Application-Arrival Rate Aware Distributed Run-Time Resource Management for Many-Core Computing Platforms

Vasileios Tsoutsouras, *Student Member, IEEE*, Sotirios Xydis, *Member, IEEE*, and Dimitrios Soudris, *Member, IEEE*

Abstract—Modern many-core computing platforms execute a diverse set of dynamic workloads in the presence of varying application arrival rates. This inflicts strict requirements on run-time management to efficiently allocate system resources. On the way towards kilo-core processor architectures, centralized resource management approaches will most probably form a severe performance bottleneck, thus focus has been turned to the study of Distributed Run-Time Resource Management (DRTRM) schemes. In this article, we examine the behavior of a DRTRM of dynamic applications with malleable characteristics against stressing incoming application interval rate scenarios, using Intel SCC as the target many-core system. We show that resource allocation is highly affected by application input rate and propose an application-arrival aware DRTRM framework implementing an effective admission control strategy by carefully utilizing voltage and frequency scaling on parts of its resource allocation infrastructure. Through extensive experimental evaluation, we quantitatively analyze the behavior of the introduced DRTRM scheme and show that it achieves up to 44 percent performance gains while consuming 31 percent less energy, in comparison to a state-of-art DRTRM solution. In comparison to a centralized RTRM, the respective metric values rise up to 62 and 45 percent performance and energy gains, respectively.

Index Terms—Distributed run-time resource management, many-core systems, intel SCC, application admission regulation policy

1 Introduction

Contemporary technological achievements and system designs such as cloud computing drive innovation towards many-core computer architectures and dynamic applications design [1]. Many-core architectures [2], [3], [4] are nowadays a reality, proposed as an effective computer organization to address the ever increasing user demands for higher performance, reliability and lower power consumption. The nature of applications' design is also evolving by incorporating dynamic characteristics such as high variability in workload, self-awareness and malleability, i.e seamless run-time adaptivity to available system resources [5], [6], [7].

The combination of highly dynamic, parallel applications and emerging technologies in many-core systems dictates the need for run-time decision making regarding resource distribution, which incorporates sophisticated logic in an effort to meet the requirements of high performance, low power consumption, safety and reliability. Inevitably, this comes at the price of high computational requirements in order to provide results within acceptable time limits.

• *The authors are with the School of Electrical and Computer Engineering, National Technical University of Athens, Athens 157 72, Greece. E-mail: {billtsou, sxydis, dsoudris}@microlab.ntua.gr.*

Manuscript received 18 Dec. 2016; revised 13 Dec. 2017; accepted 27 Dec. 2017. Date of publication 2 Feb. 2018; date of current version 14 Sept. 2018. (Corresponding author: Vasileios Tsoutsouras.)
Recommended for acceptance by S. Le Beux, P.V. Gratz, and I. O'Connor.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TMSCS.2018.2793189

To alleviate the computational bottleneck, the resource allocation paradigm has shifted from centralized to Distributed Run-Time Resource Management (DRTRM) decision making processes. The new paradigm has been adopted, leading to user-space DRTRM implementations [7], [8], design of new Operating Systems [9], [10], [11] and even novel implementations of the Linux Operating System with distributed, replicated kernel instances [12]. In overall, the advantages of DRTRM are increased scalability, inherent distribution of computational burden for decision making, support for heterogeneous systems, as well as enhanced system reliability, i.e., eliminating the single point of failure of centralized approaches.

However, the nature of distributed decision making does come with an increased complexity in the resource management process. The lack of a single point with overview of the platform leads to limited ability to adjust in scenarios that the need for resources is stressed, since numerous distributed agents need to communicate via exchanged messages in order to enforce a global policy. In this work, we identify this limited adaptivity ability by examining resource stressful scenarios, resulting from the arrival rate of incoming applications on many-core systems. We show that a very fast and resource hungry scenario of incoming applications can be the breaking point for the efficiency of the distributed framework.

The effects of the arrival rate of incoming execution requests have been widely investigated for different target systems such as Cloud infrastructure [13], Map-Reduce Clusters [14] and many core-systems [15] but all solutions

rely on centralized resource management, which allows for effective decision making under heavy input traffic. We differentiate from these works by analysing and correlating the application arrival rates in respect to the internal mechanisms of DRTRM and propose its extended application-arrival aware version, which is capable of dynamically adapting to stress-marked scenarios in a distributed manner.

Our analysis of DRTRM internals reveals that its resource allocation process forms a hierarchy in the inter-play of different agents running on the target many-core system. Furthermore, these different agents exhibit unbalanced workload characteristics by design, as for example a subset of the available system cores are dedicated only to management-related tasks, without any involvement in application workload execution. Consequently, their execution profile makes them promising candidates to enforce Voltage and Frequency scaling (VFS) techniques to balance their power and computation needs. We utilize the aforementioned concept of VFS at the DRTRM level in order to efficiently regulate resource management requirements according to system stress conditions, without compromising the distributed nature of our management framework. More specifically, the novel contribution of our work are:

- We propose and design an Application-Arrival aware Distributed Run-Time Resource Management framework for many-core systems which utilizes VFS techniques in order to enforce an application admission regulation policy in a purely distributed manner, requiring the communication of only a small subset of the system's cores.
- We provide a second level of exploitation of our VFS based techniques by meticulously mapping the parts of DRTRM on the system according to their computational requirements in order to maximize the energy consumption gains of our framework.
- We implement the proposed DRTRM on Intel Single Chip Cloud Computer (SCC) [2], a many-core system which gives us the opportunity to validate our proposed techniques in an actual setup and capture and analyze the behaviour of applications' workload execution and system's energy consumption.
- We provide an exploratory analysis of the different design knobs of the proposed framework in order to fine-tune its parameters, thus maximizing the gains in application execution latency as well as system-wide energy consumption.

The rest of the paper is organized as follows: Section 2 is dedicated to related work. Section 3 presents our Application-Arrival Aware DRTRM by summarizing its resource allocation mechanisms in Section 3.1 and detailing the proposed admission control in Section 3.2. Section 4 includes an extensive experimental analysis of the behaviour and efficiency of the proposed DRTRM solution on Intel SCC platform, while Section 5 concludes the paper.

2 Related Work

Distributed Run-Time Resource management has been investigated in various works as in [7], [8], [16], [17], with main focus on minimizing application execution latency targeting a many-core system. In [16], [17] authors propose distributed mapping using clusters of cores, which are resized at run-time according to the needs of running applications. Resources are negotiated and migrated between applications with the help of local agents and a global supervising agent. Such a global agent is not present in [7], [8] and thus application management is performed by distributed agents which negotiate individually for resources.

However, in all of these works the arrival rate of applications is not taken into account. This parameter is investigated in [18], where a heuristic for application admission control is proposed to aid the service provider of cloud or grid based systems to meet the Quality of Service requirements of the user. In [15], a job arrival aware scheduler is proposed targeting asymmetric multi-processors. The centralized scheduler has a complete overview of the system and uses queuing theory concepts and arrival rate predictions to make run-time decisions for the migration of jobs between different cluster of PEs.

Authors of [19] target embedded, hard, real-time systems and propose a feedback loop approach using control theory to regulate and fine-tune application admission. Their approach is similar to our proposed one, but targets systems of only a few processors, whereas ours is designed for many-core systems. In [20] the concept of a job queue in a many-core system is extended by introducing Isonet, a hardware based dynamic load distribution and balancing manager. This manager makes the selection of the jobs to be executed based on micro-network of load balancing modules, which take into account the current load conditions of the system, in an ultimate effort to maximize system speedup.

The described approaches, mainly focus on performance optimization, neglecting energy optimization goals, which is becoming an important requirement for contemporary management frameworks. In [21] authors propose ARTE, an Application-specific Run-Time Management framework, which makes use of queuing theory concepts to maximize the QoS of a many-core system, while respecting its power budget. The employed queuing model utilizes specific information for each application, resulting from design time analysis. In contrast to our approach, ARTE operates in a centralized manner embedded inside the OS of a multi-core system. Another centralized approach is VARSHA++ [22], a run-time framework extended with Dynamic Voltage Scaling capabilities, for application mapping in multi-processor chips operating under dark-silicon constraints. An input queue is used for application arrival, while task mapping and scheduling is performed taking into account variation characteristics and reliability predictions for the target chip.

Al-Fareque et al. [23] proposed an agent-based framework to reduce peak temperature combined with enhanced performance and reduced energy consumption. Agents' negotiations are based on a supply/demand economical model that distributes power in a proactive manner. Agent-based management is also utilized in [24], performing distributed task migration for thermal management. Neural networks are used to predict peak temperatures in cases of workload variation, showing higher performance and reduced migration overhead compared to other centralized predictive dynamic thermal managers. An Agent-based power management presented in [25] provides the opportunity of power state control of resources for each individual

application operating on the system. The determination of each power state is performed with respect to improving the overall energy consumption of the system under management. To achieve that, a distributed power management approach is proposed based on game-theory, which achieves significant results both in scalability and energy efficiency.

3 Application-Arrival Aware DRTRM

3.1 DRTRM Architecture

The proposed DRTRM framework targets many-core homogeneous platforms, where all Processing Elements (PEs) are mapped on the same chip. It is assumed that communication between PEs is achieved in a message passing way via a Network-on-Chip infrastructure. Nevertheless, our design is not dependent on the communication scheme of the PEs so it can be adapted to operate on top of shared memory based, many-core architectures.

3.1.1 Target Application Model

The proposed framework focuses on the management of parallel applications which exhibit malleable characteristics [7], i.e., they can be re-sized up/down according to resources availability. We refer to resources as the PEs occupied by each application and assume no communication between two different running applications. Each application is described by three parameters, which are its workload W, its parallelism variance σ, and its average parallelism A. According to the model utilized in [7], Eqs. (1) and (2) provide the speedup of a parallel application, that is executed on n worker PEs of a many-core system

$$\underbrace{S(n) = \begin{cases} \frac{nA}{A+\frac{\sigma}{2}(n-1)} & ,1 \leq n < A \\ \frac{nA}{\sigma(A-\frac{1}{2})+n(1-\frac{\sigma}{2})} & ,A \leq n < 2A-1 \\ A & ,n \geq 2A-1 \end{cases}}_{\sigma<1} \quad (1)$$

$$\underbrace{S_(n) = \begin{cases} \frac{nA(\sigma+1)}{\sigma(n+A-1)+A} & ,1 \leq n \leq A+A\sigma-\sigma \\ A & ,n > A+A\sigma-\sigma \end{cases}}_{\sigma\geq 1}. \quad (2)$$

Using Eqs. (1) and (2), the remaining execution time of a running application with workload W is calculated as [7]

$$T_{finish} = \frac{W}{S(n)}. \quad (3)$$

Due to the lack of a central agent responsible for application mapping, each application is managed atomically, distributing its workload to its active worker cores. Adopting application management concepts from many-core Operating Systems [10] and large scale distributed systems [26], a dedicated, unique PE per application, named *Manager core*, controls its life-cycle and manages its resources according to the embedded directives of the application developer. Its additional key aspect is to perform all the necessary actions for inter-application run-time resource negotiation, which is critical for optimizing the speedup of its application.

In the context of this work, the Manager core is responsible for distributing workload to the working nodes and gathering the output of their calculations. Given that our design aims at large many-core systems, we choose to have no application workload executed by the Manager core in order to minimize any run-time overhead imposed on application management. In addition, since we target highly scaling applications the imposed speedup overhead of this choice is greatly reduced. In the general case, a Manager core is a software entity/process which can be co-scheduled with other processes on a single core, i.e., our choice is not a hard design constraint.

We define as R_{tot} the total number of Processing Elements of the target many-core system and as N_{apps} the total number of instantiated applications at a given moment. The relationship between N_{apps} and R_{tot} is

$$1 \leq N_{apps} \leq R_{tot}. \quad (4)$$

N_{apps} is always equal or greater to 1 under the convention that there is an administrative application which is responsible for idle computational resources handling (see Section 3.1.3). If we define as R_i the resources occupied by the ith application then the following two equations are valid

$$R_i \neq R_j, \forall i,j, i \neq j, i,j \leq N_{apps} \quad (5)$$

$$\sum_{i=1}^{N_{apps}} \aleph_{R_i} = R_{tot}, \quad (6)$$

implying that that two applications cannot share the same computational resources (Eq. (5)) and the total computational resources occupied by all applications are equal to the total resources of the system (Eq. (6)). The maximum PEs that an application can occupy are dictated by the least number of cores that maximize its speedup.

3.1.2 Implemented Malleable Application

In order to provide an experimental setup able to represent realistic characteristics of emerging workloads, we implement application malleability within an integer matrix multiplication code between a square matrix of size M×M and a vector of size M×1. This computational kernel was chosen due to its significance in applications of High Performance Computing (e.g., Linear Algebra applications) and Machine Learning (e.g., Support Vector Machines classifiers). In addition, the patterns of the involved computations allow the development of a parallel application, which respects the model of malleable applications.

To create computationally intensive applications, the core multiplication is repeated W times and this variable is considered the workload of the application. Resizing is enabled only at specific synchronization points between these repetitions. The speeudp function $S(n, M)$ and remaining execution time T_{rem} of the application are modelled as

$$S(n,M) = \frac{Exec\_t(1,M)}{Exec\_t(n,M)}; \quad T_{rem} = W_{rem} * Exec\_t(n,M), \quad (7)$$

where W_{rem} refers to the remaining multiplication repetitions, n is the number of worker cores and $Exec\_t$ is a function returning the execution time for one instance of the matrix multiplication for n cores. $Exec\_t$ is dependent both on the input data-set size as well as on the number of

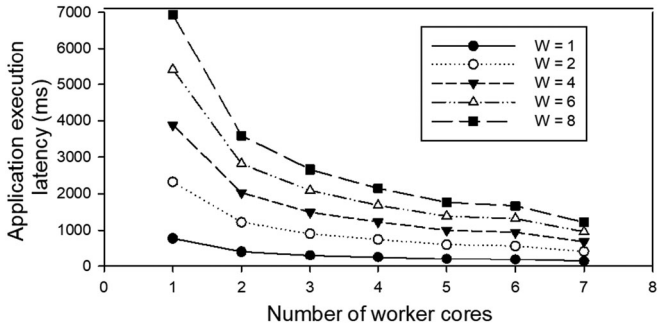


Fig. 1. Performance of malleability model in respect to #cores and workload size for matrix size M = 4,096.

processes working in parallel and is derived from extensive profiling on the target platform. The maximum number of workers for the implemented application is set to 8. Fig. 1 shows the execution latency of various workloads on Intel SCC in respect to allocated resources. We observe that as dictated by Eqs. (1)–(3) the scaling of the implemented application is irrelevant to its workload W and respects the linearity between W values and remaining execution time.

To achieve this behavior, we force DRTRM to allocate one parallel process per core, thus eliminating co-scheduling interference effects, since SCC does not support hyper-threading at the core level. At run-time, the Manager communicates to each worker the upper and lower bound of the consecutive rows of the input matrix that it has to multiply. The workload is evenly distributed and the workers do not exchange any information with one another. Each worker writes in the memory of its Manager core the computed results. In order to alleviate the overheads of memory block caching that limits the benefits of dynamic malleability, we carefully tile the workload distributed to each core to fit the SCC node's cache capacity, while also performing data prefetching at the initialization of the application.

The implemented application exhibits collective communication during workload distribution and reduction of computed results by the Manager core. Different inter-worker communication patterns can be modelled in the Speedup curve of the application and be taken into account by the Manager core at run-time.

3.1.3 Distributed Information Tracking

The purely distributed nature of the presented framework, dictates the need for a consistent way of system monitoring at run-time. Since no processing entity is aware of the status of the complete system, a number of PEs are allocated for monitoring a part of the system and communicating this information to the rest of the nodes. These dedicated PEs will be referred to as *Controller cores* and maintain a record of active Manager cores in a system region called *Cluster*. This record is called **D**istributed **D**irectory **S**ervice (*DDS*).

Clusters span to non-overlapping areas which are fixed at design-time and can be reconfigured at system initialization only. The number of Controller cores and the topology of their *Clusters* is an important design parameter which affects workload and traffic distribution on the system. Examples of different *Cluster* topologies are illustrated in Fig. 9, where cores of the same color belong to the same *Cluster* and are monitored by the same Controller core.

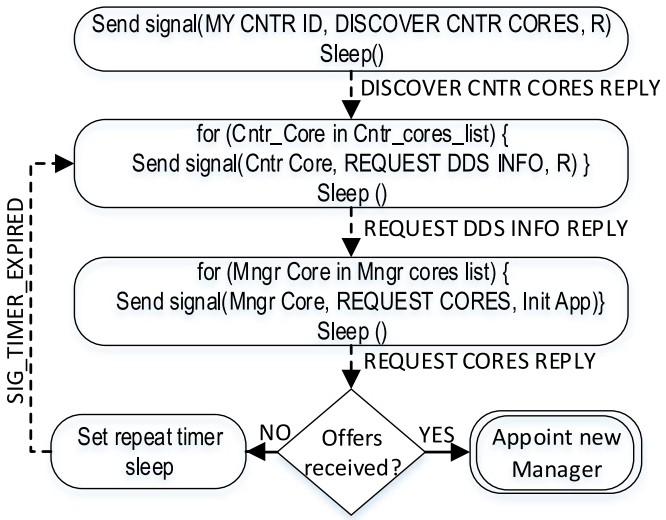


Fig. 2. Overview of the functionality of an initial core.

In order for a core to acquire information about application activity inside a region R of the system, defined by a center C and a radius r, it must issue such a request to the responsible Controller cores. A *DISCOVER_CNTR_CORES* signal requests information about which Controller cores monitor region R. A *REQUEST_DDS_INFO* signal informs about the active Manager cores inside region R. *ADD_CORES_DDS* and *REMOVE_CORES_DDS* signals issue a request by a Manager for appropriate DDS list modification. By design, the Controller core is also the owner of idle cores inside its *Cluster*. As a consequence, it receives requests by other agents for these resources, which are replied by offers according to the availability of resources.

3.1.4 Application Instantiation

The final piece to capture the complete life-cycle of an incoming application on the target many-core system is the part of its instantiation and initialisation. This task is undertaken at run-time by *Initial cores* that are responsible to discover at least one PE for the new application to be executed on. One is appointed per application, randomly assigned at run-time to enable a widespread distribution of Initial cores across the platform, thus avoiding the repeated assignment to a specific PE. It acts only as the means for the instantiation of a new application, whose actual mapping is dynamically determined and is not dependent on the position or characteristics of the Initial core. Its task is completed when a least one available PE has been discovered to facilitate the execution of the new incoming application.

In Fig. 2, the overview of the functionality of the Initial core is presented. First, a region R is defined where the Initial core will look for available resources. Then, a *DISCOVER_CNTR_CORES* signal is sent to its Controller to discover which Controllers are monitoring region R. Having sent the signal, it pauses until the reply is received. Upon the reply reception, it proceeds to the next state where a *REQUEST_DDS_INFO* signal is sent to all Controller cores responsible for region R. This step will provide the information of which Manager cores possess resources inside R. Another pause phase is initiated which ends when replies from all Controllers have been gathered. Next, a *REQUEST_FOR_CORES* signal is sent to every

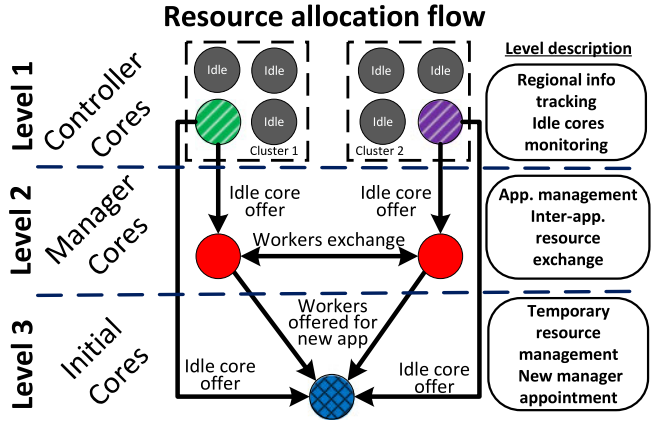


Fig. 3. Cores hierarchy and dependencies in DRTRM.

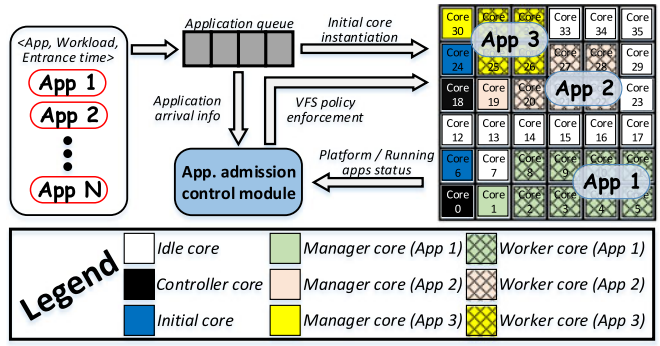


Fig. 4. The proposed application-arrival aware DRTRM.

Manager and Controller inside region R requesting resource offers for the new application. Then a timer is set and the Initial core pauses again waiting for offers. When the timer expires, if there is at least one not null offer, the new Manager core is instantiated accordingly. In the opposite case, the overall process is re-executed after a pre-defined interval.

3.1.5 Resource Management Overview

The overall resource management goal of DRTRM is to serve all applications requiring admission, while maximizing the performance of the system. This is achieved by increasing the resources of each application in such a way that the total performance is maximized in addition to serving as many incoming applications simultaneously as possible. The optimization goal of our design is overall system service latency, in contrast to other RTRM frameworks [27], which consider only performance maximization of individual applications. Its constraint is that no PE of the system is shared by two applications as stated in Eqs. (5) and (6). To dynamically guarantee the balanced allocation of PEs to all applications, resource bargaining rounds are performed periodically between active application instances, $A_i, A_j, i, j \in \mathbf{A}$.

We adopt the negotiation mechanism proposed in [8], i.e., a re-assignment of resources is performed between A_i, A_j whenever the following condition is true

$$Speedup(A_i, \#R_i - 1) < Speedup(A_j, \#R_j + 1). \quad (8)$$

This means that a PE is reallocated to A_j if the speedup loss of A_i is lower than the speedup gain of A_j. In case of a reallocation, it follows that the difference in total System Speedup (SSp) is

$$\Delta SSp = Speedup(A_j, \#R_j + 1) - Speedup(A_i, \#R_i - 1) > 0, \quad (9)$$

meaning that system speedup/throughput is increased.

The aforementioned negotiation mechanism evaluates the resource exchange possibility between two distributed agents. The exchange of resources on the system takes place only in a peer-to-peer fashion, so when many agents bargain for resources they communicate their requests to all possible resource providers. More precisely, the negotiation mechanism is utilized by Manager cores for inter-application resource exchange and by Initial cores in the process of instantiating a new application. As far as Controller cores are concerned, offering resources (idle PEs) is trivial, since they exhibit no speedup loss. Consequently, they reply positively to a request for resources, provided that there are idle cores inside the designated region.

The amount of resource bargaining decisions, was used as a representative metric to evaluate the effect of centralized resource management on simulated, scaled up versions of many-core systems. One agent is responsible for the system overview and applications negotiate resources at run-time via this agent. Our experiments validated the overburdening of the centralized manager, showing that there is a non-linear correlation between system size and the necessary resource negotiation instances, e.g., scaling up from 48 to 1,200 cores (25×) results in a 113× increase in the negotiation instances, while the ratio of incoming applications and system resources is maintained constant.

The presented DRTRM scheme implicitly imposes a hierarchy to the different roles of PEs and the resource allocation flow. This hierarchy, presented in Fig. 3, creates dependencies between PEs, which will be the key element of the extended, Application-Arrival Aware DRTRM. At the top level (Level 1), Controller cores are the building blocks of information tracking and idle resources ownership. In Level 2, Manager cores supervise application execution and facilitate inter-application resource exchange, which is critical for the system to reach an optimized state, ensuring no starvation incidence for any application. Initial cores lie in Level 3 and temporarily acquire resources from the upper two levels, in their effort to initialize a new application.

In overall, the resource management hierarchy as well the appointment of dedicated tasks, either at design time, i.e., Controller cores or at run-time, i.e., Manager, Initial cores, is mandatory to account for the lack of centralized system management. The inherent trade-off is the reduction of available workers, which we consider tolerable in future many-core systems with hundreds or thousands of PEs. In addition, the presented hierarchical scheme is favored with the intention to provide discrete and encapsulated function of distributed agents to allow them to incorporate custom run-time policies in future designs, tailored to the needs of different applications.

3.2 Adaptive and Distributed Application Admission

The envisioned structure of the system is presented in Fig. 4. It consists of i) the incoming application traffic according to differing arrival rate distributions and application characteristics, ii) an input application queue, iii) the DRTRM module responsible for applications' initialisation, resource

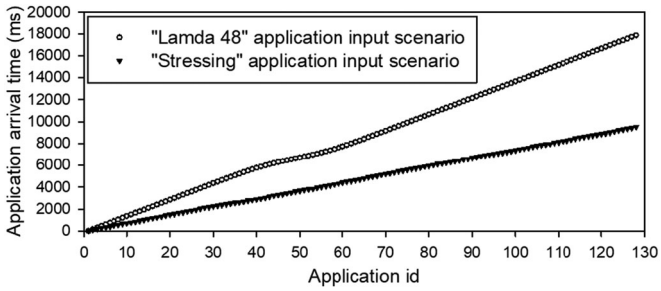


Fig. 5. Arrival times of application trace.

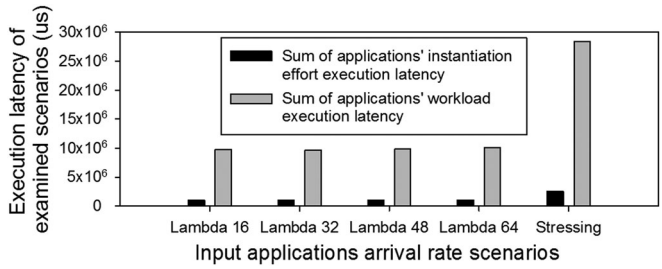


Fig. 6. DRTRM behaviour for different input application arrival rate scenarios (No application-arrival aware features).

management and execution on the many-core system and iv) an admission control module that regulates the dispatching of applications to be mapped onto the targeted platform. The proposed scheme implements a feedback loop approach, in an effort to configure DRTRM at run-time according to the rate of incoming applications and the intensity of system resources utilization by running applications.

We define as $\lambda(t)$ the input rate of new applications on the queue of the processing system and as $\mu(t)$ the total rate of application conclusion and exit from the system. Rate μ_i, that each individual application concludes its operation is

$$\mu_i = f(R_i, W_{comp_i}, W_{comm_i}), \tag{10}$$

ergo a function of its occupying computational resources R_i, the workload W_{comp_i} that the application has to execute and the amount of communication W_{comm_i} which has to perform with other applications on the system (mainly for resource exchange purposes). The correlation between the occupying resources of the application R_i and μ_i is proportional to the speedup of the application

$$\mu_i \propto speedup(R_i) = \frac{Latency(R_i)}{Latency(1)}. \tag{11}$$

The speedup of each application is calculated as the ratio of its estimated finish time when it occupies R_i resources and the corresponding values when it occupies only one resource. In the context of this work, these values are derived experimentally (see Section 3.1.2). In total, for the N_{apps} instantiated on the system at a given time t the total output rate μ is defined as

$$\mu = \sum_{n=1}^{N_{apps}} \mu_i. \tag{12}$$

The aim of the Application-Arrival Aware DRTRM is to service as many incoming applications as possible or in other words minimize the difference between incoming and outgoing application rate

$$min\ Q(t) = \lambda(t) - \mu(t). \tag{13}$$

3.2.1 Effects of the Incoming Applications' Rate on DRTRM

To motivate the necessity for a DRTRM with application-arrival aware characteristics, we quantify the resource management efficacy of DRTRM against different input application scenarios. A "Stressing" scenario was created, where the interval between successive applications is significantly small. Interval values were derived randomly to provide an unbiased input. This is compared against other application arrival scenarios, where the intervals of consecutive incoming applications derive from Poisson distributions with $Lambda$ coefficient values equal to 16, 32, 48 and 64 respectively, as in [15]. Fig. 5 illustrates the application arrival moments of two scenarios, highlighting that applications in the "Stressing" scenario arrive much earlier. All Poisson derived scenarios follow the same trend but differentiate on the moment that there is a peak in the rate of incoming applications.

In Fig. 6, we quantify the impact of $\lambda(t)$ via the aforementioned scenarios, on a DRTRM *without* application-arrival aware characteristics, using the implemented matrix-vector multiplication applications on Intel SCC. Comparison of the "Stressing" to Poisson distribution derived scenarios, shows that total application execution latency exhibits a steep rise of 190 percent in average, despite the fact that the workload intensity of applications is identical in all scenarios. Additionally, the "Sum of applications' instantiation effort execution latency", which is a quantitative indication of the effort spent in order to introduce all applications into the system is averagely 142 percent higher. This indicates that despite the shortage of available computational resources, Initial cores kept on frequently searching cores for the new application, even though it was highly probable that no PE was available and would not be available until one of the running applications finished and freed its resources.

In correlation to Eq. (13), when $\lambda(t)$ is high applications pile up on the system leading to a point when $N_{apps} \approx R_{tot}$, i.e., there are no sufficient resources for admission of new applications. Incoming applications are sent back to the queue and an initialisation process is restarted. This involves resources request from active applications, but since resources are scarce workload execution is interrupted without any gain for any of the involved agents. Inevitably, these interrupts result to an adverse effect on μ, since according to Eq. (10), which suggests that the execution of an application is affected by W_{comm_i}, i.e., its communication with other applications (owned to resource bargaining in this case).

Note that a centralized resource management framework would suffer from similar to the aforementioned inefficiency issues, since the frequent remapping requests due to the high application arrival rates and the lack of distribution of this computational burden, would hinder the run-time adaptation and efficiency of the system.

3.2.2 Proposed Adaptation Scheme

The observed vicious cycle, should be detected and mitigated by a DRTRM able to adapt application admission to

the status of resource demand and supply on the system. In a straightforward manner, such an adaptive behavior could be translated to a policy at the Initial core level, e.g., the repetition frequency of the core searching cycle could be adjusted according to system resource utilization intensity. This adaptation mechanism stumbles upon the nature of purely distributed resource management, suffering from two major drawbacks:

- It requires a central decision to assure effective adaptation of the framework in respect to the incoming application demands. Consequently, this creates a central point of information acquisition, which is intended to be avoided in a distributed framework.
- It suffers from synchronization latency. Even if the designer decides to gather the necessary information and proceed to an adaptation decision, the gathering process followed by a broadcast to the cores could create a significant latency upon the enforcement of the new policy. Eventually, when the new policy is enforced, the state of the system might be different compared to when the gathering process started.

Our proposed distributed mitigation plan is to reduce the burden of pointless application instantiation efforts indirectly, by taking advantage of the resource allocation hierarchy of DRTRM (Fig. 3) in order to slow down the core search efforts of Initial and Manager cores, in case of heavy workload scenarios when available resources are scarce. Our simple yet effective application admission regulation policy, relies on decelerating Controller cores by reducing their operating frequency using Voltage and Frequency Scaling techniques. In contrast to a centralized regulation policy, this requires co-ordination of only the Controller cores, which are very limited in number. As a consequence, the required decision making can be implemented in a distributed manner. In our VFS extended DRTRM scheme, application conclusion rate (Eq. (10)) can be re-written as

$$\mu_i = f(R_i, W_{comp_i}(f(R_i), V_{dd}(R_i)), W_{comm_i}(f(R_i), V_{dd}(R_i))), \quad (14)$$

indicating that both W_{comp_i} and W_{comm_i} are affected by the operational frequency and voltage of the utilized resources R_i of an application. Therefore, for the proposed mitigation plan to be effective, care is taken so that VFS in different PEs is such that

- W_{comm_i} is reduced by avoiding unnecessary communication between new applications under instantiation requesting resources from running ones.
- W_{comp_i} remains unaffected by not applying VFS on resources executing computational workload. This is highly important and implies that worker cores remain at the highest possible frequency, ensuring that no latency overhead is imposed on applications.

Due to the dependencies of resource allocation hierarchy in DRTRM, the deceleration of Controller cores when system utilization is maximum leads to less frequent execution of temporarily redundant tasks of other agents, such as search for free resources. Thus, significantly less negotiation overhead is imposed on running applications, allowing them to proceed with workload execution uninterrupted. In this way, their execution is concluded faster and resources become available to facilitate the needs of new applications.

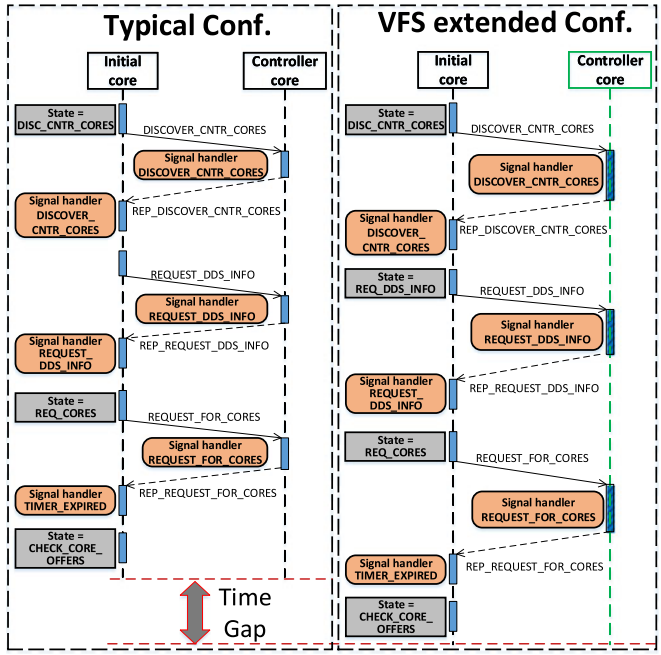


Fig. 7. Time gap in RTRM signal exchange.

Conceptually, the interplay of cores in DRTRM follows/can be modelled as a client-server model, where clients are Initial/Manager cores and the servers are Controllers. Slowing down the servers leads to prolonged waiting times in the client side, leading to increased latency for task completion. In turn, this creates longer control cycles which reduce the pointless application instantiation efforts.

Fig. 7 zooms in the communication between a Controller and an Initial/Manager core, showing in more detail how slowing down the operations of the first eventually results in slowing down the function of the others. The Initial core executes its search for resources, which is highly dependent on information acquired by the Controller core. The left side of Fig. 7 shows the evolution in time of their communication in a typical operating frequency configuration. The right side illustrates the timing for the execution of the same operations after the frequency of the Controller core has been scaled down. Since it operates on lower frequency, more time is required for its incoming request to be served. These requests create blocking points in the execution of the Initial core and indirectly its required time to execute a full resource search cycle is prolonged. This induced time gap slows down the instantiation of one new application and by scaling up to all Initial cores, it leads to reduced instantiation rate for all new applications.

Fig. 8, presents how all cores categories are affected by the VFS enabled DRTRM. Again, we see the interplay of different cores prior to and after VFS. The left side depicts the internal functionality of a Manager core, which distributes workload to its worker. Whenever the worker core finishes its calculations, it informs the Manager in order to decide how to further allocate application workload. If at that moment the Manager is occupied with serving other incoming requests such as a core request by an Initial core, then

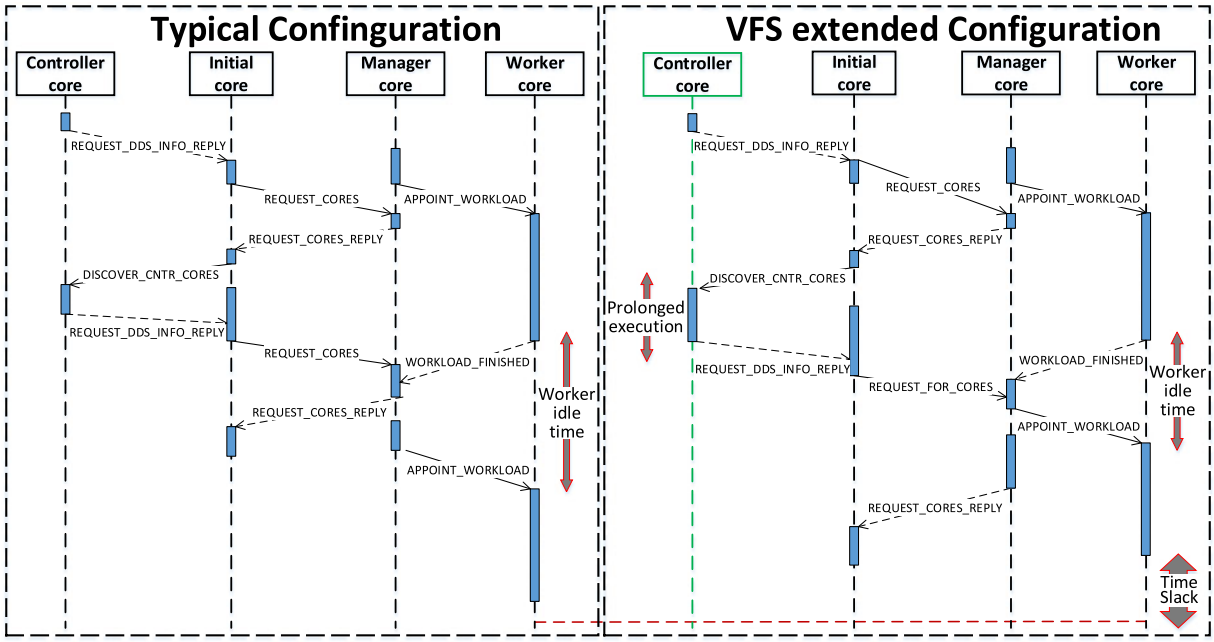


Fig. 8. VFS induced execution slack.

the elapsed time until workload re-distribution is increased and the worker remains idle for more time.

The VFS extended DRTRM, achieves more efficient workers utilization by avoiding the overburdening of Manager cores by incoming resource negotiation requests. As described, Controller cores of reduced operation frequency, stall the operation of Initial cores resulting in the generation of less requests for Managers. Consequently, a Manager core is more responsive when its workers notify their workload completion, which significantly reduces their idle time and eventually leads to lower application execution latency. The reader should note that in an actual stressful scenario, the inefficiencies presented in Fig. 8 are aggravated since the Manager core is flooded by resource requests from numerous Initial cores. This results to a queue of requests, required to be served before re-allocating workload to the workers.

A system with tunable sleeping intervals of the Controller cores would have comparable results to VFS technique with regard to the admission control mechanisms. However, it requires a fine-grained, run-time tuning mechanism, adaptive to the number of running applications, thus introducing synchronization overheads. In addition, the adoption of tunable sleep calls reduces energy efficiency, since there is no guarantee that the sleep duration is sufficient to trigger a low-power system state.

4 Experimental Evaluation

This section, presents the results of our conducted experiments regarding the limits, efficiency, extensions and robustness of our proposed methodology. For consistency purposes, *all experiments have been conducted using the "Stressing" application arrival scenario*, which we consider as the point of interest of our work. Scenarios of lower arrival intensity are sufficiently handled by DRTRM, while scenarios of higher intensity greatly surpass the capabilities of the target platform and are considered unrealistic. Evaluated metrics quantify the performance of the proposed framework as well as its energy savings resulting from the novel use of VFS techniques.

4.1 Intel SCC: Many-Core Cloud-Computing Platform

We utilize Intel Single Chip Cloud Computer [2] as the driver many-core platform of this work, which is a 48-core Network-on-Chip (NoC) platform, based on a mesh interconnection for on-chip data communication. It consists of 24 tiles, each one containing 2 processing cores of x86 instruction set. Tiles are connected through a 2D-mesh and a router inside each tile is responsible for forwarding outgoing packets. Each tile also includes an L2 cache memory shared by the two PEs and a special memory called Message Passing Buffer, which provides fast and reliable data dispatching amongst different cores. The tiles are divided into 6 Voltage Islands (V.I.) and a Voltage Regulator Controller (VRC), provides the ability to regulate the voltage of each island individually. It is also possible to regulate the operating frequency at tile granularity and always within limits dictated by the respective operation voltage.

Each PE runs its own lightweight Linux OS distribution, which significantly improves the programmability of the system. The platform comes with a library called RCCE [28], which offers MPI primitives in order to ease application development. An API is also exposed to safely work with the VFS capabilities of the system, offering the ability to scale the frequency and voltage of each island by dictating a frequency/voltage pair, ranging from <800 MHz, 1.1 V$>$, <533 MHz, 0.8 V$>$ down to <100 MHz, 0.7 V$>$ [29].

The SCC platform incorporates a power metering infrastructure, enabling the reporting of instant voltage and current values drawn by the many-core chip. We developed a custom power metering daemon program, which samples the power metering registers by periodically invoking the *"sccBmc -c status"* command. The gathered values are then numerically integrated over each time interval i, to calculate the dissipated energy: $E = \sum V_i \times I_i \times \Delta t_i$ to acquire the

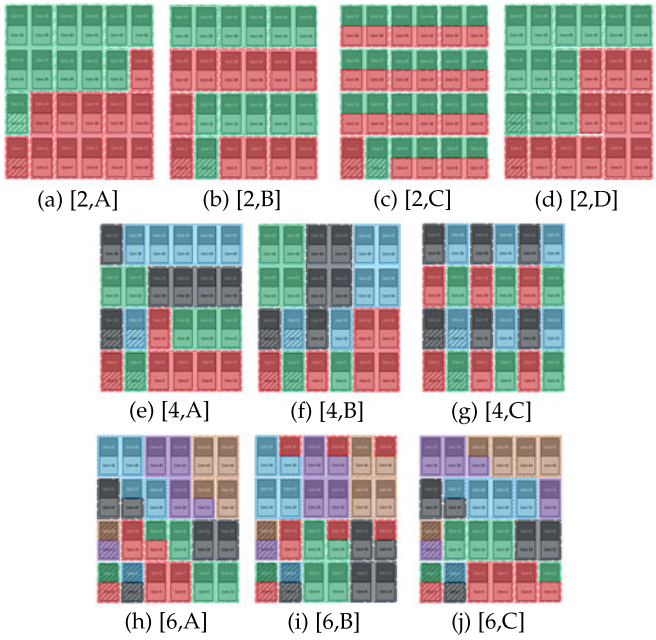


Fig. 9. Examined cluster topologies [#Controller cores, Cluster conf.]. Regions of the same color belong to the same cluster. Controller cores are shaded.

sum of the consumed energy. A similar infrastructure has been used also in [30] to examine the impact of DVFS decisions on the execution of single instances of MPI-based applications mapped onto the SCC.

An instance of DRTRM supervises each PE, operating at the user-space level of the software stack. The required user-defined signals have been developed on top of the RCCE API, which uses the shared MPB memory for inter-core communication. Taking into consideration the platform's VRC architecture of pre-configured voltage islands, we enforce a grouping of the Controller cores onto a specific voltage island. In this way, frequency scaling can be also combined with voltage scaling to further reduce the power consumption of the DRTRM infrastructure.

As a far as *Clusters* are concerned, we examine all configurations of 2, 4 and 6 Controller cores presented in Fig. 9, chosen to include both coarse and fine-grained topologies with only constraint the placement of Controller cores inside V.I. 0, in order to simultaneously regulate their voltage. However, in the general case, DRTRM can support any kind of user-defined topology.

Since the regulation of a Voltage Island affects a group of PEs, the application mapping directives of the VFS enabled DRTRM have been meticulously tweaked to avoid the mapping of a worker core inside a region of reduced frequency in order guarantee that application execution is not hindered. On the contrary, its hierarchical scheme, described in Section 3.1.5, enables the mapping of Manager cores on low power PEs because they do not execute computational workload but only orchestrate intra-application workload distribution and inter-application resource bargaining.

4.2 Performance-Power Gains of Admission Control

The first set of experiments evaluates the efficiency of the proposed admission policy to diminish the congestion created on the many-core system by the "Stressing" application

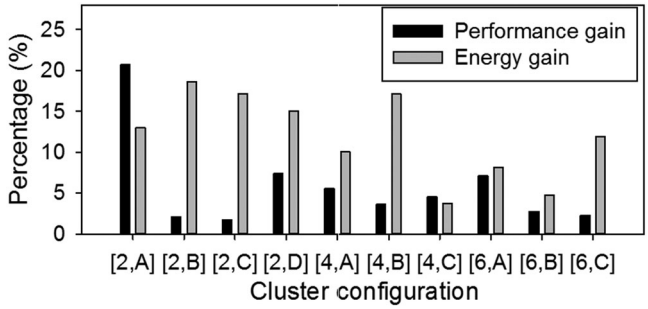


Fig. 10. Performance-energy gains from application admission control in DRTRM.

arrival scenario, using the implemented malleable applications as input. All Controllers are mapped on V.I. 0 of Intel SCC. Fig. 10 presents performance-energy metrics of DRTRM configurations with 2, 4 and 6 Controllers cores. Their operating frequency is dropped from 800 MHz to 533 MHz via a voltage drop from 1.1 V to 0.8 V in V.I. 0.

Results are expressed in normalized gain with respect to the same DRTRM topology without any voltage-frequency scaling of the Controllers. As shown, in all cases performance and energy improvements are reported. The results exhibit a lack of symmetry between the improvement in performance compared to energy. For example, in configuration [2,A] (Fig. 9a) there is a 20 percent improvement in performance accompanied by a 12 percent reduction in the amount of energy, whereas in configuration [4,B] (Fig. 9f) the respective numbers are 3 and 18 percent. This is because the utilized performance metric does not indicate the degree of concurrent execution of different applications on the system. This degree severely affects the required time for each experiment to be completed and thus its requirements in energy. Therefore, on the one hand in configuration [2,A], applications acquired more working cores, their summed execution time was small but they were executed in a more "serialized" way, thus energy gains are smaller. On the other hand, in configuration [4,B] applications are executed in a more concurrent manner meaning that they possess less cores in average. However, the experiment is concluded faster in total, which accounts for the high energy gains.

A delicate point of our design is the correlation of frequency reduction to the resource management efficiency. We emphasize that the proposed application admission policy is very effective in stressful application service requests, i.e., cases when the system load is consuming almost all hardware resources. Furthermore, the reduction of the operating frequency of the Controller cores is a critical design parameter and should not be chosen arbitrarily. Based on that, our next experiments evaluates the behavior of the admission policy extended DRTRM for all operating frequencies provided by Intel SCC.

The experiment, involves only *Cluster* topology [2,A] (Fig. 9a) of 2 Controller cores. For this DRTRM configuration, all the possible values of Controllers operating frequencies are tested. Fig. 11 reports the measured metrics, which expose an interesting performance-energy correlation. Decreasing the operating frequency of Controller cores results in reduced total application execution latency. However, the constant decrease of latency does not always imply more efficient resource allocation. Application initialization on the system

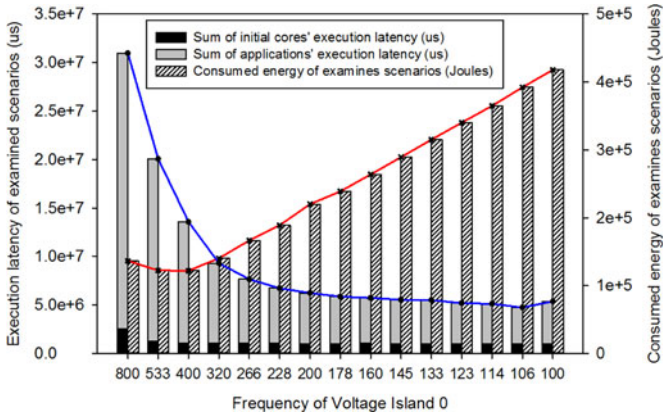


Fig. 11. DRTRM behavior for all freq. dividers of V.I. 0.

is highly related to Controller cores' operation and when their computational capabilities are excessively reduced, this initialization is performed in a slow, almost "serialized" rate. This low rate ensures that when new applications are instantiated, they are offered their maximum number of workers since few other applications occupy resources and thus their execution time is minimized.

Nevertheless, this "serialized" execution profile, results in increased running time for each experiment given that very few applications are executed in parallel. This in turn increases the consumed energy required for the experiment to be concluded. This trend is highlighted on Fig. 11 with red line for energy consumption and blue line for execution latency. The important outcome of this study is that a careless decision of VFS policy can have an adverse effect on DRTRM efficiency. On the contrary, meticulous choice of operating frequencies can give the system designer the freedom to sacrifice concurrent application execution for small energy savings and vice versa.

Symmetrically to investigating DRTRM behaviour under all available Controller cores' operating frequencies, we experiment with increased number of islands of reduced voltage. This gives the opportunity to the proposed DRTRM to utilize VFS in a more common way, i.e., to reduce power consumption of some parts of the system that are not too computationally intensive, such as Manager cores.

A set of experiments were performed, with more than one V.I. of Intel SCC operating on reduced voltage. Our design constraint is that no worker core will be mapped on these islands, in order not to stall workload execution. Results are presented in Fig. 12 including metrics about latency and energy consumption of the various configurations.

We observe that, having two islands of reduced voltage leads to high gains both in execution latency and consumed energy of the system. Further increase to the number of reduced voltage islands leads to prolonged application execution latency. This is attributed mainly to the fact that a large number of cores cannot be utilized to execute workload (worker cores) and thus applications have few resources. This increased latency, results to prolonged activity on the system and consequent increase in its consumed energy. In conclusion, this analysis shows that it is worthwhile to further investigate DRTRM configurations of up to 2 reduced voltage islands aiming at fine tuning the rest of DRTRM design options.

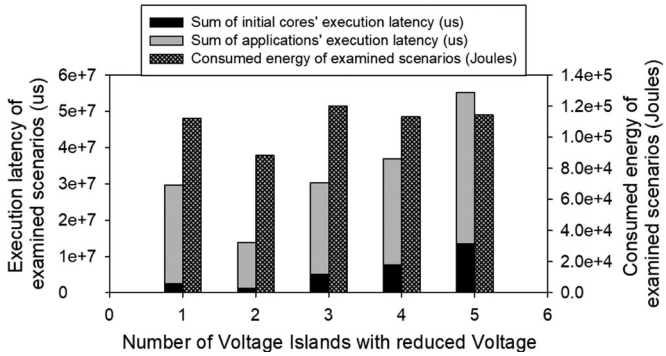


Fig. 12. DRTRM behavior for increasing reduced V.I.s.

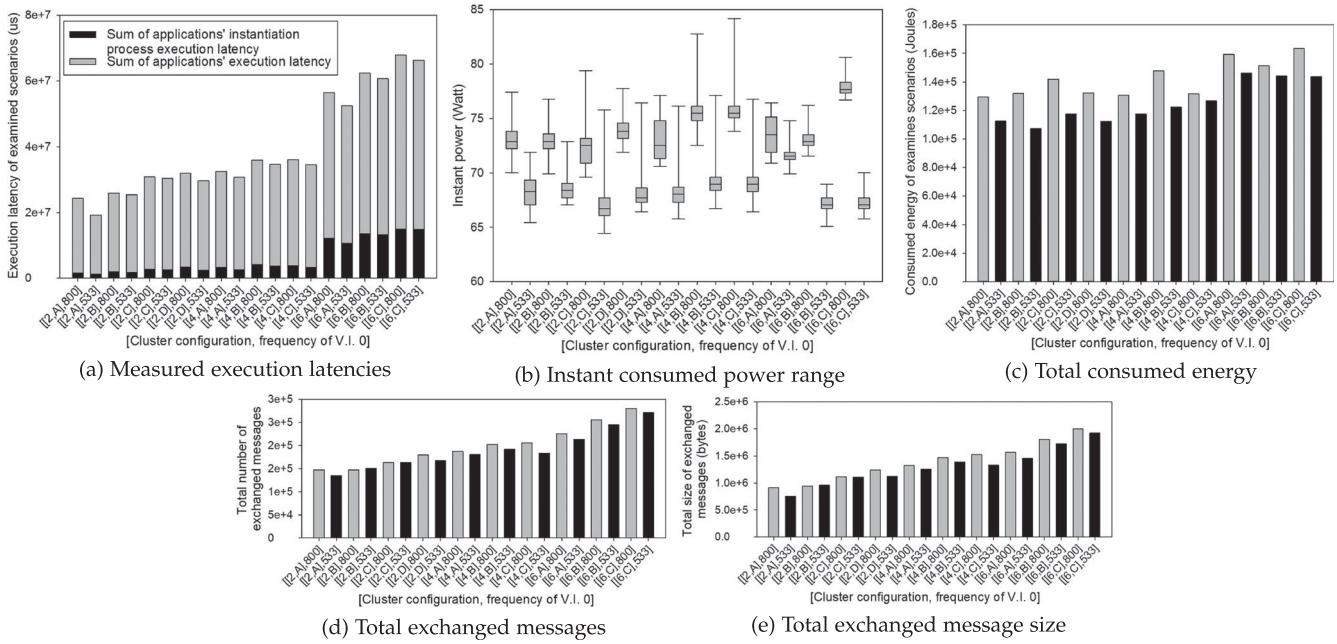
4.3 Exploratory Analysis of the DRTRM Parameters

4.3.1 *Configurations of One Island with Reduced Voltage*

In order to evaluate the combined effects of the design parameters of DRTRM and our proposed methodology, we perform a large exploration campaign over the different *Cluster* topologies. Fig. 13 summarizes the results in terms of (a) the total execution latency for all applications to be initiated (Sum of Initial cores execution latency) and executed, (b) distribution of instant power consumption of Intel SCC (c) total consumed energy, (d) total number of exchanged messages, (f) size of exchanged messages. In all diagrams, the *X*-axis is a tuple of the examined *Cluster* configuration (Fig. 9) and the frequency of Controller cores. For example, tuple [[2,A],800] means *Cluster* configuration [2,A] (Fig. 9a) with 2 Controller cores operating at 800 MHz. Tuple [[2, A],533] is about the same *Cluster* topology with Controller cores at 533 MHz. Frequency of 800 MHz implies that no admission control is performed, while frequency of 533 MHz implies that admission control is active and worker cores are mapped outside of the island of reduced voltage.

Regarding system performance, Fig. 13a validates that a high number of Controller cores in SCC platform results in increased latency for both applications' instantiation and execution. As far as the proposed admission control policy is concerned, it is shown that it results to lower latency in all cases, enabling performance optimization of 6 percent, in average. The combined energy consumption gains rise up to 12 percent (Fig. 13c). Additionally, we observe in Fig. 13b that the proposed admission control scheme leads to better power distribution both in terms of robustness (the 25- and 75-quantiles in cases of 533 MHz are consistently closer than in the case of 800 MHz) and peak power, where gains of averagely 6 percent are reported. Power distribution for cluster topologies with increased number of Controller cores ([6,B] and [6,C]), exhibits a more robust trace given that the incoming workload is distributed in an even manner across system resources due to the small size of *Clusters*.

As shown in Fig 13d, increased system performance is correlated to the number of exchanged messages. For every *Cluster* topology, the proposed admission control policy results in reduced number of exchanged messages, which validates its intended goal of regulating the intensity of core search operations. Regarding the total size of exchanged messages (Fig. 13e), the trend is the same as in their total number, but the actual values are not proportional since the



(a) Measured execution latencies (b) Instant consumed power range (c) Total consumed energy

(d) Total exchanged messages (e) Total exchanged message size

Fig. 13. Measured values for all configurations (Cluster topology & operation frequency of Voltage Island 0).

size of each message varies according to its type. For example, an offer for cores from one manager to another is a few bytes long since it involves the ids of the offered cores, while the corresponding reply message is only one byte long to indicate the acceptance/rejection of the offer.

To summarize, our experiments provide an experimentally derived proof that the proposed application arrival aware DRTRM is effective in mitigating the "Stressful" system state regardless of the chosen Cluster configuration. Nevertheless, careful choice of the Cluster configuration can maximize the achieved gains, which are not limited to performance metrics but include power and energy consumption as well as communication traffic on the system.

The experiment is repeated for configurations of 2 Controller cores, using the malleable application model presented in Section 3.1.1, to validate the efficiency of our proposed methodology using a more diverse mix of applications. Each application is characterized by its parallelism variance σ and average parallelism A, which are provided as input to DRTRM. The workload values W are maintained identical with the ones used for the Matrix Multiplication application and the experiment is executed on Intel SCC. One workload round equals to a time delay scaled according to the Speedup of the application, which is calculated with respect to its characteristics and the number of its worker cores. To create the workload mix, different values of application parameters have been randomly chosen, ranging from 0.01 to 100 for σ and from 2 to 16 for A. Applications with low σ exhibit steep scaling with high speedup for the majority of A values. Applications with high σ have smooth scaling characteristics but do not achieve as great speedup, even for high A values.

The presented results in Fig. 14, validate the efficiency of our proposed application arrival aware DRTRM, which manages to reduce the total execution latency of applications by 43.8 percent and the required latency for application admission by 472 percent in average, in comparison to the original DRTRM. The higher gains are attributed to the mixed scaling characteristics of applications, enabling the admission control policy to provide many worker cores to high scaling applications and thus conclude their operation much faster. This in turn leads to faster release of their occupied resources, thus allowing the queued incoming applications to locate working cores with highly reduced effort, due to the lack of congestion on the many-core system.

4.3.2 *Configurations of Two Islands with Reduced Voltage*

The experiments illustrated in Fig. 15 elaborate on the efficiency of DRTRM in a system with two islands of reduced voltage, inside which Manager cores are mapped. Performance and energy metrics are provided, for configurations of 2 Controller cores given that they fared better in the previous experiments. The X-axis of the plots includes tuples of the examined *Cluster* configuration (Fig. 9) and the ids of the islands of reduced voltage. For example, tuple [[2,A],R. V.I.(0.1)] implies *Cluster* configuration [2,A] (Fig. 9a) and Voltage Islands 0 and 1 operating on 0.8V at 533 MHz.

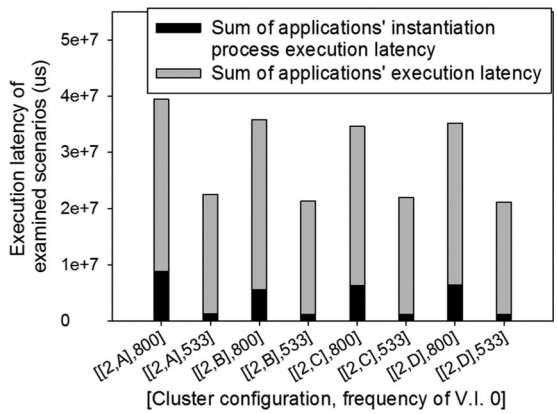
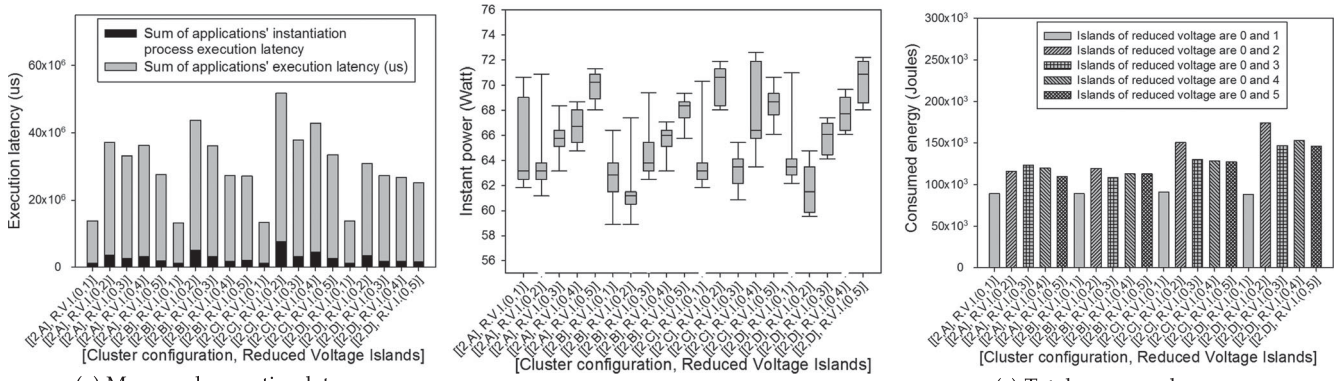


Fig. 14. Comparison of configurations for model based malleable applications (Cluster topology & frequency of V.I. 0).



(a) Measured execution latency (b) Instant consumed power range (c) Total consumed energy

Fig. 15. Measured values for all configurations (Clusters of two Controller cores + Voltage Islands with reduced voltage).

The overall trend in results is that all configurations with V.I. 0 and 1 diminished in voltage, lead to significantly better system performance compared to all other combinations of reduced V.I.s of the same *Cluster* configuration. This is because Controller and Manager cores are mapped in close proximity and thus their communication is faster and more efficient. Additionally, V.I. 1 is the access point for communication of Intel SCC with external non-volatile storage, where log files of the incoming applications are stored. As a consequence, I/O operations are sped up leading to highly reduced execution time for Manager cores. In turn, this leads to less energy consumption, compared to all other configurations of the same *Cluster* topology.

The rest of the combinations of V.I.s do not result in enhanced system performance and significant gains in consumed energy, if any. Amongst them, the one with reduced voltage in islands 0 and 5 fares better, which can be attributed to less communication traffic in the center of the platform, which uses XY-routing in its mesh. This reduced congestion in the center of the platform enables better communication between Controllers and the rest of the cores and this in turn leads to small performance improvements.

The best *Cluster* configuration in terms of performance is [2,C] (Fig. 9c), which slightly prevails over all others. Interestingly this topology is highly fragmented in the sense that *Clusters* include small areas spread throughout the platform. An example contrary topology is that of configuration [2,D] (Fig 9d), where *Clusters* are two big continuous areas. The difference in performance can be explained on two grounds. First, the fragmentation of *Clusters*, leads to highly dispersed application mapping over the platform. Given that the examined workload does not imply communication among worker nodes, this disperse mapping reduces the probability of their memory operations being congested, especially for accesses to off-chip shared DRAM. Secondarily, Initial cores are randomly distributed at run-time in a round robin fashion inside different *Clusters*. As a result, high *Cluster* fragmentation increases the probability of more widespread Initial core assignment and thus more evenly balanced core allocation in applications. In conclusion, the amalgamation of common VFS techniques in our application aware DRTRM, can optimize system performance and lead to reduced energy consumption. However, this can only be achieved by careful choice of DRTRM parameters both at design-time (e.g., Cluster configuration) and run-time (e.g., VFS topology and Managers mapping policy).

4.3.3 *Robustness against Workload Scalability*

In this section we evaluate the robustness of the proposed DRTRM scheme against scaled workloads, using the configuration [2,A] at 533 MHz. In each experiment the number of incoming applications and the intensity of their workload requirements varies. The workload W values were generated using a random number generation function based on Poisson distribution. Four different levels of workload intensity were created using values 16, 32, 48, 64 as the mean value of the random generator. The workload escalation of these four levels is provided in the legend of Figs. 16 and 17. The variation in workload was combined with an ascending number of incoming applications ranging from 16 to 128.

In the first experiment, the implemented malleable application (Section 3.1.2) is utilized and Fig. 16 presents the total execution latency of each examined input workload combination. Results show a close to linear scaling in latency for ascending number of incoming applications and workload intensity levels. In addition, in all cases the deviation from the respective mean value is very small and no unexpected behavior is observed, e.g., spikes in the execution latency. It is also important to take into account the noise (expressed through variations in latency) injected in the measured values due to the layered software stack of each SCC core (DRTRM instance - Linux OS - SCC drivers). The above observations qualify the proposed DRTRM scheme as robust in terms of the required average execution latency with respect to the examined workload related parameters.

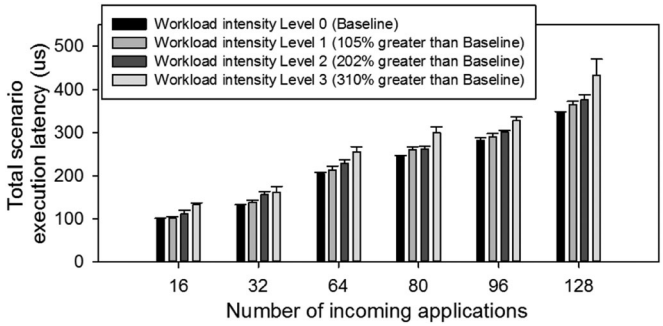


Fig. 16. Different examined Matrix Multiplication application workloads (Mean value, standard deviation format).

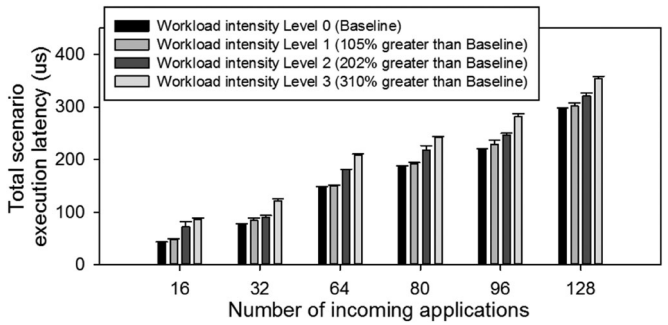


Fig. 17. Different examined model based application workloads (Mean value, standard deviation format).

The experiment is repeated for the mix of applications created using the malleable application model as described in Section 4.3.1. While the characteristics of the input applications differ, their workload requirements are identical to the ones of Fig. 16. Fig. 17 illustrates the required latency for the execution of the different scenarios of the model based workload mix. We observe that the framework maintains its robustness in handling applications of increasing workload requirements, despite of the higher diversity of their scaling characteristics. Compared to the respective results for the implemented application (Fig. 16), the deviation of the measured values is much smaller since the lack of workload computations leads to less latency variations and more deterministic interplay between the distributed agents during the execution of each input scenario.

4.4 Comparative Evaluation of Different RTRM Schemes

Finally, we present a combined comparative analysis of the different types of resource management schemes presented throughout this work. More specifically, we compare i) a centralized RTRM, where one agent monitors the entire system and aids inter-application negotiations for resources, ii) a Distributed RTRM without application aware admission control and iii) the proposed VFS extended DRTRM in versions of one or two reduced voltage islands. Examined scenarios involve the execution of 128 application arriving at the rate dictated by the "Stressful" scenario. Results are presented in Fig. 18 in terms of total application execution latency and system-wide consumed energy.

The worst RTRM regarding application execution latency is the centralized one, since in SCC a single core is not competent enough to handle the high amount of resource allocation requests generated by the input rate of incoming applications. This inefficiency is translated to elevated consumed energy of the centralized scheme. Increasing the number of RTRM related managerial agents as in the case of no VFS extended DRTRM results in much better resource management performance. However, the arrival rate of applications is still not taken into account, which leaves space for further gains when VFS extended version of DRTRM is applied. Having balanced the resource utilization under this stressful scenario, more gains are acquired in the VFS extended DRTRM with two reduced voltage islands, which maps Manager cores efficiently, taking advantage of their execution profile. In overall, when comparing the centralized approach to the best of the VFS extended one, gains of

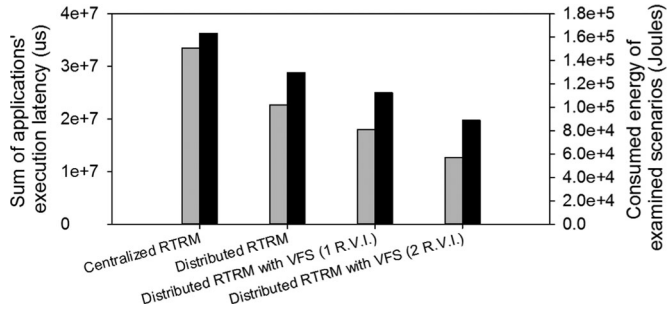


Fig. 18. Comparison of the different RTRM schemes.

62 percent in total application execution latency and 45 percent in consumed energy are exhibited.

5 Conclusion

In this paper we presented an Application-Arrival Aware Distributed Run-Time resource management framework focused on the execution of applications with malleable features. We analyzed the structure and internal mechanisms of the developed framework and proposed an operating voltage and frequency scaling strategy, that regulates application admission without degenerating its distributed nature. We showed that the efficiency of the distributed resource management is highly dependent on both the applications' arrival patterns and management policies/mechanisms. Extensive experiments on Intel SCC many-core platform showed that performance, power and energy gains are delivered in comparison to either centralized or distributed run-time resource management schemes.

Acknowledgments

This work has been partially supported by the E.C. programs AEGLE under H2020 Grant Agreement No: 644906 and VINEYARD under H2020 Grant Agreement No: 687628.

References

[1] M. Ferdman, et al., "Clearing the clouds: A study of emerging scale-out workloads on modern hardware," in *Proc. 17th Int. Conf. Archit. Support Program. Languages Operating Syst.*, 2012, pp. 37–48.

[2] J. Howard, et al., "A 48-core IA-32 message-passing processor with DVFS in 45 nm CMOS," in *Proc. IEEE Int. Solid-State Circuits Conf. Digest Tech. Papers*, 2010, pp. 108–109.

[3] G. Chrysos, "Intel® xeon phi™ coprocessor-the architecture," Intel Whitepaper, 2014.

[4] S. Bell, et al., "Tile64-processor: A 64-core SOC with mesh interconnect," in *Proc. Digest Tech. Paper IEEE Int. Solid-State Circuits Conf*, 2008, pp. 88–598.

[5] R. De Lemos, et al., "Software engineering for self-adaptive systems: A second research roadmap," in *Software Engineering for Self-Adaptive Systems II*. Berlin, Germany: Springer, 2013, pp. 1–32.

[6] J. Jahn and J. Henkel, "Pipelets: self-organizing software pipelines for many-core architectures, " in, *Proceedings of the Conference on Design, Automation and Test in Europe*. San Jose, CA, USA: EDA Consortium, 2013.

[7] S. Kobbe, L. Bauer, D. Lohmann, W. Schröder-Preikschat, and J. Henkel, "DistRM: distributed resource management for on-chip many-core systems," in *Proc. 7th IEEE/ACM/IFIP Int. Conf. Hardw./Softw. Codesign Syst. Synthesis*, 2011, pp. 119–128.

[8] I. Anagnostopoulos, V. Tsoutsouras, A. Bartzas, and D. Soudris, "Distributed run-time resource management for malleable applications on many-core platforms," in *Proc. 50th Annu. Des. Autom. Conf.*, 2013, Art. no. 168.

[9] A. Schüpbach, et al., "Embracing diversity in the barrelfish manycore operating system," in *Proc. Workshop Managed Many-Core Syst.*, 2008, Art. no. 27.

[10] J. A. Colmenares, et al., "Tessellation: Refactoring the OS around explicit resource containers with continuous adaptation," in *Proc. 50th Annu. Des. Autom. Conf.*, 2013, Art. no. 76.
[11] D. Wentzlaff, et al., "A unified operating system for clouds and manycore: FOS," Computer Science and Art. Intelligence Lab, Massachusetts Institute of Technology, Cambridge, MA, USA: Tech. Rep. MIT-CSAIL-TR-2009–059, 2009.
[12] A. Barbalace, B. Ravindran, and D. Katz, "Popcorn: a replicated-kernel os based on linux," in *Proc. Linux Symp.*, 2014.
[13] S. T. Maguluri, R. Srikant, and L. Ying, "Heavy traffic optimal resource allocation algorithms for cloud computing clusters," *Perform. Eval.*, vol. 81, pp. 20–39, 2014.
[14] W. Wang, K. Zhu, L. Ying, J. Tan, and L. Zhang, "Maptask scheduling in MapReduce with data locality: Throughput and heavy-traffic optimality," *IEEE/ACM Trans. Netw.*, vol. 24, no. 1, pp. 190–203, Feb. 2016.
[15] B. Raghunathan and S. Garg, "Job arrival rate aware scheduling for asymmetric multi-core servers in the dark silicon ERA," in *Proc. Int. Conf. Hardw./Softw. Codesign Syst. Synthesis*, 2014, pp. 14:1–14:9.
[16] M. A. Al Faruque, R. Krist, and J. Henkel, "ADAM: run-time agent-based distributed application mapping for on-chip communication," in *Proc. 45th ACM/IEEE Des. Autom. Conf.*, 2008, pp. 760–765.
[17] I. Anagnostopoulos, A. Bartzas, G. Kathareios, and D. Soudris, "A divide and conquer based distributed run-time mapping methodology for many-core platforms, " in, *Proceedings of the Conference on Design, Automation and Test in Europe*. San Jose, CA, USA: EDA Consortium, 2012.
[18] W. Zheng and R. Sakellariou, "Budget-deadline constrained workflow planning for admission control," *J. Grid Comput.*, vol. 11, no. 4, pp. 633–651, 2013.
[19] P. Dziurzanski, A. K. Singh, and L. S. Indrusiak, "Feedback-based admission control for hard real-time task allocation under dynamic workload on many-core systems," in *International Conference on Architecture of Computing Systems*. Berlin, Germany: Springer, 2016, pp. 157–169.
[20] J. Lee, C. Nicopoulos, H. G. Lee, S. Panth, S. K. Lim, and J. Kim, "IsoNet: Hardware-based job queue management for many-core architectures," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 21, no. 6, pp. 1080–1093, Jun. 2013.
[21] G. Mariani, G. Palermo, V. Zaccaria, and C. Silvano, "Arte: An application-specific run-time management framework for multi-cores based on queuing models," *Parallel Comput.*, vol. 39, no. 9, pp. 504–519, 2013.
[22] N. Kapadia and S. Pasricha, "A runtime framework for robust application scheduling with adaptive parallelism in the dark-silicon era," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 25, no. 2, pp. 534–546, Feb. 2017.
[23] T. Ebi, M. Faruque, and J. Henkel, "Tape: Thermal-aware agent-based power econom multi/many-core architectures," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.-Dig. Tech. Papers*, 2009, pp. 302–309.
[24] Y. Ge, P. Malani, and Q. Qiu, "Distributed task migration for thermal management in many-core systems," in *Proc. 47th Des. Autom. Conf.*, 2010, pp. 579–584.
[25] M. Shafique and J. Henkel, "Agent-based distributed power management for kilo-core processors," in *Proc. Int. Conf. Comput.-Aided Des.*, 2013, pp. 153–160.
[26] V. K. Vavilapalli, et al., "Apache hadoop YARN: Yet another resource negotiator," in *Proc. 4th Annu. Symp. Cloud Comput*, 2013, Art. no. 5.
[27] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel, "Mapping on multi/many-core systems: survey of current and emerging trends," in *Proc. 50th Annu. Des. Autom. Conf.*, 2013, Art. no. 1.
[28] R. F. van der Wijngaart, T. G. Mattson, and W. Haas, "Light-weight communications on Intel's single-chip cloud computer processor," *ACM SIGOPS Operating Syst. Rev.*, vol. 45, no. 1, pp. 73–83, 2011.
[29] P. Gschwandtner, T. Fahringer, and R. Prodan, "Performance analysis and benchmarking of the intel scc," *Cluster Comput. (CLUSTER), 2011 IEEE Int. Conf.*, pp. 139–149, 2011.
[30] A. Bartolini, M. Sadri, J. Furst, A. K. Coskun, and L. Benini, "Quantifying the impact of frequency scaling on the energy efficiency of the single-chip cloud computer," in *Proc. Des. Autom. Test Eur. Conf. Exhibition*, 2012, pp. 181–186.

Vasileios Tsoutsouras received the diploma degree in electrical and computer engineering from the National Technical University of Athens, Greece, in 2013, and is currently working toward the PhD degree with the National Technical University of Athens, Greece. His main research interests include resource management on many-core architectures and Internet of Things, embedded medical devices, and HW/SW co-design. He has published more than 10 technical and research papers in scientific books and international conferences. He is a student member of the IEEE.

Sotirios Xydis received the diploma and PhD degrees in electrical and computer engineering from the National Technical University of Athens, Athens, Greece, in 2005 and 2011, respectively. His research interests include design space exploration for system level and datapath synthesis, multi-/many-core and re-configurable architectures, design and optimization of arithmetic VLSI circuits and power/thermal aware design. He has published more than 80 technical and research papers in scientific books, international journals, and conferences. He is a member of the IEEE.

Dimitrios Soudris received the diploma and PhD degrees in electrical and computer engineering from the University of Patras, Greece, in 1987 and 1992, respectively. Since 1995 and for 13 years, he served as a professor in the Department of Electrical and Computer Engineering, Democritus University of Thrace, Greece. He is currently an associate professor in the School of Electrical and Computer Engineering, National Technical University of Athens, Greece. His research focuses on embedded systems design, low power VLSI design, and reconfigurable architectures. He has published more than 380 papers and is the coauthor/coeditor of six Kluwer/Springer books. He is a member of the IEEE.