.pwrEnergy$
8: **if** $newPwr > sys.pwrCap$ **then**
9: **for each** $cl_i \in perfCl_{set}$ **do**
10: $newPwr \leftarrow newPwr + cl_i.pwrVariation$
11: **if** $newPwr > sys.pwrCap$ **then**
12: **shiftOpMode**(cl_i, *energy)*
13: **end if**
14: **end for**
15: **if** $newPwr < sys.pwrCap$ **then**
16: *Allows the admission of the app. in energy mode*
17: **else**
18: *Application enqueued to be admitted later*
19: **end if**
20: **else**
21: **if** $energyCl_{set} = \emptyset$ **then**
22: $cl_{output} \leftarrow$ **maxAvailSPs**($perfCl_{set}$, $performance$)
23: **shiftOpMode**(cl_{output}, $energy$)
24: **end if**
25: *Allows the admission of the app. in energy mode*
26: **end if**
27: **end if**
28: **else** ▷ app finished its execution
29: **for each** $cl_i \in energyCl_{set}$ **do**
30: $newPwr \leftarrow sys.pwr + cl_i.pwrVariation$
31: **if** $newPwr < sys.pwrCap$ **then**
32: **shiftOpMode**(cl_i, $performance$)
33: **end if**
34: **end for**
35: **end if**

Algorithm 2 MORM Application Admission.

1: **Inputs**: app, app_{mode}, $sys.cl_{set}$
2: **Outputs**: cl_{output}
3: $cl_{set} \leftarrow \emptyset$
4: $SP_{\min} \leftarrow$ **getMinSPsAdmitApp**(app, app_{mode})
5: **for each** $cl_i \in sys.cl_{set}$ **do**
6: **if** $app_{mode} = cl_i.mode$ **and** $cl_i.freeSP \geq SP_{\min}$ **then**
7: $cl_{set} \leftarrow cl_{set} \cup cl_i$
8: **end if**
9: **end for**
10: **if** $cl_{set} \neq \emptyset$ **then**
11: $cl_{output} \leftarrow$ **maxAvailSPs**(cl_{set}, app_{mode})
12: **return** cl_{output}
13: **end if**
14: $cl_{output} \leftarrow$ **maxAvailSPs**($sys.cl_{set}$, $performance$)
15: $app_{mode} \leftarrow energy$
16: $SP_{\min} \leftarrow$ **getMinSPsAdmitApp**(app, app_{mode}) ▷ update $SP_{\min}$
17: **if** $cl_{output}.freeSP \geq SP_{\min}$ **then**
18: **shiftOpMode**(cl_{output}, $energy$)
19: **return** cl_{output}
20: **else**
21: **return** $\emptyset$
22: **end if**

Algorithms 1 and 2 have a linear complexity ($\mathcal{O}(n)$) and do not impact directly in the applications' performance since they are executed in manager processors. Such algorithms may impact in the applications' execution time when a given application changes its operation mode from performance to energy to respect the power cap when admitting a new application.

5.1. Operating mode selector

The GM decides the operation mode of each cluster based on different power values from the system, cluster and application. The power values used in the Operating Mode Selector are the following:

Definition 6. *Application Power Performance* - prediction of the application power by using the *performance mode*. This prediction is obtained from $\sum_{i=1}^{n} q_i$, where n is the application tasks' number.

Definition 7. *Application Power Energy* - prediction of the application power by using the *energy mode*. This prediction considers weights p_{ij} and q_i as a function of the mapping, and a k ratio to adjust q_i to the *vf-pair* of energy mode (Fig. 3).

Definition 8. *Cluster Power* - sum of all monitored power samples in the SPs belonging to the CM.

Definition 9. *Cluster Power Variation* - effect on the *Cluster Power* when the cluster operation mode changes (Definition 1). Section 6.4 details the cluster power variation computation.

Definition 10. *System Power* - the sum of all *Cluster Power* values.

Definition 11. *System Power Cap* - the upper bound value of power.

MORM employs proactive actuations to respect the power cap based on the estimation of power disturbances due to the application events (Definition 2) and operation mode events (Definition 4).

Application events modify the total system power and the *Operating Mode Selector* takes decisions by evaluating the expected power impact due to these events. If an application finishes its execution, the CM resets the counters of the SPs where the application was mapped and then updates the GM with new power values. When an application requests admission, the GM takes decisions based on the application power estimation (Definition 6).

Operation mode events disturb the *cluster power* (Definition 8) due to task remappings and voltage-frequency changes that the CM executes when receiving a new operation mode (Section 6). The GM is aware of the power disturbance from operation mode events by observing *cluster power variation* (Definition 9) of each cluster.

Algorithm 1is the proactive power control knob for shifting operation modes of the clusters while respecting the system power cap (Definition 11) based on the amount of power disturbance induced by application events (Definition 2). At the beginning of the system execution, all clusters operate in *performance mode*. The algorithm may update the *operation mode* of the clusters when an application requests admission into the system (lines 2–29), or it finishes the execution (lines 28–34).

The algorithm receives as inputs the application description (*app*), the set of clusters operating in *energy mode* ($energyCl_{set}$), and the set of clusters operating in *performance mode* ($perfCl_{set}$).

If *app* is requesting its admission (line 2), MORM estimates the power increases if the application is mapped in *performance mode* (line 3). If the estimation does not exceed the system power cap (Definition 11), *app* may be admitted in *performance mode* (line 5). Otherwise, MORM estimates the increasing of power to admit the application in *energy mode* (line 7):

– If the estimated power is above the cap, the loop between lines 9–14 evaluates if *app* may be admitted by changing a given cluster in $perfCl_{set}$ to *energy mode*. Line 10 makes this estimation by using

the $cl_i.pwrVariation$ (Definition 9), and if it is possible to admit *app*, cl_i shifts from performance to *energy mode* (line 12). If the power is below the capping, the *app* may be admitted in *energy mode* (line 16). Otherwise, *app* is enqueued to be executed later (line 18).

– If the estimated power is below the capping it is possible to admit *app* the in *energy mode* - lines 21–26. For that, one cluster must be in *energy mode* to receive *app*. If there is no cluster in *energy mode* (line 21), the function *maxAvailSPs* finds the cluster running in *performance mode* with the maximum number of available processors, cl_{output}. The algorithm then shifts cl_{output} to *energy mode* (line 23).

When a given application finishes its execution (lines 29–34), the algorithm verifies if it is possible to shift a cluster from *energy mode* to *performance mode*, without violating the power cap. The reasoning to change the operation mode is to benefit from the power slack and boost some applications running in clusters in *performance mode*. When applications run in *performance mode*, they are more likely to finish earlier and open room for the execution of new applications.

5.2. Application admission

After verifying the application admissibility regarding power, *Application Admission* (Algorithm 2) verifies the application admissibility regarding available resources and selects the cluster to map the application. The algorithm receives as inputs the application description (*app*), the app_{mode} defined in Algorithm 1, and the set of clusters ($sys.cl_{set}$).

The algorithm starts by creating an empty set, cl_{set}, which will contain the clusters candidate to receive *app* (line 3). Next, it computes the number of SPs required for executing an application according to the application mode, function *getMinSPsAdmitApp* (line 4). In *performance mode*, the number of SPs is equal to the number of the application tasks, and in *energy mode*, this value is smaller due to the CPU sharing among communicating tasks.

The loop between lines 5–9 fills cl_{set} with the clusters' identifiers that may receive *app*. At the end of the loop, if cl_{set} is not empty, the selected cluster is the one with the maximum SPs running no tasks (lines 10–13). The function *maxAvailSPs* returns a cluster identifier with an operation mode equal to the *app* mode (second parameter of the function).

If there is no SPs available in any cluster, i.e., cl_{set} is empty, Algorithm 2 selects a cluster in *performance mode* as the candidate to receive the new application (line 14). In this case, the application mode changes to *energy mode* (line 15) and the number of required SPs is updated (line 16). If cl_{output} can receive *app* in *energy mode, Application Admission* changes its operation mode to *energy mode* (line 18), and cl_{output} receives *app* (line 19). Otherwise, the function returns null, and the application waits in a queue for later admission.

6. MORM Cluster level decisions

The adaptability at the cluster level enables clusters to work at different operation modes simultaneously. This section describes the three mechanisms adopted at the cluster level: adaptive DVFS, task mapping and task remapping.

6.1. Adaptive DVFS

Adaptive DVFS is an adaptive threshold-based algorithm that follows the cluster operation mode [33]. From an available set of voltage-frequency pairs (*vf-pairs*), the algorithm adopts three modes:

– Performance – VF_{perf}: nominal *vf-pair*, i.e., the highest voltage and frequency values;
– Low power – $VF_{\min}$: lowest voltage and frequency values;
– EDP – VF_{EDP}: most energy efficient *vf-pair* between VF_{perf} and $VF_{\min}$.

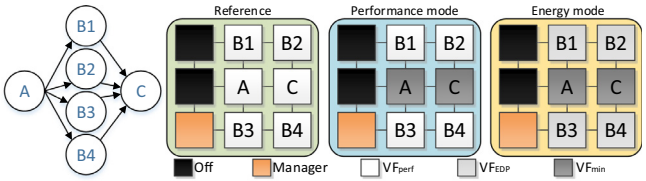


Fig. 5. MORM *Adaptative DVFS* selects the VF settings of a 3 × 3 cluster in both *operation modes* for a given application. Reference: no DVFS.

The *Adaptive DVFS* applies $VF_{\min}$ in two cases related to communication issues, regardless the operation mode:

1. HI (high injection): an SP is injecting messages on the network in a higher speed than the messages are consumed;
2. LU (low processor utilization): an SP is in idle state most of time waiting for messages.

MORM adopts a similar method to [16] to evaluate the message injection rate. The injection rate is the average utilization of the input buffer in the local port. The HI threshold is activated when the injection is higher than, for example, 75%. The LU threshold is activated when the utilization is below than, for instance, 25%.

Fig. 5 illustrates an example of how *Adaptive DVFS* associates VF settings according to the operation mode by using DVFS to save power. The CM sets most of SPs at VF_{perf} in *performance mode* and at VF_{EDP} in *energy mode*. In this example, SPs executing tasks t_A and t_C are most of the time in the idle state, because these tasks send data to the processing tasks (t_{B1} to t_{B4}) and receive the processed data, respectively. Thus, in both *performance* and *energy modes*, the SPs executing these tasks use $VF_{\min}$.

6.2. Task mapping

MORM employs two mapping algorithms to meet the different goals of the two operation modes. Besides that, the mapping algorithms also provide adaptability for remapping when the cluster operation mode changes. The mapping algorithms receive the name of the operation modes. *Performance Mapping* is a single-task mapping, which maximizes the parallelism of applications and optimizes the execution time. The *Energy Mapping* employs multi-task mapping to enable the SP sharing between tasks. The *Energy Mapping* follows three constraints: (*i*) communicating tasks may be mapped in the same SP; (*ii*) parallel tasks never share the same SP (as t_{B1} to t_{B4} in Fig. 5); (*iii*) tasks belonging to different applications never share the same SP.

The mapping algorithms benefit from the hierarchical organization to reduce the hop distance between communication tasks. Even if the system is large (e.g., 12x12), the cluster size is typically 4 × 4 [27] to reduce the search space of mapping. The mapping algorithms select the SPs in a spiral order, which is the recommended way to transverse the SPs in 2D-mesh NoCs [34]. Also, the algorithms traverse the application graph using the Breadth-First Search (BFS) algorithm. Finally, the algorithms try to find a contiguous group of free SPs in the spiral path in such a way to avoid interleaving between tasks of distinct applications when possible.

Fig. 6(a)–(c) shows the task mappings for two applications in a 4 × 4 cluster. The numbers labeling the graph vertices sort the tasks for mapping. The system admits the blue application first (Fig. 6(a)). For the applications in Fig. 6(a), for example, *Performance Mapping* produces the single task mapping (Fig. 6(b)) while *Energy Mapping* results in the multi-task mapping (Fig. 6(c)). Note that the Energy Mapping example follows the constraints defined for energy savings.

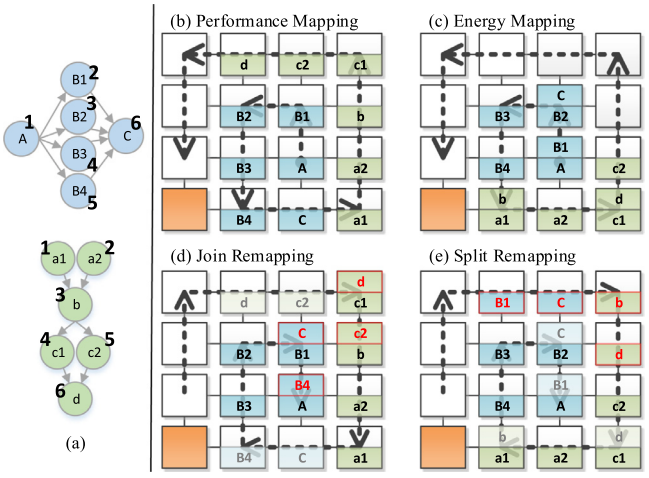


Fig. 6. How MORM maps tasks of two applications into a 4 × 4 cluster for *performance mode* and *energy mode*.

6.3. Task remapping

When the GM changes the operation mode of a cluster, the CM executes task remapping algorithms. When the change is from *performance* to *energy mode, Join Remapping* maps communicating tasks in an SP following the assumptions of *Energy Mapping*. On the other hand, *Split Remapping* algorithm looks for tasks sharing SPs to split their execution using more SPs when changing from *energy mode* to *performance mode*.

Fig. 6(d)-(e) presents the remapping results from mapping examples of Fig. 6(b)-(c). Both algorithms search for tasks to migrate on the opposite spiral order. The new SPs to receive the migrating tasks are chosen in the spiral order. This search order is required to avoid fragmentation between idle SPs and running SPs concerning one application. The fragmentation (in the spiral order) between idle SPs and running SPs can happen between applications, like the remapping shown on Fig. 6(d).

The goals of the mapping and remapping algorithms adopted in MORM are: (*i*) minimization of the task migrations, which is a time-demanding operation; (*ii*) low computational time for updating task mapping for each application and operation mode events. For instance, *Join* remapping (Fig. 6(d)) needs to migrate four tasks (in red). If a new mapping targeting energy was executed, the number of migrations would be higher (eleven in this example).

The literature presents remapping algorithms [21,35] that may lead to better results, regarding hop distance between tasks, than the ones adopted in MORM. The reason to adopt in MORM a simple algorithm comes from the adoption of the hierarchical organization, which reduces the search space, i.e., the heuristic evaluates the cluster resources instead of the entire system.

Note that, task (re)mapping and DVFS deploy a joint effort to drive the cluster according to the operation mode. MORM is aligned with Zhang et al. [17] work, which shows the cooperation of both hardware and software power knobs to optimize performance.

6.4. Cluster power variation computation

The CM provides the estimated power variation value to the GM. Since the cluster operation mode changes dynamically, power variation value is an estimation of power in the opposite operation mode to the current one. The GM employs the estimated power variation on the operation mode selector algorithm (Section 5.1). Algorithm 3 describes how CM estimates the power variation.

When the cluster is running in *energy mode*, the CM estimates the power in *performance mode* from profiling data of all running applications (lines 5–6) - Definition 6 (on page 5). Otherwise, the CM predicts

Algorithm 3 MORM Power Calculator.

```
1:  Input: cl
2:  Outputs: pwrVariation
3:  predictedPwr ← 0
4:  for each app_i ∈ cl.app_set do
5:      if cl.opMode = ENERGY then
6:          predictedPwr+ = app_i.profilePwrPerf
7:      else
8:          predictedPwr+ = app_i.profilePwrEnergy
9:      end if
10: end for
11: pwrVariation ← cl.pwr − predictedPwr
```

the power of all running applications in *energy mode* (lines 7–8). The difference between the current power value from the observing data and the predicted power variation is the estimated power variation value (line 11).

Estimate the power variation before task or application events with accuracy is challenging due to events like NoC traffic, CPU utilization, and memory accesses. A more accurate algorithm should consider all these features, but most of them are unpredictable. Note that, Algorithm 3 also does not consider remapping algorithms may generate different task placements than mapping ones (Fig. 6). Despite that, Algorithm 3 enables MORM to respect the power cap employing a lightweight algorithm since the application power profiling consider worst-case power samplings (Section 3).

7. Results

The experiments use in an in-house clock cycle accurate RTL SystemC model of the reference many-core system. The benchmarks, described in C language, are DTW (6 tasks), AES (5 tasks), MPEG (5 tasks), Dijkstra (7 tasks). Sort (5 tasks), Audio/Video (7 tasks), and Synthetic (6 tasks, communication intensive application).

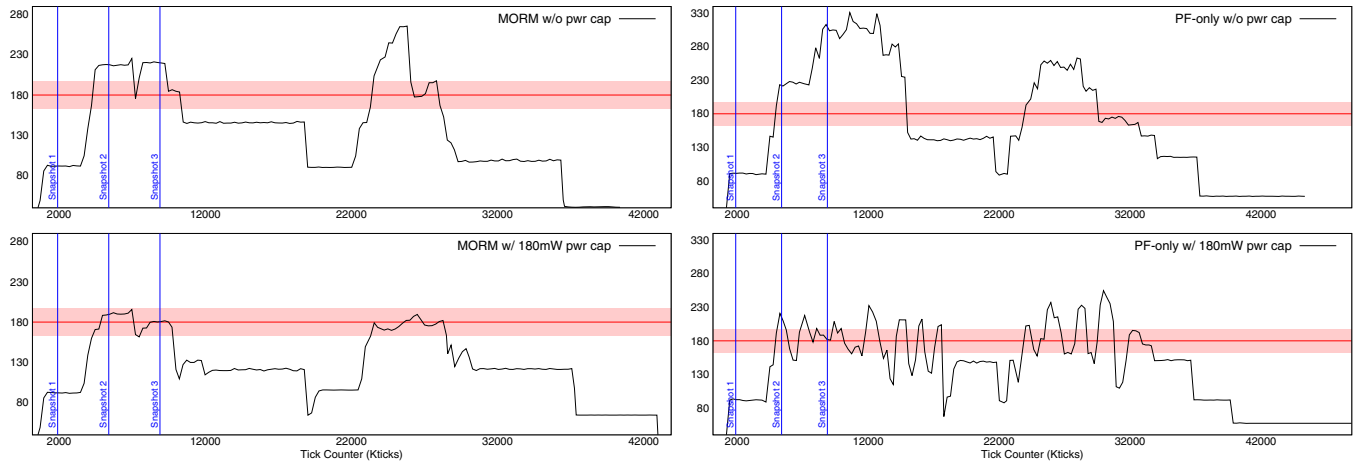
Results compare MORM to state-of-art comprehensive system management targeting dynamic workloads: *performance-objective control* (PF-only). PF-only employs a first node selection [22] for single task mapping aiming congestion reduction [36] and, from the same Authors, a feedback-based PID control that uses DVFS to follow the power cap reference [16]. The first node selection, the mapping algorithm, and the PID control were implemented in the reference platform.

Carrying out a fair comparison between MORM and PF-only requires that both systems share the following assumptions:

- Observing epoch set to 250 Kticks. This value was chosen according to experiments conducted in [27], which represents a trade-off between reaction time of the heuristics and overhead at the manager PEs. Reducing the epoch value may reduce the reaction time of the heuristics once the observed data arrives faster at the manager PE, but these PEs become overloaded to treat incoming packets. The opposite effect occurs increasing the epoch value;
- Both approaches consider the router energy (PF-only does not consider the NoC energy in its original version [16]);
- The SPs running no tasks are considered off (power gating).

The first set of experiments compares MORM and PF-only under a typical workload with peaks and valleys of utilization (Fig. 1). This evaluation details the mapping and *vf-settings* at different moments of the execution to highlight the MORM contributions concerning adaptability and to evidence the actuation differences of both RMs.

The second set of experiments isolates the peak and valley workloads for analysis. Test cases with low workload and high workloads are created to illustrate pros and cons of MORM and PF-only at distinct workload phases. In this experiment, results evaluate power, total execution time, and energy under different power caps.



(a) MORM (b) PF-only

Fig. 7. Average power results for (a) MORM and (b) PF-only running typical workload. The top graphs correspond to RMs running without power cap. The bottom graphs illustrate the power for 180 mW-power cap. Y-axis: power (mW), X-axis: time (in Kticks).

The third set of experiments evaluates the effect of RMs in the average application execution time. The goal of this experiment is to mitigate the overheads for starting applications by focusing only at the application level instead of the workload level, as in the two first experiments.

The fourth set of experiments evaluates MORM operation modes and cluster level actuation. The goal is to demonstrate that MORM trades energy and performance according to the operation mode.

7.1. Typical workload results

To create a scenario corresponding to a typical workload, applications arrive in bursts and can leave the system at any moment. Fig. 7 presents the average power consumption of MORM and PF-only. Top graphs show the average power without the actuation of any RM. Bottom graphs illustrate the average power of MORM and PF-only under a power cap of 180mW. The vertical lines in Fig. 7 correspond to snapshots taken at the same execution time for all simulations after a burst of two applications entering into the system. Thus, after the third burst, six applications are executing in the many-core.

When the RMs are executing without a power cap, the task mapping/remapping (TM/TR) algorithms are active, and the only restriction for admitting an application is the availability of resources. Since PF-only and MORM employ distinct algorithms for AA (Application Admission), TM and TR, they generate distinct power graphs for the same scenario. Although the distinct power graphs, the top graphs present power peaks and valleys for both RMs. In Snapshot 1 and 2, the average power for MORM and PF-only are similar. However, MORM power stays constant after the third burst, while PF-only power increases in Snapshot 3. The peak of MORM power in Snapshot 3 is smaller than PF-only power because MORM sets two clusters to *energy mode* to open resources room for two more applications. Therefore, after Snapshot 3, MORM executes four applications in a reduced number of PEs by employing multitasking while PF-only keeps more PEs active to execute the same six applications. As a result, MORM is 10.66% faster than PF-only concerning the simulation without power cap.

When the RMs are executing under a 180 mW-power cap, the proactive actuation adopted in the *Operating Mode Selector* allows MORM to run with no violations, while the reactive PID controller in PF-only presents power violations after a burst of applications. On the other hand, these overshoots of power for PF-only might not be a real problem due to the thermal inertia [3].

Fig. 8 details the snapshots under a 180mW-power cap. Snapshot 1 (Fig. 8(a)) shows the system status after the first burst. Due to the low workload at this moment, both RMs execute the two applications at VF_{perf} and still produce power slack (Fig. 7). Note that MORM can eventually apply $VF_{\min}$ in the SPs to save energy even with the cluster in *performance mode* by checking HI and LU thresholds.

At the second snapshot (Fig. 8(b)), MORM distributes the workload between clusters, with three clusters running in *energy mode* and one in *performance mode* (with some tasks in $VF_{\min}$), with some SPs running two tasks (multi-task mapping). PF-only selects two applications to run in VF_{perf}, one in VF_{EDP}, and one in $VF_{\min}$. On the other side, MORM enables three applications to run in VF_{EDP} when selecting *energy mode* for the three clusters due to its multi-objective cost-function (tradeoff between energy and performance). Besides the update of *vf-settings*, MORM average power meets the power cap due to reduced leakage power, i.e., a lower number of SPs are active in MORM than in PF-only. The main difference observed at Snapshot 2 is that MORM enables all applications to run at VF_{perf} and VF_{EDP} while PF-only sets one application to run at $VF_{\min}$ to respect the power cap. As a result, PF-only penalizes the performance of applications by executing them in $VF_{\min}$ to meet the power cap. Note that, MORM manages applications by the joint actuation of TR, DVFS, and PG (power gating) while PF-only employs only DVFS.

Snapshot 3 (Fig. 8(c)) illustrates both systems after the third burst. Due to power cap violation at the third burst (Fig. 7), all SPs shift to $VF_{\min}$ in PF-only. Such actuation penalizes the performance of all applications in PF-only. In MORM, shifting all clusters to *energy mode* is enough to meet the power cap because the *energy mode* allows joining applications' tasks in one SP and, consequently, generates more idle SPs to turn off. For instance, Snapshot 3 shows similar power in MORM and PF-only, but MORM has 22 active SPs running in VF_{EDP} while PF-only has 33 active SPs running in $VF_{\min}$ for the same workload.

Summarizing, despite the power overshoots identified in PF-only, both RMs respect the power cap while executing a typical workload (Fig. 7). Snapshots illustrate that increasing the number of tasks makes both RMs actuate on power and reveals the differences between both approaches. To meet the power cap, MORM employs a joint actuation of TR, PG, and DVFS at the cluster level, while PF-only performs a per-application DVFS without modifying the applications' mapping. An advantage of MORM is the fact that it may execute more tasks simultaneously than PF-only due to the multi-task mapping. Observe in Fig. 8(c) that some processors are off. Thus, only MORM could admit new applications if power is below the cap.

7.2. Low and high workload evaluation

This Section evaluates MORM and PF-only by isolating the peaks and valleys of workload. In *low* and *high* workload scenarios, the many-

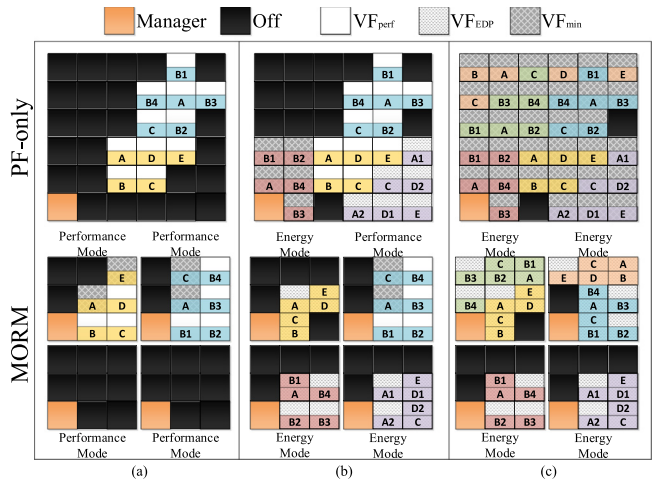


Fig. 8. System snapshots (6 × 6 many-core) taken according the moments highlighted in Fig. 7 to detail the task allocation and *vf-settings*.

core loads applications at the beginning of the simulation in such a way that the number of executing tasks corresponds to 50% and 125% of the number of SPs, respectively. The *low workload* scenario corresponds to the valley behavior while the *high workload* corresponds to the peak one. The experimental setup of this Section contains four scenarios: (*i*) *LW* – low workload; (*ii*) *HW*210 – high workload, power cap equal to 210 mW; (*iii*) *HW*180 – high workload - power cap equal to 180 mW; (*iv*) *HW*150 – high workload - power cap equal to 150 mW.

Fig. 9 illustrates the average power consumption for the four evaluated scenarios. The first row of power graphs compares the power for the *LW* scenario, while the remaining three rows show the *HW*s ones. The red shaded area represents the power cap.

In the *LW* scenario, the power graphs present a similar behavior up to 30,000 Kticks because all applications execute in VF_{perf} in PF-only, and all clusters are in *performance mode* in MORM. After 30,000 Kticks, applications finish their execution. The longer execution time of the system managed by PF-only comes mainly to the different mapping heuristics (PF-only adopts a more complex heuristic, which has main cost function the hop distance reduction), and higher monitoring data due to the centralized management.

In the *HW* scenarios, both RMs respect the power cap. Note that, the execution time reduces when the power cap increases since the relaxed power constraint allows more applications to run in *performance mode* (MORM) and in VF_{perf} (PF-only).

MORM power curves in *HW* scenarios meet the power cap, but the power presents underutilization. The power underutilization comes from the worst-case power predictions (Section 3) and the proactive-only actuation. At 210 mW power cap, all clusters run in *energy mode* to open room for all applications. Thus, MORM cannot take advantage of a relaxed power cap to speed-up some cluster. As a consequence, *HW*210 presents the largest power underutilization. Concerning the smallest power caps (150 and 180 mW), the power curves stay steady longer and closer to the cap because the *Application Admission* proactively blocks the allocation of a new application to avoid a power violation. At 150 mW and 180 mW power cap, a period without samples is observed in MORM (5,000-35,000 Kticks at 150mW, and 10,000-30,000 Kticks at 180mW) but without power cap violations.[2]

PF-only power curves in *HW* scenarios stay near to the power cap most of the time, despite some overshoots. The oscillation near to the cap line occurs because the PID control is constantly setting *vf-pairs*. This behavior in *HW* has two reasons: (*i*) the actuation at the application level, which creates a larger impact in power compared to the SP grain adopted by MORM; (*ii*) the NoC congestion is higher in *HW* than in *LW* since the NoC traffic has a correlation with *vf-settings* (higher *vf-pairs* increases the packet injection rate) [16] and the system load.

Fig. 10 compares the execution time, energy and EDP (Energy-Delay Product), normalized to the PF-only results. In *LW* (Fig. 10(a)), MORM is 5.93% faster because the adoption of a hierarchical management reduces the NoC traffic traversing the applications. For instance, for the same scenario with an observing epoch eight times larger, PF-only is 3.22% faster than MORM and 16.72% faster than PF-only with the original epoch settings. Further, MORM and PF-only present similar energy consumption because the number of tasks and the number of active SPs are the same for both RMs. As a result, MORM is 7.47% more efficient at *LW*.

In *HW* scenarios, the smaller execution time in MORM comes from: (*i*) hierarchical organization; (*ii*) adaptability; and (*iii*) comprehensiveness. Concerning adaptability, PF-only selects some applications to run in VF_{min} (as depicted in Fig. 8(c)) while MORM can reduce the power and delays by joining tasks in the same SP running in VF_{EDP}. Concerning comprehensiveness, MORM applies a joint actuation of TR, PG, and DVFS while PF-only employs DVFS to deal with power cap. For instance, Snapshot 3 (Fig. 8(c)) evidences the contrast between both RMs in HW and reveals the benefit of adopting multi-objective management in a peak of workload: MORM respects the power capping without penalizing the applications' performance. Furthermore, MORM can map all applications in *HW* scenarios due to the multitasking feature while PF-only has to wait for available SPs.

Although MORM total execution time is significantly lower than PF-only, the energy consumption savings do not follow the same trend. The reason comes from the fact that PF-only executes most of the tasks in VF_{min}, which has the lowest leakage, and demands one task per PE while MORM runs most of the time in VF_{EDP}, which has an intermediate leakage but uses a smaller number of PEs than PF-only due to the multitasking mapping.

Concluding, MORM is superior compared to PF-only at workload peaks, as shown by the EDP bars in Fig. 10. It is worthwhile to mention that MORM has space for optimization because the power can stay below the cap for extended periods.

7.3. Application level evaluation

Previous Sections evaluated RMs regarding execution time, energy, and power at the system level by comparing MORM and PF-only at peaks and valleys of workload. This section evaluates the average execution time of applications considering their start and end times. Although applications require admission at the same time, they start their execution at different moments due to the different mapping heuristics and resources availability. Fig. 11 presents the normalized average execution time per application for the same scenarios evaluated in Section 7.2.

At *LW*, applications that exchange data intensively (communication profile) may present a smaller execution time with PF-only since its mapping heuristic has as main cost function the hop distance reduction. For instance, Dijkstra is 11.88% faster with PF-only, while MPEG (computation profile) is 14.11% slower.

In *HW* scenarios (Fig. 11(b-d)), MORM executes applications 18.62% faster than PF-only in average. Two reasons explain this result. First, PF-only set some application to run at VF_{min} in *HW* to meet the power cap, penalizing the application execution time. Second, as PF-only uses centralized management, the NoC traffic increases with the system load, creating congested areas in the NoC. Thus, the advantage observed in the *LW* scenario is not observed in the *HW* because the increasing in NoC traffic penalizes applications regardless of the application profile. Regarding the power cap variation in *HW* scenarios, the applications get

[2] In our framework, the GM does not register power samples when an application is waiting for admission. However, the power measures per cluster (not shown in the graph) guarantee the power cap is met and provide data for decisions.

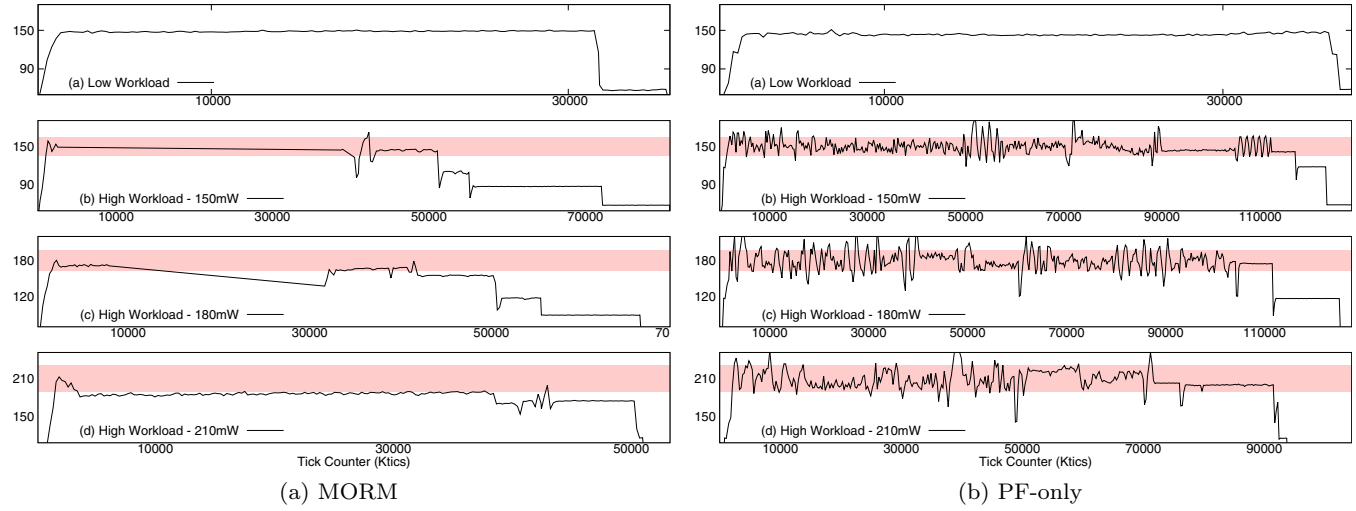


Fig. 9. Average power results for MORM and PF-only running low and high workloads. Y-axis: power (mW), X-axis: time (in Kticks).

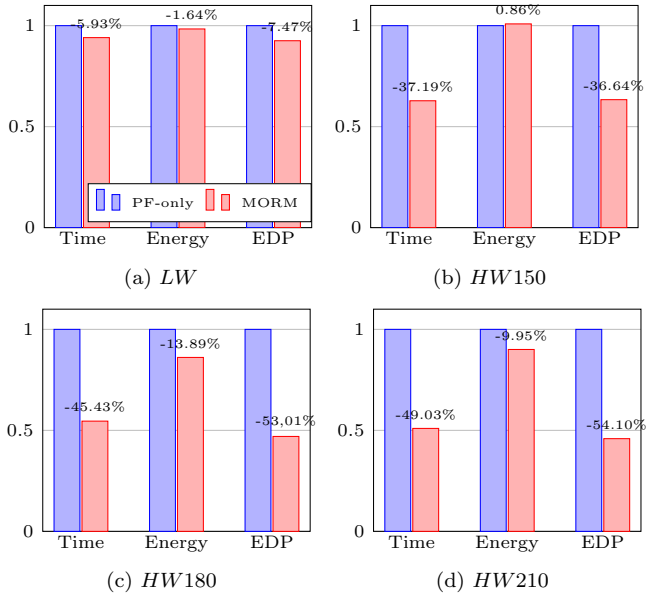


Fig. 10. System level results concerning execution time, energy and EDP. The normalized histograms depict the results for the scenarios that represent valleys (a – low workload) and peaks (b-d – high workload) of workload concerning power capping variation.

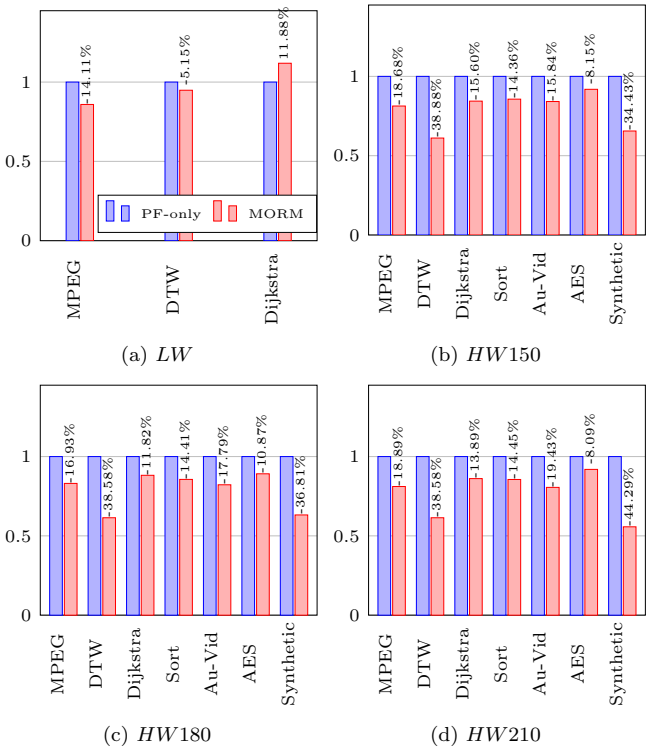


Fig. 11. Execution time results per application, for low and high workload scenarios, normalized w.r.t the PF-only method.

a slight traffic rise in more relaxed caps (*HW*210) because processing tasks in low frequencies reduces network congestion.

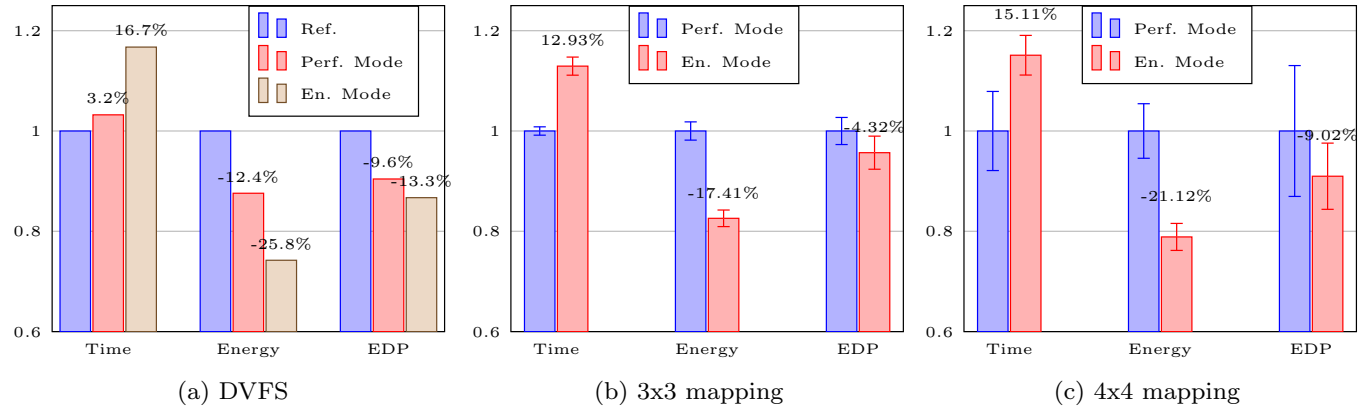
Concluding, the applications' performance in *LW* scenarios relies on the application profile. PF-only can reduce the execution time of applications with a communication profile. On the other hand, in *HW* scenarios the hierarchical approach and the comprehensive actuation employed in MORM prevent losses in the performance of all applications, for all power caps, regardless of the application profile.

7.4. Cluster level results

This Section evaluates the MORM actuation at the cluster level, by isolating the DFVS and the mapping contributions.

Fig. 12(a) compares the cluster using the reference platform (no actuation, *vf*-nominal) regarding to a cluster at *performance mode* and *energy mode*. Both operation modes use the same mapping and the DVFS is the only actuation. The *performance mode* slightly increases the execution time because MORM can apply VF_{min} to SPs by identifying opportunities in tasks phases to reduce the total energy consumption. In *energy mode* the execution time increases because MORM applies VF_{EDP} to the SPs to prioritize the energy efficiency. Both operation modes present better EDP than the reference platform.

Fig. 12(b) and (c) evaluate task mapping algorithms for each operation mode considering two cluster sizes, normalized regarding the *performance mode*. For both modes, five different mappings are considered. Results confirm that multi-task mapping (*energy mode*) is suitable for energy savings while the single task mapping is suitable for boosting performance (*performance mode*). 3 × 3 cluster results reveal a small standard deviation since the smaller cluster size restricts the mapping search space (Fig. 12(b)). Although 4 × 4 cluster results produce a larger stan-



(a) DVFS (b) 3x3 mapping (c) 4x4 mapping

Fig. 12. Cluster level results. DVFS (a) results normalized regarding the reference scenario, for *performance* and *energy modes* (e.g.: Fig. 5). Average and standard deviation mapping results for (b) 3 × 3 and (c) 4 × 4 clusters normalized regarding the *performance mode*.

dard deviation, the operation mode is correctly driven for both cases. The standard deviation for *performance mode* is larger in 4 × 4 clusters since the traffic creates some congestion on the NoC because some mappings increase the distance between communicating tasks. The congestion effect in the *energy mode* is smaller because the multi-task mapping reduces the NoC traffic.

Concluding, this Section demonstrates that a cluster in *performance mode* actually prioritizes performance as well as a cluster in *energy mode* emphasizes energy. Besides, the adaptability allows MORM to trade performance and energy at runtime.

8. Conclusions

This work proposed the resource management for many-core systems called MORM. MORM can dynamically adapt the running applications according to peaks and valleys of workload inherent to real systems while guaranteeing the power cap. The cluster level adaptability, named as Operation Mode, allows distinct areas of the many-core to execute under different goals: energy or performance. MORM jointly coordinates DVFS, mapping, and remapping to carry-out cluster adaptability at runtime. At the system level, if power and resources are available, the proactive MORM heuristics admit an application and choose the adequate cluster and operation mode.

The management of operation modes at the cluster level stands out from related works since clusters can dynamically prioritize a given objective, as long as the power cap is met. Considering a typical dynamic workload of many-core systems, MORM can run applications up to 49% faster in a workload peak compared to a single-objective approach optimized for performance.

Future works include: (*i*) include reactive actuation to avoid power underutilization; (*ii*) propose a method for estimating the power of tasks at runtime; (*iii*) include additional operation modes to MORM to meet goals such as reliability, lifetime, and security; (*iv*) include in the heuristics the effect of the temperature with the goal to avoid hotspots.

Supplementary material

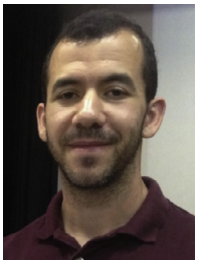
Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.sysarc.2019.01.006.

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.
[2] M.-H. Haghbayan, A.-M. Rahmani, A.Y. Weldezion, P. Liljeberg, J. Plosila, A. Jantsch, H. Tenhunen, Dark silicon aware power management for manycore systems under dynamic workloads, in: ICCD, 2014, pp. 509–512.
[3] A.M. Rahmani, P. Liljeberg, A. Hemani, A. Jantsch, H. Tenhunen, Dark Side of Silicon, Springer International Publishing, AG Switzerland, 2016 10.1007/978-3-319-31596-6.
[4] A.K. Singh, P. Dziurzanski, H.R. Mendis, L.S. Indrusiak, A survey and comparative study of hard and soft real-Time dynamic resource allocation strategies for multi-/many-Core systems, ACM Comput. Surv. 50 (2) (2017) 24:1–24:40.
[5] H. Khdr, S. Pagani, M. Shafique, J. Henkel, Thermal constrained resource management for mixed ILP-TLP workloads in dark silicon chips, in: DAC, 2015, pp. 1–6.
[6] 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.
[7] 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.
[8] A. Pathania, H. Khdr, M. Shafique, T. Mitra, J. Henkel, Scalable probabilistic power budgeting for many-cores, in: DATE, 2017, pp. 864–869.
[9] M. Shafique, B. Vogel, J. Henkel, Self-adaptive hybrid dynamic power management for many-core systems, in: Proceedings of the Conference on Design, Automation and Test in Europe, EDA Consortium, 2013, pp. 51–56.
[10] H. Khdr, S. Pagani, E. Sousa, V. Lari, A. Pathania, F. Hannig, M. Shafique, J. Teich, J. Henkel, Power density-aware resource management for heterogeneous tiled multicores, IEEE Trans. Comput. 66 (3) (2017) 488–501.
[11] X. Meng, C. Isci, J.O. Kephart, L. Zhang, E. Bouillet, D.E. Pendarakis, Efficient resource provisioning in compute clouds via VM multiplexing, in: ICAC, 2010, pp. 11–20.
[12] NVIDIA Corporation, Variable SMP - A multi-core CPU architecture for low power and high performance, Technical Report, NVIDIA Corporation, 2011.
[13] NVIDIA Corporation, The benefits of quad core CPUs in mobile devices, Technical Report, NVIDIA Corporation, 2011.
[14] H. Liu, A measurement study of server utilization in public clouds, in: DASC, 2011, pp. 435–442.
[15] A.M. Rahmani, A. Jantsch, N. Dutt, HDGM: hierarchical dynamic goal management for many-Core resource allocation, IEEE Embed. Syst. Lett. PP (99) (2017) 1–4.
[16] 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.
[17] H. Zhang, H. Hoffmann, Maximizing performance under a power cap: a comparison of hardware, software, and hybrid techniquess, ACM SIGPLAN Notices 51 (4) (2016) 545–559.
[18] D. Olsen, I. Anagnostopoulos, Performance-aware resource management of multi-threaded applications on many-core systems, in: GLSVLSI, 2017, pp. 119–124.
[19] T. Kim, Z. Sun, H.-B. Chen, H. Wang, S.X.-D. Tan, Energy and lifetime optimizations for dark silicon manycore microprocessor considering both hard and soft errors, IEEE Trans. Very Large Scale Integr. VLSI Syst. 25 (9) (2017) 2561–2574.
[20] A. Kanduri, M.-H. Haghbayan, A.M. Rahmani, P. Liljeberg, A. Jantsch, H. Tenhunen, N. Dutt, Accuracy-aware power management for many-core systems running error-resilient applications, IEEE Trans. Very Large Scale Integr. VLSI Syst. 25 (10) (2017) 2749–2762.
[21] A. Pathania, V. Venkataramani, M. Shafique, T. Mitra, J. Henkel, Defragmentation of tasks in many-core architecture, ACM Trans. Archit. Code Optim. 14 (1) (2017) 21.
[22] M. Fattah, M. Ramirez, M. Daneshtalab, P. Liljeberg, J. Plosila, CoNA: Dynamic application mapping for congestion reduction in many-core systems, in: ICCD, 2012, pp. 364–370.
[23] M. Mandelli, L. Ost, G. Sassatelli, F. Moraes, Trading-off system load and communication in mapping heuristics for improving NoC-based MPSoCs reliability, in: ISQED, 2015, pp. 392–396.
[24] A. Kanduri, M.-H. Haghbayan, A.-M. Rahmani, P. Liljeberg, A. Jantsch, H. Tenhunen, Dark silicon aware runtime mapping for many-core systems: a patterning approach, in: Computer Design (ICCD), 2015 33rd IEEE International Conference on, IEEE, 2015, pp. 573–580.

[25] M. Ruaro, F.G. Moraes, Demystifying the cost of task migration in distributed memory many-core systems, in: ISCAS, 2017, pp. 148–151.
[26] W. Quan, A.D. Pimentel, A hierarchical run-time adaptive resource allocation framework for large-scale MPSoc systems, Design Automat. Embedded Syst. 20 (4) (2016) 311–339.
[27] 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.
[28] K. Moazzemi, A. Kanduri, D. Juhász, A. Miele, A.M. Rahmani, P. Liljeberg, A. Jantsch, N. Dutt, Trends in on-chip dynamic resource management, in: 2018 21st Euromicro Conference on Digital System Design (DSD), IEEE, 2018, pp. 62–69.
[29] A.L. Martins, D.R. Silva, G.M. Castilhos, T.M. Monteiro, F.G. Moraes, A method for NoC-based MPSoC energy consumption estimation, in: ICECS, 2014, pp. 427–430.
[30] N. Chatterjee, S. Paul, P. Mukherjee, S. Chattopadhyay, Deadline and energy aware dynamic task mapping and scheduling for network-on-Chip based multi-core platform, J. Syst. Archit. 74 (2017) 61–77.
[31] M. Fattah, M. Daneshtalab, P. Liljeberg, J. Plosila, Smart hill climbing for agile dynamic mapping in many-core systems, in: DAC, 2013, pp. 1–6.
[32] N. Dutt, A. Jantsch, S. Sarma, Self-aware cyber-physical systems-on-chip, in: ICCAD, 2015, pp. 46–50.
[33] A.L. Martins, A.C. Sant'Ana, F.G. Moraes, Runtime energy management for many–core systems, in: ICECS, 2016, pp. 380–383.
[34] N. Bansal, S. Gupta, N. Dutt, A. Nicolau, R. Gupta, Network topology exploration of mesh-based coarse-grain reconfigurable architectures, in: DATE, 2004, pp. 474–479.
[35] J. Ng, X. Wang, A.K. Singh, T. Mak, Defrag: Defragmentation for efficient runtime resource allocation in noc-based many-core systems, in: Parallel, Distributed and Network-Based Processing (PDP), 2015 23rd Euromicro International Conference on, IEEE, 2015, pp. 345–352.
[36] M.-H. Haghbayan, A. Kanduri, A.-M. Rahmani, P. Liljeberg, A. Jantsch, H. Tenhunen, Mappro: Proactive runtime mapping for dynamic workloads by quantifying ripple effect of applications on networks-on-chip, in: NOCs, 2015, pp. 1–8.

André Luís del Mestre 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-rio-grandense Federal Institute of Education (IF-Sul). His main research interest includes multiprocessor systems on chip (MPSoCs) and networks on chip (NoCs).



Alzemiro Henrique Lucas da Silva received the M.Sc. degree in 2010 from the Pontifical Catholic University of Rio Grande do Sul (PUCRS), and currently is a Ph.D. candidate at the same University. He worked for 7 years in industry as a digital FPGA engineer and software development for network protocols. His main research interest includes multiprocessor systems on chip (MPSoCs) and energy efficient computing.

Amir M. Rahmani (SM) received his Masters degree from Department of ECE, University of Tehran, Iran, in 2009 and Ph.D. degree from Department of IT, University of Turku, Finland, in 2012. He is currently Marie Curie Global Fellow at University of California Irvine (USA) and TU Wien (Austria). His research interests span Self-aware Computing, Energy-efficient Many-core Systems, Runtime Re-source Management, Healthcare Internet of Things, and Fog/Edge Computing. He has served on a large number of technical program committees of international conferences, and guest editor for special issues in journals. He is the author of more than 150 peer-reviewed publications.

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 Gehm Moraes (SM12) 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 dInformatique, 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.