0-1 ILP-based Run-Time Hierarchical Energy Optimization for Heterogeneous Cluster-based multi/many-core Systems

Simei Yang, Sébastien Le Nours, Maria Mendez Real, Sébastien Pillement

Version of Record: https://www.sciencedirect.com/science/article/pii/S1383762121000370
Manuscript_04e952ecddd9c25b00f862096a1655ac

0-1 ILP-based Run-Time Hierarchical Energy Optimization for Heterogeneous Cluster-based multi/many-core Systems

Simei Yang*, Sébastien Le Nours, Maria Mendez Real and Sébastien Pillement

University of Nantes, CNRS, IETR UMR 6164, F-44000 Nantes, France

ARTICLE INFO

Keywords:
0-1 ILP
run-time management
energy efficiency
task mapping
per-cluster DVFS

ABSTRACT

Heterogeneous cluster-based multi/many-core platforms are on the edge, delivering high computing and energy-efficient embedded systems. These platforms support Dynamic Voltage/Frequency Scaling (DVFS), allowing to change the voltage/frequency levels for each cluster independently. Mapping dynamic applications on such platforms at run-time is a tedious task. This article presents a 0-1 Integer Linear Programming (ILP) based run-time management approach that aims to optimize the overall system energy. The proposed approach adopts a hierarchical management organization. A global management strategy determines application-to-cluster assignments and setups the cluster frequency configurations. A local management strategy determines task-to-core mapping in each cluster to minimize resource usage. Our approach achieves optimized solutions with reduced complexity and shows good scalability on different platform sizes. The experimental results show that, compared with the state-of-the-art approaches of similar complexity, the proposed global management strategy can reduce the average power consumption of the overall system by 80.3%. The experiment also demonstrates that resource minimization in the local management can significantly impact global management decisions, and thereby further reducing overall average power by up to 60.72%.

1. Introduction

Heterogeneous cluster-based multi/many-core platforms represent promising solutions to deliver high computing performance and energy efficiency in modern embedded systems. On such platforms, cores are grouped into clusters, while distinct features of different core types can be exploited in the different clusters. One example of such a platform is the Exynos 5 Octa (5422) [1] with ARM's big.LITTLE architecture composing of a LITTLE cluster (quad-core Cortex-A7) and a big cluster (quad-core Cortex-A15). As reported in [2], the performance of Cortex-A15 is twice that of Cortex-A7, but with a power consumption 4 times higher (in average for different calculation types, *e.g.*, floating-point and integer). It is expected that more clusters/cores will be largely adopted in future systems [2, 3], such as in automotive and IoT domains. For power consumption optimization, these platforms often support per-cluster Dynamic Voltage/Frequency Scaling (DVFS), allowing each cluster to run under its own voltage/frequency (v/f) levels independently. The different computation resources among clusters and the allowable v/f cluster configurations allow systems to achieve a good compromise between computation performance and power consumption. That is one of the reasons why the ARM big.LITTLE architecture is widely used, for instance in the field of smartphones [4].

The increasing number of cores on modern platforms provides opportunities to execute more applications at the same time. As defined in [5], we adopt the notion of *use-case* to designate a set of applications that are active concurrently on the system. From one use-case to another, workloads on the platform resources change dynamically over time. The increasing application complexity and dynamism in multi/many-core platforms arise the need for resources management. This management should optimize the energy efficiency of the overall system with acceptable computation complexity. The management process includes appropriate mapping activate applications (*i.e.*, application-to-cluster assignments, task-to-core mapping within a single cluster) and, for energy optimization purpose, applying DVFS (*i.e.*, selecting v/f configurations for each cluster) to the system [3].

Obviously, application mapping and cluster v/f configurations directly influence the overall energy consumption of the system. In cluster-based multi/many-core platforms, small changes in the application mapping can significantly change the cluster v/f configurations, and consequently increase/decrease the energy consumption of the overall system. Obtaining the optimal application mapping under system constraints is known as a NP-hard problem [6]. The solution space exploration increases with the number of applications and the number of cores. The exploration space becomes even larger when DVFS is taken into account. This problem is even more complex when dealing with heterogeneous platforms.

To explore mapping and DVFS solutions, two management approaches can be applied. Design-time management approaches fix the system configuration (*i.e.*, task mapping, and v/f level) while the system is designed (or before system execution), and thus cannot handle application dynamism. Run-time management approaches intend to avoid the limitations of design-time management [7, 8], by adapting tasks mapping and DVFS settings to the dynamic workloads of a system. When performing run-time management on a heterogeneous cluster-based system, two problems need to be addressed.

The first problem is how to apply task mapping and DVFS effectively to optimize the energy consumption of the overall system? First approaches, such as [9, 10], apply first

the task mapping and then configure the DVFS *separately*. If this approach simplifies the optimization problem, it might lead to smaller energy efficiency of the overall system. This is because the mapping can be further optimized after the v/f level changes. A more energy-efficient solution is to apply task mapping and DVFS *integratedly* [7, 8]. Any change in the mapping can trigger an update of v/f levels under system constraints. Such approaches take into account the mutual influence between mapping and DVFS. As [7, 8] target one application executing on a multi-core system, a new *integrated* approach is required to consider multiple applications executing concurrently.

The second problem is how to make the proposed run-time management strategy scalable according to platform sizes (e.g., different numbers of clusters, different numbers of cores within a cluster)? A hierarchical management structure [11] is known as scalable. Such a structure mainly uses 2-level hierarchical management, consisting of a global and a local management strategy. Generally, the global management considers a monolithic application-to-cluster assignment approach [9, 12, 13]. These solutions are proposed for small-heterogeneous cluster-based platforms (*e.g.*, ARM big.LITTLE, with 4 cores within a cluster), and they might present scalability issues for many-core platforms encompassing more clusters or more cores within each cluster (as can be seen in Section 6). A new application-to-cluster allocation strategy is needed to achieve energy efficiency on different platform sizes with acceptable computational complexity.

Our work aims to propose a run-time management approach for heterogeneous cluster-based multi/many-core platforms (supporting per-cluster DVFS). We consider several applications running concurrently on top of the platform. As the management needs to be scalable, the approach considers a 2-level hierarchical management structure. The mapping process and DVFS setting are handled *integratedly* in order to achieve energy efficiency of the overall system. More precisely the contributions of this work are:

- We formulate the dependencies between task mapping and cluster v/f into a 0-1 Integer Linear Programming (ILP) model. This model is the base to *integratedly* handle the mapping and the cluster v/f settings.
- To support scalability issues and achieve an optimized solution to the 0-1 ILP problem with acceptable exploration effort, we propose a greedy search strategy in the global management. The global strategy assigns applications with similar timing constraints into the same clusters, and keeps cluster v/f as low as possible. Besides, the proposed strategy allows a configurable number of applications to be migrated from one cluster to another.
- To minimize the resource usage within each cluster, we propose a local management strategy to optimize task-to-core mapping. The local management allows the global management to handle the heterogeneity of the platform more efficiently. The reduction of resource usage can impact the decision of the global management, and consequently leads to better energy efficiency of the overall system.
- We evaluate our strategies for different use-cases (*i.e.*, different sets of simultaneously active applications) and on different platform sizes. The experiments compare the proposed strategies to state-of-the-art approaches in terms of energy efficiency, scalability and strategy complexity.

The remainder of this paper is organized as follows. Section 2 discusses related work on run-time management for heterogeneous cluster-based multi/many-core systems. Section 3 then presents the considered application, platform and power/energy models. Section 4 defines the problem overview of the hierarchical management, while the Section 5 introduces the proposed hierarchical run-time management strategy. Section 6 discusses the experimental results. Finally, Section 7 concludes this paper.

2. Related works

As introduced above, run-time management in heterogeneous cluster-based multi/many-core systems includes two tasks: 1) performing a dynamic mapping of applications to resources and 2) setting the best values of voltage and frequency to optimize energy consumption (*i.e.* by using DVFS techniques). These two steps can be applied *separately* or *integratedly*. The former way considers the two management techniques into two independent steps, and the latter way makes management decisions considering the potential impact of one technique on the other one.

In cluster-based systems, task mapping needs be considered at cluster-level and at core-level. In [3, 9, 12] authors separate cluster-level and core-level mapping from DVFS. These works apply simple cluster-level mapping strategies, and then achieve workload balance within each cluster. Based on the selected mapping, cluster v/f levels are decreased as much as possible under timing constraints or power budgets. Particularly at cluster-level, the authors of [3] applies a Low-Energy-First (LEF) mapping strategy. It assigns each *independent periodic task* to the cluster that can achieve the lowest energy consumption at the maximum cluster v/f level. This work holds the hypothesis that the energy consumption of a task at a given v/f level only depends on the used core type, meaning that different mapping priorities are given to different clusters (at the maximum cluster v/f levels). Similarly, the works [9, 12] also give different priorities to different clusters when mapping applications onto the ARM big.LITTLE platforms. They use a Low-Power-First (LPF) based application-to-cluster assignment strategy, where each application attempts to be assigned to the lowest-power cluster (*i.e.*, with the little cores) first. If the performance constraint is not satisfied, the high-performance cluster (*i.e.*, big core cluster) is used. Both LEF and LPF strategies give the highest priority to

one cluster type during cluster-level mapping. For platforms with more clusters or more cores within a cluster, we can predict that the workload of the highest-priority clusters might be very heavy with increasing numbers of active applications. Furthermore, low-priority clusters might be empty without any executed application, or significantly under used, thus missing the opportunity for further energy optimization.

In [13–15], the authors consider the cluster-level mapping independently, but *integrate* the core-level mapping and DVFS within each cluster (for ARM big.LITTLE based systems). In [13], every new active application is first mapped onto the little cluster, corresponding to the LPF strategy. Then, within each cluster, this work considers the mutual influence of task mapping and DVFS by estimating the performance (*i.e.*, throughput) gain/loss of each application according to different management decisions. This work first applies the DVFS technique *i.e.*, decrease cluster v/f levels to achieve power gain, and estimates the resulting system performance. If the estimated result at the new v/f level violates performance constraints, the task mapping is performed to try to compensate for the performance loss. Besides, the works in [14, 15] assume that application-to-cluster assignments are known at design-time, and apply the two steps of management through iterative evaluations within each cluster. These works try different mappings and DVFS configurations in each cluster until all active applications are successfully mapped at the lowest cluster v/f level. However, as these works focus on local optimization within each cluster, they cannot benefit from dynamic global optimization at the overall system level.

In contrast to previous works, the authors of [8, 16] integrates mapping and DVFS at both, cluster and core levels to achieve more energy-efficient solutions. In [8], a genetic algorithm is presented to characterize task mapping and v/f levels decisions on the same chromosome. This work targets design-time management strategies for a single application, while our work considers a run-time management strategy for multiple applications executing concurrently. A recent work [16] presents a machine-learning based approach to *integratedly* manage task mapping and DVFS on an Arm big.LITTLE platform. At design-time, this work explores the best configurations (*i.e.*, the number of used cores and frequency level in each cluster) for the dynamic workload of each application. Then, it uses a Neural Network to build the relationship between workloads and these configurations. At run-time, the neural network chooses between different configurations according to the execution context. This work provides a new way to achieve system optimization for multiple applications that execute sequentially, where one application executes after another. On our side, our work considers multiple applications executing concurrently.

Table 1 summarizes the strategies of previous works and also compares the management structures (last column) of the different approaches. The centralized structure adopted in [3, 8, 12, 13, 16] enables management simplicity, but it may introduce severe performance bottlenecks and heavy computation burdens for complex systems. A better scalability can be provided by distributed and hierarchical managements. By using a distributed structure, the works in [14, 15] divide the entire management problem into several sub-problems to achieve local optimization on each sub-system (cluster). Hierarchical management considers system optimization at different levels, having the possibility to achieve both local and global optimization. The work in [9] optimizes the system at the system-level, the cluster-level, and the task-level.

Table 1: Comparison of related management strategies in heterogeneous cluster-based systems

<table>
<tr><th>Ref</th><th>Cluster-based Platform</th><th>Apply Mapping and DVFS</th><th>Application-to-cluster</th><th>Management Structure</th></tr>
<tr><td>[3]</td><td>different sizes (1-task apps)</td><td>separated</td><td>LEF</td><td>centralized</td></tr>
<tr><td>[9]</td><td rowspan="5">ARM big.LITTLE 2clusters x 4cores (multi-task apps)</td><td rowspan="2">separated</td><td>LPF</td><td>hierarchical</td></tr>
<tr><td>[12]</td><td>LPF</td><td>centralized</td></tr>
<tr><td>[13]</td><td rowspan="3">integrated in each cluster</td><td>LPF</td><td>centralized</td></tr>
<tr><td>[14]</td><td>Known</td><td rowspan="2">distributed</td></tr>
<tr><td>[15]</td><td>Known</td></tr>
<tr><td>[8]</td><td>2clusters x 2cores (single app)</td><td rowspan="3">integrated among clusters and integrated in each cluster</td><td>Genetic</td><td rowspan="2">centralized</td></tr>
<tr><td>[16]</td><td>ARM big.LITTLE (sequential apps)</td><td>Machine-learning</td></tr>
<tr><td>This work</td><td>different sizes (concurrent apps)</td><td>0-1 ILP model</td><td>hierarchical</td></tr>
</table>

Our work achieves scalability by a 2-level hierarchical management structure as in [11], targeting global optimization at the cluster-level and local optimization at the core-level. It is worth noting this work considers a different architecture (*i.e.*, homogeneous at cluster-level, heterogeneous at core-level, no DVFS consideration) to our work. Besides, the work of [17] also uses a hierarchical management structure on many-core platforms of different sizes. Nevertheless, it considers homogeneous platform supporting per-core DVFS (*i.e.*, each core can have an independent v/f level). In our work, we focus on platforms derived from Arm big.LITTLE architectures [1] (*i.e.*, heterogeneous at cluster level, homogeneous at core-level, per-cluster DVFS). Firstly, we formulate the mutual influence between application-to-cluster assignments and per-cluster DVFS into an 0-1 ILP model. The 0-1 ILP formation is based on design-time prepared data, which helps to reduce complexity of the run-time calculation. To avoid issues of LEF and LPF, a dedicated global management strategy based on the proposed model is developed to determine application-to-cluster assignments and cluster v/f levels. Secondly, a local management strategy performs task-to-core mapping by selecting an appropriate design-time prepared mapping for each active application within each cluster. The local management aims to minimize resource utilization, by merging selected mappings with respect to the cluster v/f (set by the global management). The reduction of resource usage, in turn, leads to more energy-efficient solutions (*i.e.*,application-to-cluster assignments and cluster

v/f levels) in the overall system. Compared to the state-of-the-art management approaches, our work provides a new solution to deal with mapping and cluster-level DVFS. Furthermore, as we apply a hierarchical decision process and use design-time prepared data, our solution is scalable in terms of numbers of clusters and/or cores as discussed in [18].

3. System models

3.1. Application models

In this work, we consider periodic multi-task applications. The period of an application app_i is denoted as $Period_{app_i}$. As illustrated in Figure 1, an application consists of a set of H computation tasks: $T_{app_i} = \{t_{1,i}, t_{2,i}, \cdots, t_{H,i}\}$ and a set of G communication edges: $E_{app_i} = \{e_{1,i}, e_{2,i}, \cdots, e_{G,i}\}$ representing dependencies among the tasks. Tasks and edges in app_i are respectively indexed by $t_{h,i}$ and $e_{g,i}$. In the scope of this work, Synchronous Data Flow (SDF) semantic [19] is used to capture the application activity. As shown in Figure 1, input tokens define the number of tokens that are read from the edge before executing a task and the output tokens define the number of tokens that are written through the edge after executing the task.

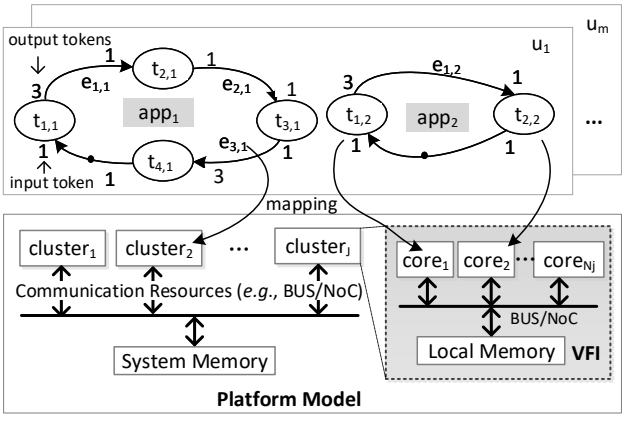


Figure 1: System model with multiple applications executed dynamically on a cluster-based multi/many-core platform.

A use-case u_m is defined as a set of simultaneously active applications according to [5]. We denote a use-case by $u_m = \{app_1, app_2, \cdots, app_I\}$, where I is the total number of active applications in the use-case. For the example in Figure 1, u_1 encompasses two active applications (*i.e.*, $u_1 = \{app_1, app_2\}$). Different use-cases might be active over time leading to different combinations of active set of applications.

3.2. Platform model

This work targets heterogeneous cluster-based multi/many-core platforms. The platform (illustrated in the bottom of Figure 1) is composed of J clusters associated with a shared memory and communication resources. Each cluster, indexed by $cluster_j$, is composed of N_j number of cores. The core type within each cluster is the same, but core types between clusters can be different (*i.e.*, defining a locally homogeneous, globally heterogeneous platform).

Besides, the considered platform supports per-cluster DVFS, making each cluster as one voltage/frequency island (VFI) [20]. All the cores within a cluster share the same voltage/frequency (v/f) level, while each cluster has its own v/f range. The available discrete frequency levels of $cluster_j$ are denoted as $\{f_{j,1}, f_{j,2}, \ldots, f_{j,max}\}$ and operating voltages are adapted to the frequency settings, as illustrated in [15], providing an adapted v/f couple.

According to the platform model, the time used to complete the computation activity of a task ($t_{h,i}$) is defined as its computation time ($CompTime_{h,i}$), while the time used to complete communication activity between dependent tasks via an edge ($e_{g,i}$) is defined as communication time ($CommTime_{g,i}$). $CompTime_{h,i}$ and $CommTime_{g,i}$ can be different due to processed data dependencies, mapping strategies and platform configurations (processing element, v/f level, $\cdots$).

The communication resources among clusters and within each cluster can be either a bus or Network-on-Chip (NoC). Traditionally busses present scalability issues when the number of cores or clusters increase [8]. From bus to NoC, a more detailed communication model can be used to capture the difference in communication time (*e.g.*, due to communication distance and possible communication congestion). As our approach considers that each application is mapped on a specific cluster (no communication among different clusters), the communication time within each cluster is initially included in its execution trace (see Section 4.1.1). For sake of simplicity, the communication power/energy is not taken into account explicitly in the experimental evaluations for all the implemented algorithms (Section 6).

3.3. Power and energy models

Energy efficiency refers to the optimization of energy consumption for the execution of the applications. Since energy consumption is the integration of the power consumption over time (*i.e.*, $Energy = Power \times Time$), we characterize energy efficiency by *the average dynamic power consumption* of active applications on platform resources over an execution period.

For a given use-case executed on a cluster-based platform, the system average power (P_{sys}^{avg}) can be expressed as the sum of the average power of all active applications in all clusters as follows.

$$P_{sys}^{avg} = \sum_{i=1}^{I} \sum_{j=1}^{J} P_{app_i}^{avg}(cluster_j, f_j) \quad (1)$$

where I and J are the total number of active applications and the total number of clusters, respectively. $P_{app_i}^{avg}$ is the average power of app_i, this value depending on the underlying type of cluster ($cluster_j$) and the cluster frequency configuration (f_j).

For an application, its $P_{app_i}^{avg}$ within a period ($Period_{app_i}$) can be computed as the amount of energy (E_{app_i}) consumed

in a unit of time as in Eq. (2). Our work assumes that the considered applications are computation intensive. As already mentioned, we do not consider the communication power/energy as in [15, 21]. The communication energy could be additionally considered in our future work, by including communications energy models as developed in existing works such as [7, 8, 22] . In our work, E_{app_i} is mainly estimated as the sum of energy consumed by all the tasks of the application.

$$P^{avg}_{app_i}(cluster_j, f_j) = \frac{E_{app_i}}{Period_{app_i}} \approx \frac{\sum_{h=1}^{H} E_{h,i}(cluster_j, f_j)}{Period_{app_i}} \quad (2)$$

where $E_{h,i}$ refers to the computation energy of $t_{h,i}$. $E_{h,i}$ can be further estimated by the integration of the power consumption ($P_{h,j}$) over its task computation time ($CompTime_{h,i}$), *i.e.*, $E_{h,i} = P_{h,i} \times CompTime_{h,i}$.

In this work as we target a big.LITTLE based architectures we reuse the power model proposed in [15]. It models the dynamic power as a third degree polynomial of frequency, as shown in Eq. (3). With $\xi_{h,i,j}$ a power coefficient that is dependent on the task ($t_{h,i}$) and the assigned core type.

$$P_{h,i}(cluster_j, f_j) = \xi_{h,i,j} \times f_j^3 \quad (3)$$

To describe the evolution of task computation time with operating frequency f_j, we use the same performance model as in [3, 23, 24] and defined in Eq. (4).

$$CompTime_{h,i}(cluster_j, f_j) \approx \frac{W_{h,i}}{f_j} \approx \frac{CompTime_{h,i}(cluster_j, f_0) \times f_0}{f_j} \quad (4)$$

where $W_{h,i}$ is the total number of execution cycles of task $t_{h,i}$, which can be understood as the amount of work that has to be done. $W_{h,i}$ can be known at a reference frequency (f_0) [15]. f_0 can be any frequency level used to evaluate (*e.g.*, by measurements) some design-time information (*e.g.*, computation time, power...).

To better evaluate the power/performance characteristics of different core types, we reuse the *performance/power ratios* defined in [2]. Let R^{perf}_j and R^{power}_j be the performance/power ratios of $t_{h,i}$ executed on $cluster_j$ respectively, while $cluster_r$ is defined as the reference cluster.

$$R^{perf}_j = \frac{CompTime_{h,i}(cluster_j, f_0)}{CompTime_{h,i}(cluster_r, f_0)} \quad (5)$$

$$R^{power}_j = \frac{P_{h,i}(cluster_j, f_0)}{P_{h,j}(cluster_r, f_0)} = \frac{\xi_{h,i,j}}{\xi_{h,j,r}} \quad (6)$$

Based on Eq. (3),(4),(5) and (6), the energy of a task $t_{h,i}$ can be further written as:

$$E_{h,i}(cluster_j, f_j) = f_j^2 \times Q_{h,i,r} \times R^{power}_j \times R^{perf}_j \quad (7)$$

where $Q_{h,i,r} = \xi_{h,i,r} \times f_0 \times CompTime_{h,i,r}(f_0)$, which is not dependent on the core type of clusters ($cluster_j$). As a consequence, the system average power P^{avg}_{sys} (Eq. (1) can be further written into Eq. (8).

$$P^{avg}_{sys} = \sum_{i=1}^{I} \sum_{j=1}^{J} f_j^2 \times \frac{\sum_{h=1}^{H} Q_{h,i,r} \times R^{power}_j \times R^{perf}_j}{Period_{app_i}} \quad (8)$$

This equation will be used latter in the optimization problem.

It has to be noticed that we do not consider Dynamic Power Management (DPM) strategy [4], and this would be a part of our future work. Thus, DPM approaches like shutting down non-used clusters/cores to reduce *static power/energy* is beyond the scope of this work.

4. Management organization and problem definition

The main objective of this work is to optimize the average dynamic power consumption of the overall system, where multiple applications are executed concurrently in a particular use-case. This objective is achieved by run-time hierarchical management that appropriately determines application mappings (*i.e.*, inter and intra cluster) and dynamically sets cluster frequencies (the voltage being dependent on the selected frequency). Figure 2 shows the overview of the proposed approach.

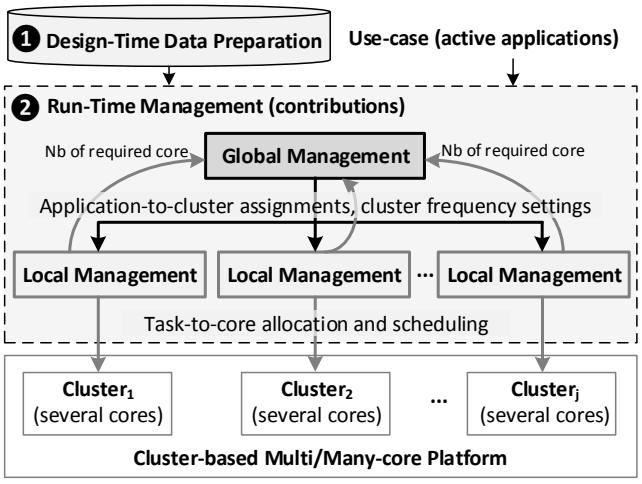


Figure 2: Run-time hierarchical management for several applications executed dynamically on a cluster-based heterogeneous multi/many-core platform.

The run-time management is based on *design-time data preparation* (see ❶ in Fig. 2), these design-time prepared

data are presented in Subsection 4.1. These design-time analysis helps to reduce run-time computation burdens and still guarantees mapping quality [21]. Moreover, the run-time management is considered in a hierarchical organization (see ❷). For each use-case, the global management determines the *application-to-cluster assignments* and sets accordingly the *cluster frequency levels*. The global management considers that each application is assigned to one cluster (no mapping at task-level) to avoid costly communications among clusters. For the assigned applications in each cluster, the local management determines the *task-to-core allocation and scheduling* without changing the cluster frequency set by the global management. As already stated, the hierarchical approach and the use of design-time prepared data support the scalability of the approach.

4.1. Design-time data preparation

At design-time, we prepare a set of different mappings for each application. The prepared mappings are evaluated (computing time and communication time as a whole) in a reference cluster ($cluster_r$) at a reference frequency (f_0). Besides, for each prepared mapping, we also evaluate the minimum required frequency (defined as MAF) to meet the application timing constraints.

4.1.1. Mappings preparation

We consider that multiple mappings (*i.e.*, having different resource usage and performance trade-off) are prepared for each supported application. These mappings establish the usage (*i.e.*, allocation and scheduling) of cores for the different tasks of the applications. Each mapping is characterized by its execution trace. An execution trace is a set of instants defining the start time (x_s) and end time (x_e) of each task executed at a given frequency on the targeted cluster within a period. The execution trace of an application app_i mapped on c cores is defined by $X^c_{app_i} = \{x_{s\_t_{h,i}}(1), x_{e\_t_{h,i}}(1), \cdots x_{s\_t_{h,i}}(k), x_{e\_t_{h,i}}(k)\}$, where k refers to the k^{th} instance of a given task. Figure 3 illustrates two possible mappings of app_1 (see Fig. 1), using (a) one core and (b) two cores.

Different mappings lead to different application latencies ($Latency_{app_i}$ refers to the latency of app_i). It can be observed from Figure 3 that the latency ($Latency_{app_1}$) of $X^2_{app_1}$ is smaller than $X^1_{app_1}$ due to execution parallelism. As a result, the slack time of the two prepared execution traces are different with respect to $Period_{app_1}$. The slack time can be used to scale down the cluster frequency while still meeting the considered timing constraint.

At run-time, the global management uses only the mapping that leads to the minimum latency (*i.e.*, that uses the maximum number of cores) among all the prepared mappings. We particularly denote this mapping $X^{c_i}_{app_i}$, which uses c_i number of cores. The global management uses the information of $X^{c_i}_{app_i}$ to achieve the lowest frequency configurations of clusters (discussed in Section 5.2). On the other hand, the local management can access all prepared mappings of each application, including $X^{c_i}_{app_i}$ and other

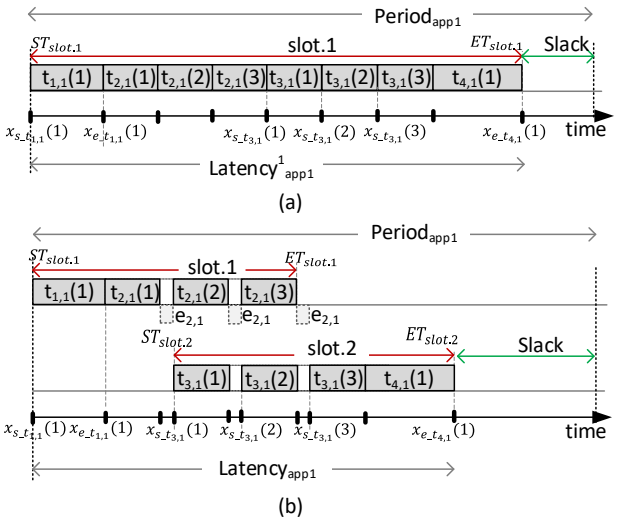


Figure 3: Design-time prepared mapping for app_1 running (a) on one core ($X^1_{app_1}$) and (b) on two cores ($X^2_{app_1}$).

$X^c_{app_i}$. The local management then uses the information of all prepared mappings to minimize resource utilization (discussed in Section 5.3).

4.1.2. Evaluation of the Minimum Allowed Frequency (MAF)

We define as the allowed frequencies, for a given mapping, the set of frequencies that allow the application to respect its timing constraints. The allowed frequencies can be obtained by experimental measurements or by application performance models (see Eq.(4)). In our approach, only the Minimum Allowed Frequency (MAF), of each prepared mapping is stored within design-time prepared data. We define $MAF^c_{i,j}$ as the MAF of app_i mapped on c cores in $cluster_j$. Different mappings of an application can lead to different MAFs with respect to the same timing constraint. For the example in Figure 3, $MAF^2_{1,j}$ can be smaller than $MAF^1_{1,j}$ because the longer slack time of $X^2_{app_1}$ allows execution at a lower frequency level without timing violation.

$MAF^c_{i,j}$ can be deduced approximately based on $MAF^c_{i,r}$ according to application performance model (*i.e.*, $MAF^c_{i,j} = MAF^c_{i,r} \times R_j^{perf}$, from Eq.(5)). Such an estimation is accurate for platforms that support continuous frequencies scaling. However, this estimation can be a little biased in our considered platforms that support discrete frequency levels. As the current platform has a limited number of clusters, we evaluate $MAF^c_{i,j}$ for every different cluster at design-time to overcome this issue.

As previously discussed, only the information of $X^{c_i}_{app_i}$, having the minimum MAF (*e.g.*, $MAF^{c_i}_{i,j}$) among all is used by the global management. At run-time, $MAF^{c_i}_{i,j}$ will be used by the global management to set cluster frequencies, while the others $MAF^c_{i,j}$ will be used by the local management to explore task-to-core mapping to minimize resource usage. The local resource optimization is performed at the

cluster frequency set by the global management.

At this point, the global management reduces the exploration space by withdrawing some prepared mappings during decision-makings. Besides, the local management reduces the exploration space by excluding cluster v/f in its optimization objective.

4.2. 0-1 ILP formulation of the hierarchical run-time management

The hierarchical run-time management problem is formulated as a 0-1 ILP model. In the formulation, the *application-to-cluster assignment* decision is expressed as *variables*. When the assignment variables change at run-time, the *cluster frequency configuration* is then set to respect **(1)** application timing and **(2)** cluster frequency constraints. The local management decision *task-to-core mapping* is performed afterwards under **(3)** cluster resource constraints.

• *Application-to-cluster assignment*: For a set of simultaneously active applications (in each use-case), we define a matrix *variables* $[a_{i,j}]_{I\times J}$, where $a_{i,j}$ is a binary variable equal to:

$$a_{i,j} = \begin{cases} 1, & \text{if } app_i \text{ is assigned onto } cluster_j. \\ 0, & \text{otherwise.} \end{cases} \quad (9)$$

• *Cluster frequency configurations*: The global management aims to reduce the cluster frequencies as much as possible, using the design-time prepared $MAF_{i,j}^{c_i}$. To meet **(1)** application timing constraints in a cluster ($cluster_j$), the optimized cluster frequency (f_j) should not be lower than the highest $MAF_{i,j}^{c_i}$ value of all $cluster_j$ assigned applications.

$$f_j >= \max\{MAF_{i,j}^{c_i} \times a_{i,j}\}, \forall i \quad (10)$$

Besides, to satisfy the **(2)** cluster frequency constraints, the optimized frequency of each cluster should be within the frequency range of the cluster.

$$f_j \in \{f_{j,1}, f_{j,2}, \ldots, f_{j,max}\}, \forall j \quad (11)$$

• *Task-to-core mapping:* The local management aims to reduce the number of used cores in the cluster under **(3)** cluster resource constraints. In each cluster, the number of used cores (N_j^{used}) by the assigned applications should not be larger than the available cores (N_j). N_j^{used} depends on the application-to-cluster assignment matrix variables, and on the applied local management strategy. This latter establishes task-to-core mapping under the guidance of all prepared mappings $X_{app_i}^c$ and $MAF_{i,j}^c$.

$$N_j >= N_j^{used}([a_{i,j}]_{I\times J}, \textit{Local Management Strategy}) \quad (12)$$

The *objective* of the hierarchical management is to minimize the average power consumption of the system based on Eq.(8). The objective is expressed as Eq.(13), which is subject to Eq.(10), Eq.(11) and Eq.(12).

$$\begin{aligned} \min P_{sys}^{avg} &= \min \sum_{i=1}^{I} P_{app_i}^{avg} \\ &= \min_{\{f_j, a_{i,j}\}} \sum_{j=1}^{J} f_j^2 \sum_{i=1}^{I} a_{i,j} \frac{\sum_{h=1}^{H} Q_{h,i,r} \times R_j^{power} \times R_j^{perf}}{Period_{app_i}} \end{aligned} \quad (13)$$

For a particular situation where an application (app_i) requires to be assigned to one *empty* cluster among several candidates, the average dynamic power of the application ($P_{app_i}^{avg}$) Eq.(13) can be rewritten as in Eq.(14). Note that $P_{app_i}^{avg}$ is the average power of app_i after frequency reduction under the timing constraint, and the cluster frequency (f_j) is determined by $MAF_{i,j}^{c_j}$.

$$\begin{aligned} P_{app_i}^{avg} &= (MAF_{i,j}^{c_i})^2 \times \frac{\sum_{h=1}^{H} Q_{h,i,r} \times R_j^{power} \times R_j^{perf}}{Period_{app_i}} \\ &= (MAF_{i,r}^{c_i})^2 \, (R_j^{perf})^2 \, \frac{\sum_{h=1}^{H} Q_{h,i,r} \times R_j^{power} \times R_j^{perf}}{Period_{app_i}} \end{aligned} \quad (14)$$

In Eq.(14), $MAF_{i,j}^{c_j}$ can be further approximately expressed as $MAF_{i,r}^{c} \times R_j^{perf}$ (Eq.(5)). Additionally, let define $EF_j = R_j^{perf} \times R_j^{perf} \times R_j^{perf} \times R_j^{power}$, which refers to *energy factor* of $cluster_j$, EF_j only depends on the core type of the cluster. $P_{app_i}^{avg}$ becomes:

$$P_{app_i}^{avg} = EF_j \times (MAF_{i,r}^{c_i})^2 \times \frac{\sum_{h=1}^{H} Q_{h,i,r}}{Period_{app_i}} \quad (15)$$

The minimum $P_{app_i}^{avg}$ (*i.e.*, P_{sys}^{avg} for one application) can be achieved in the cluster with the smallest EF_j. This *property* will be used to reduce exploration space of the proposed global management strategy in Section 5.2.

5. Proposed run-time global and local management strategies

5.1. General architecture

Solving the above 0-1 ILP optimization problem is too computationally expensive to be done at run-time. In this work, we aim to achieve optimized solutions through heuristic strategies in a reasonable time. The overview of the proposed run-time global and local strategies (in hierarchical management) is shown in Figure 4.

Global management considers the application-to-cluster assignment of active applications one after another. For the

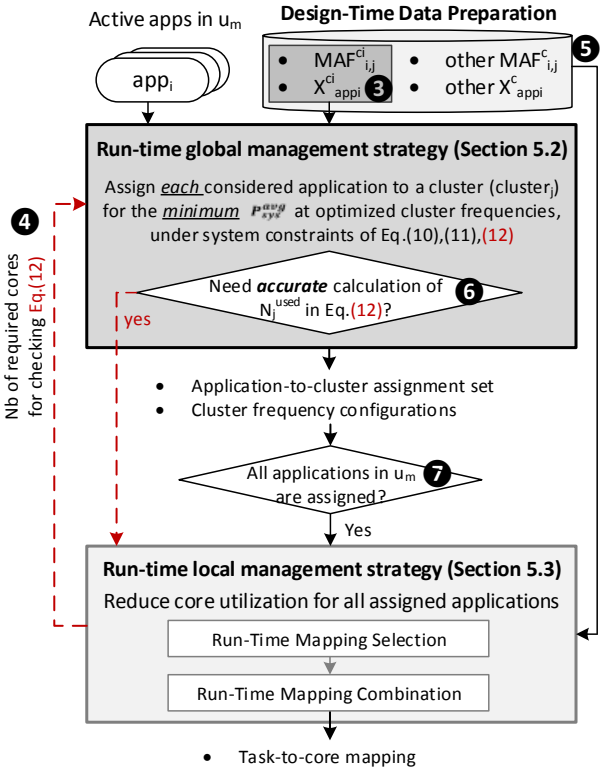


Figure 4: The overview of the proposed run-time hierarchical management: a global strategy and a local strategy.

current application, a *greedy* strategy is used to estimate optimized cluster f_j levels (Eq.(10), Eq.(11)) as well as P_{sys}^{avg} (Eq.(13)) of different assignment *attempts*. It then assigns the application to the cluster with the minimum P_{sys}^{avg} under the system constraints. Application timing and cluster frequency constraints are guaranteed by the selected f_j levels based on design-time prepared $MAF_{i,j}^{c_i}$ (see ❸ in Figure 4). The frequency of each cluster is determined by the assigned application, which has the largest $MAF_{i,j}^{c_i}$ in the same cluster (see Eq.(10)). As a consequence, other applications in the same cluster might work at a higher frequency than their own prepared MAFs. The *greedy* global strategy intends to assign the active applications with close $MAF_{i,j}^{c_i}$ to the same cluster. This helps to keep the frequency increase of applications in the same cluster as small as possible. Nevertheless, cluster resource constraints cannot be guaranteed as N_j^{used} (Eq.(12)) cannot be known unless the local management strategy is executed. It means that global management requires the estimation of the parameter N_j^{used} from local management to justify whether cluster resource constraints are met for different assignment *attempts* (see ❹ in Figure 4).

In the local management, a heuristic strategy is used to determine task-to-core mapping for the assigned applications in each cluster, at the frequency (f_j) set by the global management. The local strategy aims to minimize the number of used cores in each cluster, using all design-time prepared data (*i.e.*, all $X_{app_i}^c$ and $MAF_{i,j}^c$, ❺ in Figure 4). Fewer resource usage within each cluster allows the global management to assign more applications to more energy-efficient clusters, and consequently results in more energy efficiency in the overall system. Resource minimization in the local management is achieved through run-time *mapping selection* and *combination* steps. The first step selects one of the prepared mappings for each assigned application according to f_j, under the $MAF_{i,j}^c$ constraint of the applications. The second step combines the selected mappings together. This process is detailed in Section 5.3. Note that at different f_j, the selected mapping for each assigned application as well as their combined mapping can be different, which makes the N_j^{used} parameter available only after local management execution.

However, computing N_j^{used} in local management and then feeding back to global management in every application assignment *attempts* will increase computational overhead. To reduce this overhead a *pessimistic estimation* and an *accurate calculation* of N_j^{used} can be used.

Pessimistic estimation: We define the pessimistic estimation of N_j^{used} at the global management level, as $N_j^{used,max}$, which refers to the maximum number of used cores for the assigned applications in each cluster. That is $N_j^{used,max} = \sum_{i=1}^{I} c_i \times a_{i,j}$ by assuming that each application uses its maximum number of cores (c_i). When the total number of available cores in a cluster (N_j) is large enough (*i.e.*, $N_j >= N_j^{used,max}$), global management can ensure that the resource constraints (Eq.(12)) are met in application assignment *attempts*.

Accurate calculation: When $N_j < N_j^{used,max}$, the pessimistic estimation cannot ensure that the resource constraint can be achieved. In this case, the accurate N_j^{used} feedback from the local management is required. The accurate calculation (see ❻ in Figure 4) is performed only when $0 < N_j^{used,max} - N_j <= Secure\ Margin$. *Secure Margin* is an integer value to reduce unpromising explorations when $N_j^{used,max}$ is too high compared to N_j. In our experiments, *Secure Margin* is arbitrarily set to 10. This value can be adapted according to the performance of the considered local management strategy and to the available cores (N_j) in each cluster.

Once global management is completed for all active applications (❼ in Figure 4), the local strategy is executed to definitely map tasks to cores in each cluster at the determined cluster frequency (f_j).

5.2. Global management strategy

When a new use-case starts, the proposed global management strategy called *Greedy Search Application-to-Cluster Assignment (GSACA)* is executed. The GSACA strategy first assigns newly active applications (in the current use-case) to clusters, and then allows migration of old existing applications (from the previous use-case) from one cluster to another. If there is no new active application (only applications resuming), only application migration is performed.

The Algorithm 1 presents the GSACA strategy, which consists of three basic steps: *new application(s) ordering,*

Algorithm 1: Greedy Search Application-to-Cluster Assignment (GSACA) Strategy

Input: design-time prepared data, active applications in the current use-case, application timing constraints, cluster-based multi/many-core platform
Output: application-to-cluster assignments, cluster frequency configurations

1 //Step 1: New application ordering
2 Average I newly active applications into J groups, in decreasing order of $MAF_{i,r}^{c_i}$;
3 *Order* newly active applications in a particular way;
4 //Step 2: New application assignment
5 **for** *each new* app_i *or released old* app_i **do**
6 //Step 2.a: select some clusters satisfying Eq.(10),(11)
7 Select *used or empty* clusters that meet $f_{j,max} >= MAF_{i,j}^{c_i}$;
8 //Step 2.b: solution attempts by updating $a_{i,p}$,f_p and P_{sys}^{avg}
9 **for** *each selected cluster* $cluster_p$ **do**
10 Estimate cluster frequency f_p (Eq.(10)) and P_{sys}^{avg} (Eq.(13));
11 **if** *the first feasible* $cluster_p$ *or* P_{sys}^{avg} *decreases* **then**
12 //Step 2.c: check Eq.(12) by *pessimistic estimation* or *accurate calculation*
13 **if** $0 < \sum_{i=1}^{I} c_i \times a_{i,p} - N_p <= 10$ **then**
14 *Accurately* calculates N_p^{used} by executing local strategy
15 **end**
16 **if** $\sum_{i=1}^{I} c_i \times a_{i,j} >= N_p$ *or* $N_p >= N_p^{used}$ **then**
17 Assign app_i to $cluster_p$;
18 Update frequency level of $cluster_p$;
19 **end**
20 **end**
21 **end**
22 **if** *No solution can be found for new* app_i **then**
23 Release an already assigned new application that has lower $MAF_{i,r}^{c_i}$ and can provide available cores for the currently considered new application;
24 *Otherwise*, application assignment fails;
25 **end**
26 **end**
27 //Step 3: Old application migration
28 Initialize further allowed migration $Nm = 0$;
29 **if** $Nm <= Allow_{migration}$ **then**
30 Assign the old unmigrated application with the minimum $MAF_{i,r}$ (line 6-18);
31 $Nm++$;
32 **end**

new application(s) assignment, old existing application(s) migration. Due to possible migration costs, the GSACA allows controlling the number of migrations ($Allow_{migration}$ in Algorithm 1).

5.2.1. New application(s) ordering

In the first step, newly active applications are ordered (line 1-3 in Algorithm 1), according to design-time prepared $MAF_{i,r}^{c_i}$ on the reference cluster ($cluster_r$). The application ordering aims to avoid assigning applications with *close* $MAF_{i,r}^{c_i}$ successively. As previously illustrated in Eq.(11), each cluster frequency level is chosen by the assigned application with the maximum $MAF_{i,r}^{c_i}$. In order to achieve the minimum P_{sys}^{avg}, an application could be assigned to an empty cluster to avoid its frequency being increased by an other application already assigned in the candidate cluster. When all empty clusters are taken up by former considered applications (with *close* $MAF_{i,r}^{c_i}$ values), it means that all cluster will have close f_j. Then latter considered applications can also be assigned to different clusters due to resource constraints. Separating applications with *close* and *high* $MAF_{i,r}^{c_i}$ into different clusters would result in different clusters with high frequency levels and thus may globally increase P_{sys}^{avg} at the end.

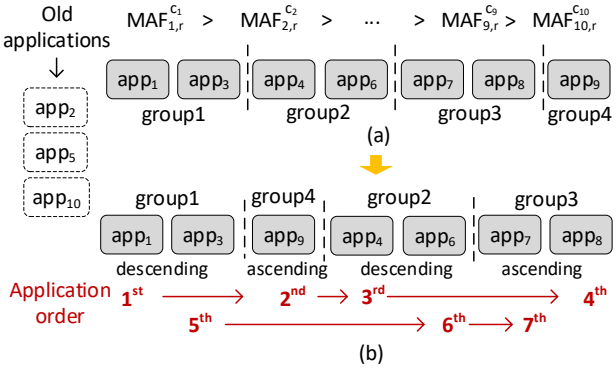


Figure 5: GSACA order result for 7 newly active applications in 4 groups.

Our particular ordering approach aims to avoid assigning applications with *close* $MAF_{i,r}^{c_i}$ successively. For this purpose, the GSACA strategy divides newly active applications into J (*i.e.*, the number of clusters) groups according to their descending order of $MAF_{i,r}^{c_i}$.

In the example of Figure 5 (a), 10 applications are active in the considered use-case. The 7 newly active applications (in gray in the figure) are sorted according to the descending order of their respective $MAF_{i,r}^{c_i}$. In this example, app_1 and app_3 have relatively *close* $MAF_{i,r}^{c_i}$ values. In Figure 5 (a), the 7 newly active applications are divided into 4 groups (the considered platform having 4 clusters). To give applications in each group an opportunity to occupy an empty cluster, GSACA successively indexes an application in each groups. In the meanwhile, we expect that the successively indexed applications have *distant* $MAF_{i,r}^{c_i}$. To this end, we first re-order application groups, which refer to group1-group4-group2-group3 (see Figure 5 (b)). Then, in each group, applications are indexed (from 1^{st} to 7^{th}) according to descending order and ascending order of $MAF_{i,r}^{c_i}$ alternatively.

5.2.2. New application(s) assignment

Following the obtained application order, the second step of the approach is to assign each newly active application to a cluster that leads to the minimum P_{sys}^{avg}. For each application-to-cluster assignment, system constraints (*i.e.*, Eq.(10),(11),(12)) are checked.

The GSACA strategy first selects potential clusters (used or empty) under the application timing constraint and cluster frequency constraint (*i.e.*, $f_{j,max} >= MAF_{i,j}^{c_i}$) (step 2.a in Algorithm 1). If several empty clusters meet the constraints, only the cluster with the minimum EF_j (see Section 4.2) is selected for further evaluation. This helps to reduce the number of selected clusters. Figure 6.(a) gives an example of the selection of clusters for application app_1. In this example, the old existing applications (app_2, app_5, app_{10}) are kept in the same clusters as in previous use-case. The GSACA strategy searches for possible clusters for app_1. As

$cluster_2$ cannot guarantee the application timing constraint (as $f_{2,max} < MAF^{c_1}_{1,2}$), the remaining used cluster ($cluster_1$) and the empty cluster ($cluster_3$) with the lowest energy factor EF_3 are selected for further evaluations.

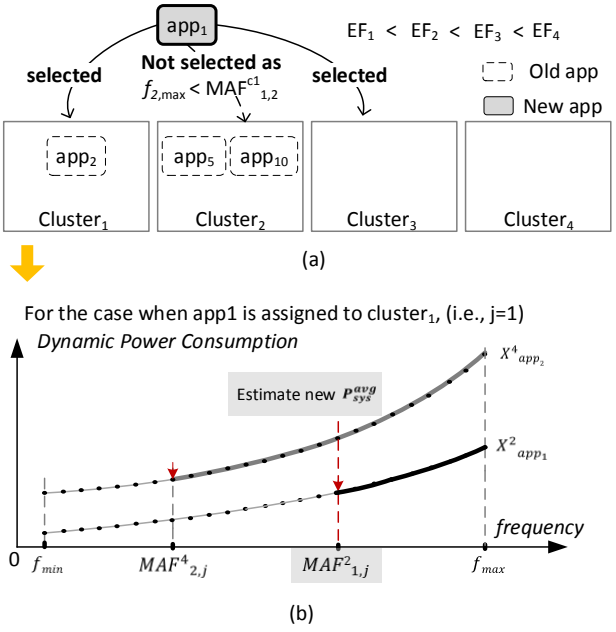


Figure 6: (a) Selection of possible clusters for app_1; (b) estimation of the optimized cluster frequency when app_1 is assigned to $cluster_1$.

For each possible selected cluster, GSACA estimates the optimized cluster frequency (f_j) and the overall P^{avg}_{sys} at this frequency (step 2.b in Algorithm 1). The optimized cluster frequency is determined by all the $MAF^{c_i}_{i,j}$ of the applications assigned to the cluster (Eq.(11)). Figure 6 (b) illustrates how the cluster frequency is updated if app_1 is assigned to $cluster_1$. One can see the dynamic power consumption evolution of the considered prepared mappings $X^{c_i}_{app_i}$(*i.e.*, $X^2_{app_1}$ and $X^4_{app_2}$) for the active applications (app_1 and app_2) in $cluster_1$. The allowed frequency range of each application is highlighted (in bold line). In this case, the global management sets the cluster frequency to the minimum common frequency that supports the execution of both applications, which corresponds to the maximum MAF ($MAF^2_{1,j}$ in the example). Given the cluster frequency, P^{avg}_{sys} is estimated using Eq.(13).

The estimated P^{avg}_{sys} for all selected clusters ($cluster_1$ and $cluster_3$ in Figure 6) are compared. For the first evaluated cluster (or when P^{avg}_{sys} decreases compared to the assignment in previous evaluated cluster), GSACA checks whether the resource constraints (Eq.(12)) are met using the *pessimistic* or the *accurate* estimation of N^{used}_j (step 2.c in Algorithm 1). By doing this for all selected clusters, GSACA can explore the assignment of each application to find the minimum P^{avg}_{sys} under current system constraints.

However, it is possible that no cluster can be found for a new application, if the clusters that meet the application timing constraints are already used (line 25-28). In this case, GSACA checks already assigned new applications that have lower $MAF^{c_i}_{i,r}$, in order to release some cores. The released application will then be reassigned to another cluster by greedy searching (step 2 of Algo.1). If no application can be released, the application assignment fails.

5.2.3. *Old existing application(s) migration*

After all new applications are assigned to clusters, the third stage is to further optimize P^{avg}_{sys} by migrating old existing applications from one cluster to another (line 23-29 in Algorithm 1). The number of allowed migrations ($Allow_{migration}$ in line 29) can be set by the user. The GSACA strategy migrates any old application from one cluster to another one by greedy searching for a new assignment solution. The new solution should reduce P^{avg}_{sys} as much as possible at the new optimized cluster frequencies (f_j in Eq.(11)). In this work, we choose to migrate the old application with the minimum $MAF^{c_i}_{i,r}$ first. This is because the frequencies of old applications with low $MAF^{c_i}_{i,r}$ can be highly increased by newly active applications assigned to the same cluster. This could happen when resources are not sufficient for the newly active applications. Notice that migrations could reduce P^{avg}_{sys} at the expense of migration overhead. For the sake of simplicity, this work does not model migration overheads and considers only the number of allowed migrations. The migration overheads can be neglected if tasks are running for a sufficiently long time [25]. Besides, we assume that migrations are performed at the end of application periods to avoid migrating intermediate task states.

It has to be noticed that at the end of this process, all empty clusters (or even unused processors) could be switched off to reduce static power consumption. This point has not been evaluated in this work, and will be taken into account in our future works (*i.e.*,for DPM consideration).

5.3. Local management strategy

The local management strategy is executed in each cluster under two situations. First, when global management requests an *accurate* calculation of the required cores (N^{used}_j). Second, when the application-to-cluster assignment in a use-case is completed, the local strategy updates the task-to-core mapping in the cluster. As previously discussed in Section 5.1, the proposed local management strategy is based on all design-time prepared data (*i.e.*, $X^c_{app_i}$ and MAF^c_i). It aims to minimize core utilization of task-to-core mapping by performing the *run-time mapping selection and combination* steps. The two steps are performed for all applications in the cluster.

5.3.1. *Run-time mapping selection*

The run-time mapping selection step selects the mapping for each application based on their MAFs and the cluster frequency (f_j) set by the global management. The selection aims to reduce the number of used cores for all assigned applications in the cluster.

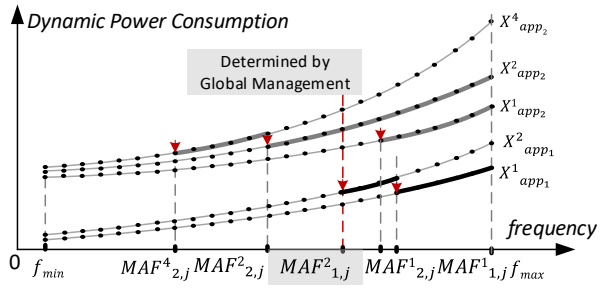


Figure 7: MAF-based mapping selection for active applications app_1 and app_2 at the global management determined frequency $f_j = MAF^2_1$.

In the example of Figure 7, app_1 and app_2 are assigned to the same cluster and cluster frequency ($f_j = MAF^2_{1,j}$) is selected by the global management decision (see Fig. 6 (b)). Figure 7 shows the dynamic power consumption evolution of all prepared mappings for the two active applications in the cluster. The figure highlights the selected design-time mapping (bold line) for each application at different frequencies. At the cluster frequency $f_j = MAF^2_1$, the selected mappings are $X^2_{app_1}$ and $X^2_{app_2}$ (see the red dotted line). Compared to the considered mappings ($X^2_{app_1}$ and $X^4_{app_2}$) in the global management, the local management reduces 2 cores of the required resources. For more active applications with more prepared mappings, the total number of reduced cores can be more significant.

5.3.2. *Run-time mapping combination*

After the mappings selection, the selected mappings are combined at the optimized cluster frequency (f_j). The combined mapping must respect the timing constraints of all assigned applications. This work considers two different mapping combination strategies, including the commonly used *First-Come-First-Serve* (FCFS) [26] strategy, and our previous proposed heuristic strategy called *Grouped Applications Packing under Varied Constraints* (GAPVC) [27]. As introduced in [27], these two strategies perform task-to-core mapping without degrading application performance. Since FCFS allows each core to be used only by one application, some processing resources can be wasted. To reduce resource usage, GAPVC can allocate tasks of different applications to the same core. Figure 8 compares the mappings results of FCFS and GAPVC for app_1 and app_2. The combination is performed within the Least Common Multiple (LCM) of application periods.

Figure 8 (a) abstract the design-time prepared execution trace $X^2_{app_1}$ (from Figure 3 (b)) at the slot level. Slots are formed by combining multiple adjacent executions of tasks on the same core (see *slot*.1 and *slot*.2). This slot-level execution trace is then extended at the cluster frequency ($f_j = MAF^2_1$) within the LCM of periods of the active applications. The same slots executed in different periods are defined as periodic (*i.e.*, *slot*.1 and *slot*.5 in Figure 8).

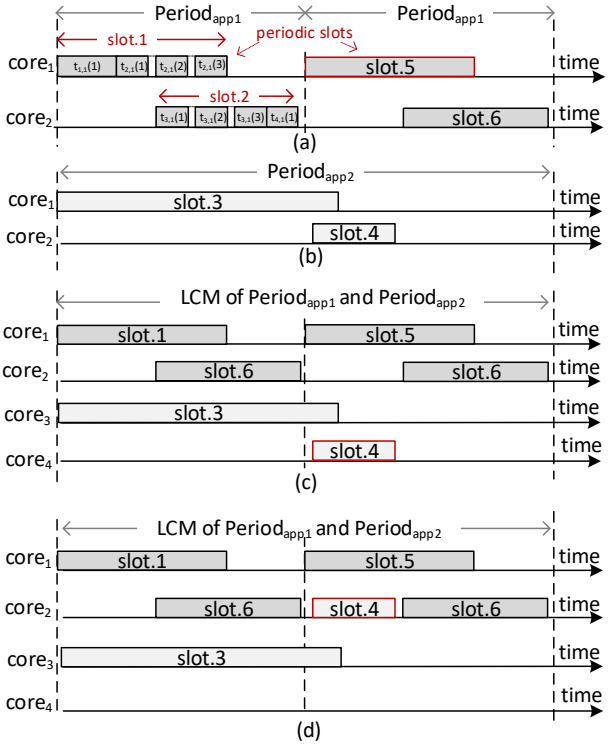


Figure 8: Mappings combination of app_1 and app_2: (a) $X^2_{app_1}$ and (b) $X^2_{app_2}$ extended at the cluster frequency ($f_j = MAF^2_1$) within the least common multiple (LCM) of periods; The combined mappings on four cores using (c) FCFS and (d) GAPVC.

The periodic slots are normally packed onto the same core, in order to reduce the migration overhead. Figure 8 (b) shows the slot-level execution trace $X^2_{app_2}$. Notice that GAPVC provides a higher packing priority for slots with earlier end times (*e.g.*, see the slot order in part (a) and (b)), to resolve communication contention between concurrent tasks as in [7, 28]. Part (c) and (d) of Figure 8 show the combined mappings using FCFS and GAPVC respectively. As it can be seen, GAPVC uses fewer cores than FCFS to combine the selected mappings together. This is because GAPVC allows slots of different applications (*e.g.*, *slot*.4 and *slot*.6 in part (d)) to be mapped onto the same core. More details about the GAPVC strategy can be found in [27].

6. Experimental evaluations

6.1. Experiment settings

For experimental evaluations, all management strategies are implemented in C++. To go beyond the existing small-sized heterogeneous cluster-based platforms (*i.e.*, ARM big.LITTLE, Exynos 5422 [1]), we consider 4 core types to compose different cluster-based platforms. We set up the platforms based on [2], which focuses on big.LITTLE architectures of different sizes using ARM Cortex-A series processors (*e.g.*, ARM-Cortex-A9, A15, A7, A17). The physical characteristics of the 4 core types [2] are reported in Table 2. The values of performance ratio (R^{perf}_j in Eq.(5))

and power ratio (R_j^{power} in Eq.(6)) are normalized to Cortex-A9 (gray in Table 2). The frequency of the core types can be differently set with a $f_{step} = 0.1GHz$ increment. Based on these 4 core types, up-to 8 clusters are considered. In order to evaluate the scalability of the different management strategies, the experimental evaluations are performed for different platform sizes (*i.e.*, number of clusters × the number of cores inside each cluster). We considered 2-cluster platforms ($cluster_1$, $cluster_2$), 4-cluster platforms ($cluster_1$ to $cluster_4$), 6-cluster platforms ($cluster_1$ to $cluster_6$) and 8-cluster platforms ($cluster_1$ to $cluster_8$), where each cluster can encompass 4, 8, 16 or 24 cores.

Table 2: Characteristics of the 4 considered core types, and performance ratio values from [2]

Cortex-	R^{perf}	R^{power}	f_{min} (GHz)	f_{max} (GHz)	f_{step} (GHz)	cluster
A7	1.25	0.55	0.4	1.4	0.1	$cluster_3, cluster_7$
A9	1	1	0.4	2.0	0.1	$cluster_1, cluster_5$
A15	0.625	2.25	0.4	2.0	0.1	$cluster_2, cluster_6$
A17	0.645	1.3	0.4	1.8	0.1	$cluster_4, cluster_8$

The 10 considered applications (app_1 to app_{10}) are defined in Table 3. They are derived from reference applications (H263 encoder and H263 and JPEG decoders) with different input/output tokens. Each application was captured as an SDF model [29]. As can be seen in Table 3, we prepared for each application several mappings with different numbers of cores providing different MAF values. MAF values are evaluated regarding the used core types (Cortex-A9, -A15, -A7, -A17) in the reference cluster. We assume that the worst-case computation time of each task provided by SDF3 [29] is derived from the Cortex-A9 cluster at $f_{max} = 2.0GHz$. The computation time and MAF values of the different mappings (*e.g.*, corresponding to different core types and different f_j configurations) are obtained according to the performance ratio R_j^{perf} (see Eq.(4),(5) in Section 3). In Table 3, the design-time prepared data used at the global management level are highlighted (in gray), and all prepared data can be accessed at the local level.

The experimental evaluations compare our proposed run-time strategies (*i.e.*, global and local) to state-of-the-art strategies. The comparisons are performed according to their achieved P_{sys}^{avg} and strategy complexities. In the realistic assumption that each use-case runs for a sufficient long time [25], the impact of application migration power/energy (or run-time exploration power/energy) on P_{sys}^{avg} can be neglected. We particularly evaluate the exploration time of different run-time strategies, based on their average time to explore a run-time configuration. The comparison of exploration time reflects the computation complexity of different strategies.

6.2. Evaluations of Global Management Strategies

In this section, we compare the proposed GSACA strategy to three state-of-the-art global management strategies. We considered first an *Exhaustive* strategy. It evaluates all possible application-to-cluster assignments and provides the optimal solution (the lowest P_{sys}^{avg} at the optimized cluster frequencies). The *LPF* [9] strategy assign all active applications to the clusters with the lowest power consumption (*i.e.*, Cortex-A7 due to the minimum power ratio R_j^{power}). While *LEF* [3] assign the application first to the cluster with the lowest energy consumption (*i.e.*, Cortex-A17 due to the minimum energy factor EF_j). In *LPF* and *LEF*, when the cluster with the lowest power/energy consumption is not available for an application, the cluster with the next lowest power/energy consumption is considered. For all these experiments, the local management uses the FCFS strategy [26] based on a single prepared mapping ($X_{app_i}^{c_i}$), denoted as $FCFS_S$. This ensures that the comparison considers only the achievements of global management. Global management strategies are compared in two cases: a *special case* where the application assignments take place on an always empty platform (in any use-case), and the *general case* which takes into account already mapped applications from the previous use-case.

6.2.1. Global management solution (special case)

The *special* case assumes that each use-case starts on an empty platform, making the assignment of each use-case independent from the others. The global management considers the assignments of all active applications in each use-case.

Figure 9 shows the compared P_{sys}^{avg} achieved by the global management strategies for this special case. The comparison is performed for all the possible (*i.e.*,1023) use-cases of the 10 considered applications, and for different platform sizes. The P_{sys}^{avg} of each global management strategy is normalized to the *Exhaustive* result under the constraints of the *8 clusters* × *8 cores* platform. The values in Figure 9 refer to the average normalized P_{sys}^{avg} of the 1023 use-cases.

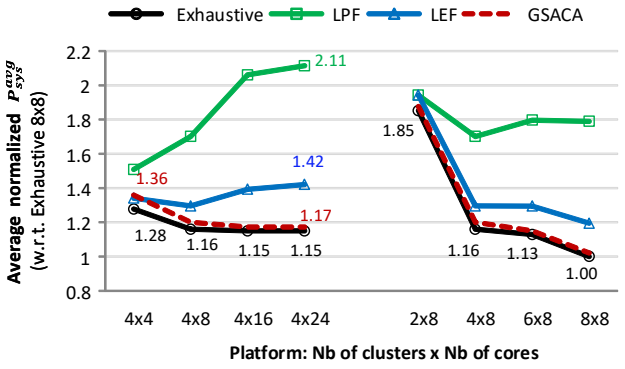


Figure 9: Average normalized P_{sys}^{avg} with respect to Exhaustive strategy (constrained by 8×8 platform size) for the 1023 use-cases. Results are given for different platforms constraints.

The *Exhaustive* results show that the optimal P_{sys}^{avg} is highly dependent on platform constraints. With more cores per cluster in heterogeneous platforms, reduced P_{sys}^{avg} can

Table 3
Design-time prepared data of the 10 considered applications

Application				Prepared Mappings		
Type	app_i	Nb of tokens of each task	Period (μs)	$X^1_{app_i}$ Nb of cores: MAFs[1] (GHz)	$X^2_{app_i}$ Nb of cores: MAFs (GHz)	$X^4_{app_i}$ Nb of cores: MAFs (GHz)
H263 decoder :4 tasks :3 edges	app_1	{1, 6, 6, 1}	60	1: {1.0, 0.7, 1.3, 0.7}	2: {1.0, 0.6, 1.2, 0.7}	
	app_2	{1, 4, 4, 1}	180	1: {0.4, 0.3, 0.5, 0.3}	2: {0.4, 0.2, 0.4, 0.2}	
	app_3	{1, 264, 264, 1}	360	1: {1.3, 0.8, 1.6, 0.8}	2: {0.8, 0.5, 1.0, 0.5}	
H263 encoder :5 tasks :4 edges	app_4	{1, 5, 5, 5, 1}	540	1: {1.3, 0.8, 1.6, 0.9}	2: {1.2, 0.8, 1.5, 0.8}	
	app_5	{1, 15, 15, 15, 1}	1080	1: {0.9, 0.6, 1.1, 0.6}	2: {0.7, 0.5, 0.9, 0.5}	
	app_6	{1, 45, 45, 45, 1}	1080	1: {1.5, 0.9, 1.8, 1.0}	2: {1.1, 0.7, 1.3, 0.7}	
JPEG decoder :6 tasks :5 edges	app_7	{1, 7, 7, 7, 7, 1}	180	1: {1.3, 0.9, 1.7, 0.9}	2: {1.1, 0.7, 1.4, 0.7}	4: {1.0, 0.7, 1.3, 0.7}
	app_8	{1, 9, 9, 9, 9, 1}	360	1: {0.8, 0.5, 1.0, 0.5}	2: {0.7, 0.4, 0.8, 0.4}	4: {0.6, 0.4, 0.7, 0.4}
	app_9	{1, 22, 22, 22, 22, 1}	1080	1: {0.5, 0.4, 0.7, 0.4}	2: {0.4, 0.3, 0.5, 0.3}	4: {0.4, 0.2, 0.4, 0.2}
	app_{10}	{1, 12, 12, 12, 12, 1}	180	1: {1.9, 1.2, 2.4, 1.2}	2: {1.5, 1.0, 1.9, 1.0}	4: {1.3, 0.8, 1.6, 0.9}

[3] MAFs depends on the prepared mapping for the different core types: {A9,A15,A7,A17}.

be achieved (*e.g.*, $P_{sys}^{avg} = 1.28$ on the 4×4 platform, $P_{sys}^{avg} = 1.16$ on the 4×8 platform), this is due to the fact that more applications are assigned to a more energy efficient cluster (in terms of heterogeneous clusters or lowest cluster frequencies). On the other hand, more clusters can also lead to reduced P_{sys}^{avg} (*e.g.*, $P_{sys}^{avg} = 1.85$ on the 2×8 platform, $P_{sys}^{avg} = 1.16$ on the 4×8 platform). As applications with different MAFs can be separated into more different clusters, lower cluster frequencies can be achieved for less P_{sys}^{avg}. The Exhaustive results show that more platform resources (*e.g.*, more clusters and more cores within each cluster) leads to reduced P_{sys}^{avg}.

Besides, we can observe that the results of *LPF* and *LEF* are very different from the previous trend. The two strategies give different assignment priorities to clusters. They might achieve close results to the optimal ones for small platforms (*e.g.*, the 4×4 or 2×8 platforms). However, with more cores per cluster, more applications are assigned to the cluster while leaving other clusters empty without any assigned application. Consequently, the frequency of a cluster may increase significantly and result in P_{sys}^{avg} increase. The maximum P_{sys}^{avg} of *LPF* and *LEF* strategies can be observed on the 4 clusters × 24 cores heterogeneous platform. They respectively entail 97.28% (*i.e.*, $\frac{2.11-1.15}{1.15}$) and 21.96% (*i.e.*, $\frac{1.42-1.15}{1.15}$) higher P_{sys}^{avg} than *Exhaustive* strategy.

It can be seen in Figure 9 that *GSACA* achieves optimized results that are close to optimal solutions. Such observations can be seen for different platform sizes, demonstrating the management scalability of *GSACA*. The largest difference between *GSACA* and *Exhaustive* is up to 6.40% on the 4x4 platform. The results indicate that our greedy strategy based on a particular application assignment order is able to avoid assigning too many applications to one cluster and to assign applications with close MAFs to same clusters. Compared to *LPF* and *LEF*, *GSACA* results in P_{sys}^{avg} reduction by up to 80.3% (*i.e.*, $\frac{2.11-1.17}{1.17}$) and 21.2% (*i.e.*, $\frac{1.42-1.17}{1.17}$) respectively.

6.2.2. *Global management solution (general case)*

The general case considers different use-cases, supposing that the assignment in each use-case is dependent on the assignment in the previous use-case. In this evaluation, we consider P_{sys}^{avg} for each global management strategy for the 1023 use-cases that are generated in random sequences. To illustrate the influence of the number of application migrations from one cluster to another, we consider global management allowing *0 migration* and *limited migrations* per use-case.

Allowing 0 migration per use-case: Figure 10 shows P_{sys}^{avg} results of each global management strategy with no migration per use-case. We denote the compared strategies as *Exhaustive_0*, *LPF_0*, *LEF_0* and *GSACA_0*.

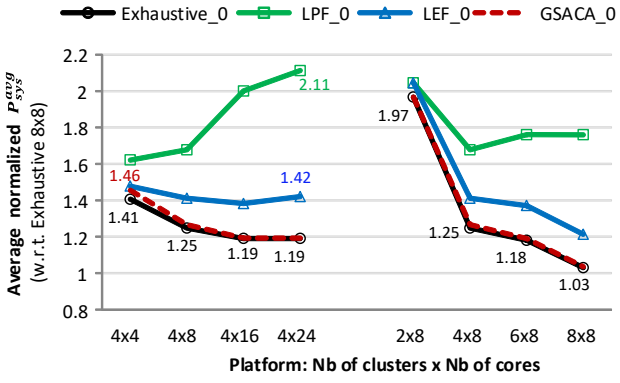


Figure 10: Average normalized P_{sys}^{avg} normalized to Exhaustive (on 8×8 platform) with no migration allowed.

In Figure 10, the P_{sys}^{avg} values obtained on different platform sizes are normalized to the *Exhaustive* result under the constraint of a 8 clusters × 8 cores platform of the special case. This aims to highlight how global management strategies are influenced by previous mappings. *Exhaustive_0* results in a $\frac{1.41-1.28}{1.28} = 10.6\%$ increased P_{sys}^{avg} on the 4×4 platform. Because *Exhaustive_0* considers only newly active application, the assignments of the old existing applications reduce the possibilities of P_{sys}^{avg} reduction. Otherwise, the

trends of P_{sys}^{avg} are similar to the special case (Figure 9). The results of *LPF_0* and *LEF_0* are respectively up to 77.3% and 19.3% higher than *Exhaustive_0*. Differently, *GSACA_0* achieves very close results as *Exhaustive_0* (up to 3.5% difference on the 4x4 platform).

Allowing limited migrations in each use-case: To evaluate the influence of application migrations, we consider GSACA with 1, 2 or 3 allowed migrations (*i.e.*, $Allow_{migration}$ in Algo.1) per use-case, the related strategies are denoted $GSACA\_Allow_{migration}$.

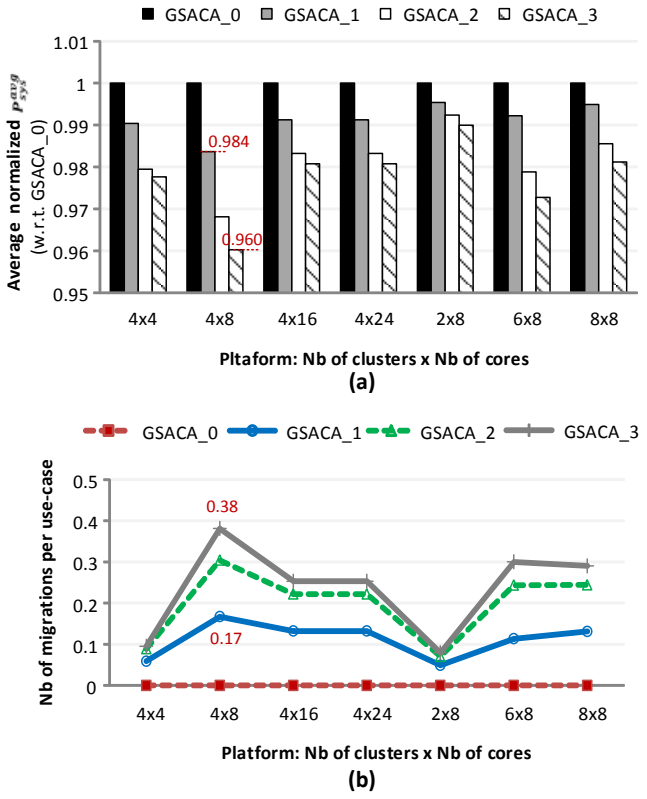


Figure 11: Average P_{sys}^{avg} normalized to *GSACA_0* (constrained by each platform size) for the 1023 use-cases.

Figure 11 (a) compares the average P_{sys}^{avg} normalized to the results of *GSACA_0* among the 1023 use-cases, while part (b) shows the average number of migrations per use-case for each strategy. As can be seen allowing migrations can permit to further reduce P_{sys}^{avg} compared to *GSACA_0* that prohibits migrations. As an example, it can be observed that for *GSACA_1* and *GSACA_3* on the 4×8 platform, P_{sys}^{avg} decreases from 0.984 to 0.960, while migrations increase from 0.17 to 0.38 (see Figure 11 (b)). However it may be interesting to accept migration overheads in order to achieve lower P_{sys}^{avg} if the energy saving after migration is larger than the migration costs. This result can be jeopardized if the migration cost is too high, or if the active duration of the current use-case is not sufficiently long (as stated is Section 5.2.3).

Generally, application migration can be performed when migration benefits overweight migration overheads, namely, $(P_{avg}^{sys\prime} - P_{avg}^{sys}) \times T > E_{app_i}^{m}$ [2]. The left part denotes the energy saving due to migration, while the right part denotes the energy cost of the migration of the application (app_i). The difficulty would be on how to measure the migration overhead of each application ($E_{app_i}^{m}$) and how to correctly predict the duration of the current use-case (T). Migration overhead highly depends on the amount of data to be migrated, and the organization of communication and memory resources. As the use-case duration (*i.e.*, T) increases, the influence of migration overheads on average system power ($P_{avg}^{sys\prime}$ and P_{avg}^{sys}) decrease. Under the assumption that each use-case executes for a long time (as in [25]), our proposed GSACA strategy presents the capability of controlling the number of migration. Further trade-offs between migration benefits and costs would be considered in future work.

6.2.3. *Complexity of global management strategy*

To compare the complexity of the global management strategies, we consider the average exploration time spent for each use-case and for different platform size, results are shown in Figure 12. The exploration is performed for different global management strategies in Visual Studio running on an Intel Core i5 processor, with 16 GB of DDR.

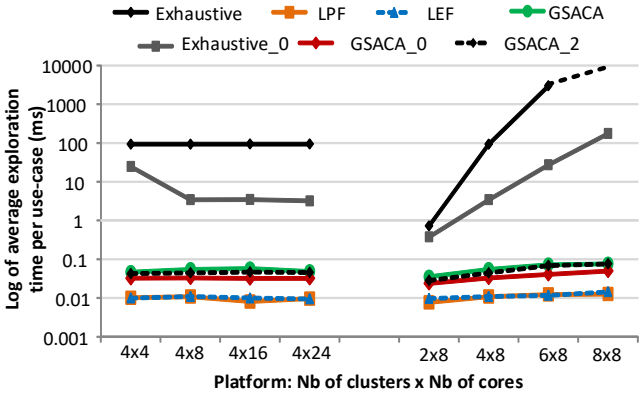


Figure 12: The average exploration time (in ms, log scale) of a use-case among the 1023 use-cases. Results are given for different platform constraints.

We first compare four global management strategies (*i.e.*, Exhaustive, LPF, LEF and GSACA) in the special case. It can be observed that LPF and LEF have the minimum exploration time (less than $0.011ms$ in 4-cluster platforms). However, as previously discussed, these two strategies are limited in energy efficiency on platforms with more resources. Compared to Exhaustive, GSACA needs much less exploration time. For the 4-cluster platforms, Exhaustive strategy uses about $93.6ms$ for exploration, while GSACA requires less than $0.058ms$. This means GSACA runs 1614 times faster than Exhaustive, but entails only up to 6.40% increase in P_{sys}^{avg}. When compared to LPF and LEF,

[2] $(P_{avg}^{sys\prime} - P_{avg}^{sys}) \times T > E_{app_i}^{m}$, where P_{avg}^{sys} and $P_{avg}^{sys\prime}$ refer to the system average power before and after the migration. T denotes a use-case duration. $E_{app_i}^{m}$ denotes the migration energy of the application (app_i).

GSACA achieves 80.3% reduction in P_{sys}^{avg} with a similar level of complexity.

Then, we consider (*i.e.*, Exhaustive_0, GSACA_0 and GSACA_2) for the general case. For the sake of clarity, we do not present the exploration time of LPF_0, LEF_0 which are less than $0.005ms$. It can be observed that GSACA_0 and GSACA_2 are much faster than Exhaustive_0 (up to $24.611ms$ on 4-cluster platforms). For example GSACA_0 runs 746 times faster than Exhaustive_0, with only 3.7% increase in P_{sys}^{avg} (see Section 6.2.2). It can be noticed that GSACA_2 (which allows migrations) takes more exploration time than GSACA_0. It indicates that more time is spent exploring an energy-efficient assignment for each migrated application.

Finally, we can also observe that the exploration time does not significantly change with the number of cores, but increases with the number of clusters. For instance, from 2×8 to 8×8 platforms, the exploration time for Exhaustive_0 increases from 0.038 ms to 175.68 ms. Nevertheless, the proposed strategies (GSACA, GSACA_0, GSACA_2) present slower growth in exploration time (*e.g.*, from 0.028 ms to 0.076 ms for GSACA_2, from 2×8 to 8×8 platforms). This also demonstrates the scalability of the proposed greedy strategy for different platform sizes.

6.3. Evaluations of local management influence

Global decisions can be different due to different resource usages in each cluster. In this work we used FCFS, in a first version denoted $FCFS_S$ using a *single* prepared mapping ($X_{app_i}^{c_i}$). A second version $FCFS_M$ is based on *multiple* prepared mappings like the $GAPVC_M$ strategy. For these two last local strategies, our *run-time selection* strategy is used to select a mapping for each active application before the *run-time mapping combination* by FCFS and GAPVC strategies.

6.3.1. Local and global management solutions

In the first experiment, we evaluated the impact of the local management on the results using the different global management strategies. Table 4 evaluates improvements of using $FCFS_M$ and $GAPVC_M$ in all 1023 use-cases compared to using $FCFS_S$, for the Exhaustive, GSACA, Exhaustive_0 and GSACA_2) global management strategies. LPF and LEF global management strategies are not considered due to their limitations (*i.e.*, high P_{avg}^{sys}) on platforms with more resources.

First, we compare the number of use-cases where the system fails to find a feasible mapping respecting the constraints (see column ① in Table 4). When platform resources are not sufficient for all active applications, the mapping strategy can fail to achieve a possible solution. For the *Exhaustive* global management strategy on small-sized platforms (*e.g.*, 4×4, 2×8), using $FCFS_S$ local management strategy can have up to 296 failed use-cases (over the 1023 in total) in the general case. Using $FCFS_M$ and $GAPVC_M$ local strategies can effectively reduce the number of failures. As $FCFS_M$ and $GAPVC_M$ are based on multiple prepared mappings for each application, the *run-time mapping selection* strategy helps to reduce the number of used cores, and use-cases with more active applications can be better handled as a result. In particular, $GAPVC_M$ can reduce the failed use-cases to 0 (only 1 failure on all our experiments), which is even better than $FCFS_M$ (with up to 54 failures over 1023 use-cases). This is because the GAPVC mapping combination strategy can use fewer cores than FCFS based on the same selected mappings, without sacrificing application performance.

Then, we consider the common feasible use-cases, where the compared strategies (either $FCFS_M$ or $GAPVC_M$ compared with $FCFS_S$) can all achieve feasible solutions. Column ② shows the number of core reduction achievable by $FCFS_M$ and $GAPVC_M$ over $FCFS_S$. For the Exhaustive global management strategy on the 4x4 platform, $FCFS_M$ and $GAPVC_M$ reduce the number of used cores by 16.63% and 32.09% respectively. The maximum resource reductions are respectively 31.80% and 49.95% (Exhaustive_0 on the 2x8 platform). The results show that using multiple prepared mapping leads to better resource usage efficiency.

As the local management can reduce resource usage, it influences also the system power consumption P_{sys}^{avg} (see column ③). As can be seen, the local management strategy can lead to up to 70.25% use-cases with a reduced power. The range of P_{sys}^{avg} reduction is highlighted (column ④). For the 4x4 platform using Exhaustive, $FCFS_M$ and $GSPVC_M$ achieve lower P_{sys}^{avg} (w.r.t. $FCFS_S$) in 47.20% and 67.47% respectively (in their common feasible uses-cases). Among these use-cases, $FCFS_M$ reduces P_{sys}^{avg} from 0.19% to 27.83%, and $GAPVC_M$ achieves up to 31.96% reduction. As $FCFS_M$ and $GAPVC_M$ use fewer cores, they allow more applications to be assigned to more efficient clusters (*e.g.*, due to heterogeneous cluster or low cluster frequency), resulting in a more efficient mapping. The maximum P_{sys}^{avg} reduction (up to 60.72%) can be observed for the *Exhaustive_0* on the 4×8 platform. Additionally, it can be observed that the three local strategies lead to the same P_{sys}^{avg} on the 4×16 and 4×24 platforms. It suggests that, when platform resources are sufficient, the different resource usage inside clusters do not change global management decisions (*i.e.*, application-to-cluster assignments and cluster frequency configurations).

To summarize, the resource usage efficiency achieved by $FCFS_M$ and $GAPVC_M$ can lead to less P_{sys}^{avg} in the overall system. When platform resources are insufficient, $FCFS_M$ and $GAPVC_M$ entail fewer failed use-cases. On the other hand, when platform resources are sufficient, fewer resource usage in each cluster does not change application-to-cluster assignment decisions in global management nor P_{sys}^{avg} results. These observations can be seen for the different global management strategies.

6.3.2. Complexity of the hierarchical strategy

Using different local strategies can lead to different complexity for the whole management system. As previ-

Table 4
Comparison of different local management strategies in hierarchical management among the 1023 use-cases

Global Strategy	platform	① Nb of failed u_m (over 1023 use-cases)			In common feasible u_m					
					② Total reduced cores w.r.t FCFS$_S$		③ pct. of u_m with less P_{sys}^{avg} w.r.t FCFS$_S$		④ P_{sys}^{avg} reduction w.r.t FCFS$_S$ (from min to max)	
		FCFS$_S$	FCFS$_M$	GAPVC$_M$	FCFS$_M$	GAPVC$_M$	FCFS$_M$	GAPVC$_M$	FCFS$_M$	GAPVC$_M$
Exhaustive (Special case)	4x4	273	6	0	16.63%	32.09%	47.20%	67.47%	0.19%-27.83%	<0.01%-31.96%
	2x8	273	0	0	30.23%	48.77%	31.47%	31.60%	0.08%-22.02%	0.08%-22.02%
	4x8	0	0	0	20.24%	32.00%	22.19%	22.19%	<0.01%-9.94%	<0.01%-9.94%
	6x8	0	0	0	20.00%	32.00%	22.19%	22.19%	<0.01%-3.51%	<0.01%-3.51%
	4x16	0	0	0	20.24%	32.00%	0	0	0	0
	4x24	0	0	0	20.24%	32.00%	0	0	0	0
GSACA (Special case)	4x4	273	36	0	20.49%	33.80%	48.40%	66.27%	0.08%-46.20%	0.24%-46.20%
	2x8	273	8	0	30.79%	49.88%	24.13%	26.40%	0.18%-27.74%	0.18%-27.74%
	4x8	0	0	0	20.90%	31.72%	24.14%	24.14%	0.13%-33.62%	0.27%-33.62%
	6x8	0	0	0	19.70%	30.21%	19.75%	19.75%	0.24%-17.98%	0.24%-17.98%
	4x16	0	0	0	21.01%	31.72%	0	0	0	0
	4x24	0	0	0	21.01%	31.72%	0	0	0	0
Exhaustive_0 (General case)	4x4	296	37	1	20.95%	34.82%	43.05%	64.65%	0.10%-48.09%	0.02%-59.07%
	2x8	278	1	0	31.80%	49.95%	34.63%	37.32%	0.18%-51.92%	0.18%-57.92%
	4x8	0	0	0	21.72%	33.77%	43.60%	44.97%	<0.01%-60.72%	<0.01%-60.72%
	6x8	0	0	0	20.51%	31.86%	26.49%	26.49%	<0.01%-37.62%	<0.01%-37.62%
	4x16	0	0	0	21.99%	33.77%	0	0	0	0
	4x24	0	0	0	21.99%	33.77%	0	0	0	0
GSACA_2 (General case)	4x4	307	54	0	21.11%	33.74%	50.43%	70.25%	0.21%-53.93%	0.24%-55.22%
	2x8	287	3	0	31.37%	49.62%	33.70%	36.82%	<0.01%-43.41%	0.17%-43.41%
	4x8	0	0	0	21.52%	33.22%	22.19%	22.19%	<0.01%-9.94%	<0.01%9.94%
	6x8	0	0	0	19.80%	30.64%	27.18%	27.47%	<0.01%-30.85%	<0.01%-30.85%
	4x16	0	0	0	21.74%	33.22%	0	0	0	0
	4x24	0	0	0	21.74%	33.22%	0	0	0	0

[1] Common feasible use-cases: the use-cases where the considered local management strategy ($FCFS_M$ or $GAPVC_M$) and the counterpart $FCFS_S$ can achieve a mapping result.

ously introduced in Section 5.1, global management verifies the resource constraint for a new assigned application by estimating N_j^{used} in a cluster. This value can be estimated through the *pessimistic estimation* directly by the global manager or *accurately* by invoking the local management. $FCFS_S$ does not consider resource optimization within a cluster, and its N_j^{used} is equals to the *pessimistic estimation* in global management. On the other hand, $FCFS_M$ and $GAPVC_M$ optimize resource usage based on multiple prepared mappings. The two local strategies are invoked for an *accurate* estimation when $0 < N_j^{used,max} - N_j <= 10$, as introduced early. Consequently, more calculations are required to complete management decisions in certain use-cases.

Table 5 shows the average exploration time spent per use-case when using different local strategies. Due to the scalability issue of the Exhaustive approach, we only consider GSACA and GSACA_2 as global management strategies for this comparison. Table 5 indicates that $FCFS_M$ and $GAPVC_M$ require more time to find a possible mapping. When platform resources are insufficient (*i.e.*, 4x4, 2x8 platforms), the exploration time of $FCFS_M$ and $GAPVC_M$ are 11.98 and 14.17 times bigger than the pessimistic evaluation (*i.e.* using $FCFS_S$). But as seen in the previous section (Table 4) they considerably reduce the number of failed mappings. When platform resources are sufficient

Table 5: Normalized exploration time for the hierarchical manager

Global	Platform	Time (ms) of FCFS$_S$	Normalized exploration time		
			FCFS$_S$	FCFS$_M$	GAPVC$_M$
GSACA	4x4	0.049	1	11.98	14.17
	2x8	0.036	1	4.90	8.76
	4x8	0.055	1	3.25	4.28
	6x8	0.072	1	2.34	2.83
	4x16	0.053	1	1.11	1.13
	4x24	0.058	1	0.95	0.97
GSACA_2	4x4	0.044	1	11.87	13.39
	2x8	0.027	1	7.13	11.66
	4x8	0.040	1	3.48	6.28
	6x8	0.055	1	3.04	3.87
	4x16	0.043	1	1.20	1.15
	4x24	0.044	1	1.12	1.07

(*i.e.*, 4x8, 6x8 platforms), the difference in exploration time decreases. When the platform resources are quite enough, there is almost no difference in the exploration time. In this situation the number of active applications competing for limited resources is reduced (*i.e.*, when $0 < N_j^{used,max} \times a_{i,j} - N_j <= 10$), and $FCFS_M$ and $GAPVC_M$ are less called for *accurate calculation* of N_j^{used} (see ❻ in Figure 4).

7. Conclusions

This paper presents a hierarchical run-time management strategy to achieve the overall energy efficiency of a heterogeneous cluster-based multi/many-core platform, on which multiple applications can execute concurrently and dynamically. Based on some design-time prepared data for each application, the global management *integratedly* determines application-to-cluster assignment and sets cluster frequency, while the local management in each cluster determines task-to-core allocation and scheduling. The global management problem is formulated as a 0-1 ILP model using the information of only one prepared mapping for each application. A greedy global strategy is proposed to achieve optimized solutions to the 0-1 ILP model while reducing computation complexity. The local management aims to minimize resource usage within each cluster by using the information of multiple prepared mappings for each application. For this purpose, we propose a local management strategy consisting of a mapping selection and a combination process of the selected mappings.

Our experimental results show that the proposed global management strategy outperforms the state-of-art LPF and LEF strategies, in terms of management scalability on different platform sizes. Compared to LPF and LEF, our proposed global strategy (GSACA, a greedy strategy) can reduce the achieved average power consumption (per use-case) by 80.3%, with a similar level of complexity. Compared to the exhaustive strategy (providing the optimal solution), the exploration time of our proposed GSACA is up to 1614 times faster, with at maximum a 6.4% difference in the achieved average power consumption (per use-case). Furthermore, our greedy strategy is able to assign only the newly active applications as well as to allow limited application migration from one cluster to another. Our evaluation indicates that allowing 0.21 more migration per use-case can lead to 2.6% improvement in energy efficiency. Additionally, compared to the state-of-the-art local management strategy FCFS, our proposed local strategy can achieve up to 50% resource usage reduction. The resource reduction in the local management can change the global management decision and consequently improve the energy efficiency of the overall system up to 60.72% (per use-case).

In the future, we will extend our work by incorporating a communication energy model into the local management. Such an extension requires additional calculation of communication energy in the 0-1 ILP model, without changing the hierarchical management structure. Besides, we will also take into account migration overheads to provide a more accurate evaluation of the proposed strategy. By introducing an estimation of migration cost for each application, our management process could be further optimized to appropriately select active applications to migrate. Finally, we will study the introduction of dynamic power management techniques to further reduce the energy consumption, and to handle static power reduction (by switching-off unused clusters for example).

Acknowledgement

This work was supported by China Scholarship Council under contract number 201606380135.

References

[1] Exynos 5 octa (5422), Available:http://www.samsung.com/exynos, 2019.

[2] A. Butko, F. Bruguier, D. Novo, A. Gamatié, G. Sassatelli, Exploration of performance and energy trade-offs for heterogeneous multicore architectures, arXiv preprint arXiv:1902.02343 (2019).

[3] S. Pagani, A. Pathania, M. Shafique, J.-J. Chen, J. Henkel, Energy efficiency for clustered heterogeneous multicores, IEEE Transactions on Parallel and Distributed Systems 28 (2016) 1315–1330.

[4] Y. G. Kim, J. Kong, S. W. Chung, A survey on recent os-level energy management techniques for mobile processing units, IEEE Transactions on Parallel and Distributed Systems 29 (2018) 2388–2401.

[5] L. Benini, D. Bertozzi, M. Milano, Resource management policy handling multiple use-cases in mpsoc platforms using constraint programming, in: International Conference on Logic Programming, Springer, 2008, pp. 470–484.

[6] H. Aydin, Q. Yang, Energy-aware partitioning for multiprocessor real-time systems, in: Proceedings International Parallel and Distributed Processing Symposium, IEEE, 2003, pp. 9–pp.

[7] H. Ali, U. U. Tariq, Y. Zheng, X. Zhai, L. Liu, Contention & energy-aware real-time task mapping on noc based heterogeneous mpsocs, IEEE Access 6 (2018) 75110–75123.

[8] U. U. Tariq, H. Ali, L. Liu, J. Panneerselvam, X. Zhai, Energy-efficient static task scheduling on vfi-based noc-hmpsocs for intelligent edge devices in cyber-physical systems, ACM Transactions on Intelligent Systems and Technology (TIST) 10 (2019) 1–22.

[9] T. S. Muthukaruppan, M. Pricopi, V. Venkataramani, T. Mitra, S. Vishin, Hierarchical power management for asymmetric multi-core in dark silicon era, in: Proceedings of the 50th Annual Design Automation Conference, ACM, 2013, p. 174.

[10] V. Petrucci, O. Loques, D. Mossé, R. Melhem, N. A. Gazala, S. Gobriel, Energy-efficient thread assignment optimization for heterogeneous multicore systems, ACM Transactions on Embedded Computing Systems (TECS) 14 (2015) 15.

[11] W. Quan, A. D. Pimentel, A hierarchical run-time adaptive resource allocation framework for large-scale mpsoc systems, Design Automation for Embedded Systems 20 (2016) 311–339.

[12] P.-C. Hsiu, P.-H. Tseng, W.-M. Chen, C.-C. Pan, T.-W. Kuo, User-centric scheduling and governing on mobile devices with big. little processors, ACM Transactions on Embedded Computing Systems (TECS) 15 (2016) 17.

[13] A. Kanduri, A. Miele, A. M. Rahmani, P. Liljeberg, C. Bolchini, N. Dutt, Approximation-aware coordinated power/performance management for heterogeneous multi-cores, in: 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC), IEEE, 2018, pp. 1–6.

[14] 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 Transactions on Computers 66 (2016) 488–501.

[15] H.-E. Zahaf, A. E. H. Benyamina, R. Olejnik, G. Lipari, Energy-efficient scheduling for moldable real-time tasks on heterogeneous computing platforms, Journal of Systems Architecture 74 (2017) 46–60.

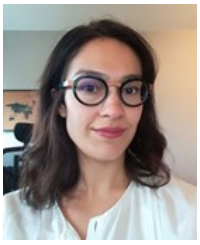
[16] S. K. Mandal, G. Bhat, J. R. Doppa, P. P. Pande, U. Y. Ogras, An energy-aware online learning framework for resource management in heterogeneous platforms, ACM Transactions on Design Automation of Electronic Systems (TODAES) 25 (2020) 1–26.

[17] A. L. del Mestre Martins, A. H. L. da Silva, A. M. Rahmani, N. Dutt, F. G. Moraes, Hierarchical adaptive multi-objective resource

management for many-core systems, Journal of Systems Architecture 97 (2019) 416–427.
[18] 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 Computing Surveys (CSUR) 50 (2017) 1–40.
[19] E. A. Lee, D. G. Messerschmitt, Static scheduling of synchronous data flow programs for digital signal processing, IEEE Transactions on Computers C-36 (1987) 24–35.
[20] T. Lionel, B. Pascal, S. Giles, R. Michel, An introduction to multicore system on chip. trends and challenges. multiprocessor system-on-chip: Hardware design and tool integration. pag. 1-18, 2010.
[21] A. K. Singh, M. Shafique, A. Kumar, J. Henkel, Resource and throughput aware execution trace analysis for efficient run-time mapping on mpsocs, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 35 (2016) 72–85.
[22] U. U. Tariq, H. Wu, S. Abd Ishak, Energy and memory-aware software pipelining streaming applications on noc-based mpsocs, Future Generation Computer Systems (2020).
[23] C.-H. Hsu, U. Kremer, M. Hsiao, Compiler-directed dynamic voltage/frequency scheduling for energy reduction in microprocessors, in: ISLPED'01: Proceedings of the 2001 International Symposium on Low Power Electronics and Design (IEEE Cat. No. 01TH8581), IEEE, 2001, pp. 275–278.
[24] J. Mei, K. Li, J. Hu, S. Yin, E. H.-M. Sha, Energy-aware preemptive scheduling algorithm for sporadic tasks on dvs platform, Microprocessors and Microsystems 37 (2013) 99–112.
[25] W. Quan, A. D. Pimentel, A hybrid task mapping algorithm for heterogeneous mpsocs, ACM Transactions on Embedded Computing Systems (TECS) 14 (2015) 14.
[26] A. K. Singh, A. Kumar, T. Srikanthan, A hybrid strategy for mapping multiple throughput-constrained applications on mpsocs, in: Proceedings of the 14th international conference on Compilers, architectures and synthesis for embedded systems, ACM, 2011, pp. 175–184.
[27] S. Yang, S. Le Nours, M. mendez Real, S. Pillement, Mapping and frequency joint optimization for energy efficient execution of multiple applications on multicore systems, in: 2019 Conference on Design and Architectures for Signal and Image Processing (DASIP), IEEE, 2019, pp. 29–34.
[28] A. K. Singh, M. Shafique, A. Kumar, J. Henkel, Resource and throughput aware execution trace analysis for efficient run-time mapping on mpsocs, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 35 (2015) 72–85.
[29] Sdf3, Available:http://www.es.ele.tue.nl/sdf3, 2019.

Simei Yang received the B.S. and M.S. degrees in microelectronics from Sun Yat-sen University, Guangzhou, China, in 2014 and 2016, respectively. She received her Ph.D. degree in the embedded system in IETR UMRCNRS 6164-Polytech Nantes - Université de Nantes, France, in 2020. She is currently a postdoctoral researcher in IMEC and KU Leuven. Her research interests include DNN evaluation framework on compute-in-memory architectures, run-time management of energy efficiency on multi/many-core system, and system-level modelling and simulation.

Sébastien Le Nours received the M.S. degree in electrical engineering and computer science from ISEN engineer school (Brest, France) in 2000. He received the PhD degree in Electronic from National Institute of Applied Science (Rennes, France) in 2003. He is an associate professor in electrical engineering and computer science at Polytech Nantes, the engineering school of University of Nantes (France), where he has been since 2004. His research concerns system-level design and methodologies, embedded computer systems, specification and modeling languages, and communication system design. Dr. Le Nours is a member of the IETR laboratory, UMR CNRS 6164.



Maria Méndez Real is Associate Professor in Electrical and Computer Engineering at Ecole Polytechnique Universitaire de l'Université de Nantes, France. Her research is within IETR Lab, UMR CNRS 6164. She received her PhD in Electrical and Computer Engineering from Université de Bretagne-Sud, France, in 2017 within the frame of the French ANR TSUNAMY project within Lab-STICC Lab. In 2015 she spent 4 months as invited researcher at Ruhr-University of Bochum, Germany. Her research interests include System and Hardware Security, Multi and Many-core Systems, Networks-on-Chip and Simulators/Virtual Platforms.

Sébastien Pillement received the PhD degree and Habilitation degrees in computer engineering, respectively, from the University of Montpellier II and the University of Rennes 1. Since 2012, he is a full professor at Ecole Polytechnique of Nantes University, France. From 1999 to 2012 he was with IUT in Lannion, the subdivision of the University of Rennes 1, France, and a research member of the CAIRN INRIA Research Team. He is now a member of the IETR laboratory, UMR CNRS 6164. His research interests include dynamically reconfigurable architectures, system on chips, design methodology and Network on Chip (NoC)-based circuits. He focuses his research on designing flexible and efficient architectures managed in real time. He is the author or coauthor of about 120 journal and conference papers.