A Survey and Comparative Study of Hard and Soft Real-Time Dynamic Resource Allocation Strategies for Multi-/Many-Core Systems

AMIT KUMAR SINGH, University of York and University of Southampton
PIOTR DZIURZANSKI, University of York and Staffordshire University
HASHAN ROSHANTHA MENDIS and LEANDRO SOARES INDRUSIAK, University of York

Multi-/many-core systems are envisioned to satisfy the ever-increasing performance requirements of complex applications in various domains such as embedded and high-performance computing. Such systems need to cater to increasingly dynamic workloads, requiring efficient dynamic resource allocation strategies to satisfy hard or soft real-time constraints. This article provides an extensive survey of hard and soft real-time dynamic resource allocation strategies proposed since the mid-1990s and highlights the emerging trends for multi-/many-core systems. The survey covers a taxonomy of the resource allocation strategies and considers their various optimization objectives, which have been used to provide comprehensive comparison. The strategies employ various principles, such as market and biological concepts, to perform the optimizations. The trend followed by the resource allocation strategies, open research challenges, and likely emerging research directions have also been provided.

CCS Concepts: • **Computer systems organization** → **Real-time systems**; **Embedded systems**; **Architectures**

Additional Key Words and Phrases: Many-core systems, hard real-time, soft real-time, design-space exploration, resource allocation, performance, energy consumption

ACM Reference Format:
Amit Kumar Singh, Piotr Dziurzanski, Hashan Roshantha Mendis, and Leandro Soares Indrusiak. 2017. A survey and comparative study of hard and soft real-time dynamic resource allocation strategies for multi-/many-core systems. ACM Comput. Surv. 50, 2, Article 24 (April 2017), 40 pages.
DOI: http://dx.doi.org/10.1145/3057267

1. INTRODUCTION

A paradigm shift to the adoption of multi-/many-core systems can be observed in various domains such as embedded and high-performance computing (HPC). The reason behind such adoption lies in the fact that the performance requirements of applications cannot be satisfied by simply increasing the frequency of a single-core processor, which leads to high power and heat dissipation. In multi-/many-core systems, chip manufactures are trying to overcome these bottlenecks by integrating multiple cores operating at low frequencies, where the cores can cohesively communicate with each other [Jerraya et al. 2005; Borkar 2007]. These systems provide increased parallelism that motivates us to

This work has been funded in part by the EU FP7 project DreamCloud (project number 611411).
Authors' addresses: A. K. Singh, School of Electronics and Computer Science, University of Southampton, Building 59, Level 4, Room 4237, Southampton SO17 1BJ United Kingdom; email: a.k.singh@soton.ac.uk; P. Dziurzanski, School of Computing and Digital Tech, Staffordshire University, Mellor Building, College Road, Stoke-on-Trent ST4 2DE United Kingdom; email: Piotr.Dziurzanski@staffs.ac.uk; H. R. Mendis and L. S. Indrusiak, Department of Computer Science, University of York, Deramore Ln, Heslington, York YO10 5GH United Kingdom; emails: {hrm506, leandro.indrusiak}@york.ac.uk.

DOI: http://dx.doi.org/10.1145/3057267

partition applications into many small tasks and allocate them onto different cores in order to perform parallel executions towards satisfying the increased performance requirements [Manimaran et al. 1998].

The many-core processors have been designed by several chip manufactures. Some examples include Intel's Teraflop 80-core processor [Vangal et al. 2007], Tilera's TILE-Gx family 100-core processor [TILE-Gx 2009], AMD's Opteron 16-core processor [AMD 2011], and Kalray's multi purpose processor array (MPPA) 256-core processor [De Dinechin et al. 2014]. Recently, a joint effort between IBM and UCDavis has produced the KiloCore 1000-core chip [Bohnenstiehl et al. 2016]. The large number of cores are usually connected by an on-chip interconnection network [Benini and De Micheli 2002; Worm et al. 2002; Bjerregaard and Mahadevan 2006]. These many-core processors are designed to be exploited in various application domains towards realizing different systems, referred to as many-core systems. Additionally, different types of cores have been integrated to exploit their distinct features towards meeting the functional and non-functional requirements [Smit et al. 2004]. The integration of different types of cores leads to the development of heterogeneous multi-/many-core systems that become a formidable computing alternative where applications witness large improvements over their homogeneous (consisting of identical cores) counterparts. Further, the technological advancements will enable integration of higher number of cores in the same chip.

1.1. Resource Allocation for Multi-/Many-Core Systems

A resource allocation (mapping) process defines assignment and ordering of the tasks and their communications onto resources of a multi-/many-core system[1] in view of some optimization criteria such as compute performance and energy consumption. The many-core systems (contain a relatively large number of cores) usually extend multi-core systems that contain small numbers of cores, and thus resource allocation techniques for these systems can be interchangeably employed if they have architectural similarities [Woo and Lee 2008]. However, to better exploit the many-core resources, the techniques tailored for multi-cores might need some modifications due to different interconnects. For these systems, usually, the applications need to be partitioned (parallelized) into multiple tasks that can be executed concurrently on different cores. Such partitioning is referred to as functional partitioning and can be furnished with the help of state-of-the-art application parallelization tools, for example, MPSoC Application Programming Studio [Ceng et al. 2008], the MNEMEE project tool-chain [Mallik et al. 2011], and/or manual analysis. This procedure requires detailed application knowledge and involves finding the tasks, adding synchronization and inter-task communication in the tasks, management of the memory hierarchy communication and checking of the parallelized code (tasks) to ensure for correct functionality [Martin 2006]. In case the multi-/many-core system is heterogeneous, that is, contains different types of cores, a task *binding* process that specifies the core types for possible allocation of the task along with the the cost of allocation on each core type is required [Smit et al. 2004]. To compute the allocation cost, the binding process analyses the implementation costs (e.g., performance, power, and resource utilization) of each task on different supported core types, such as general-purpose processor (GPP), digital signal processor (DSP), and coarse-grained re-configurable hardware.

An example *resource allocation* along with the application *parallelization* is shown in Figure 1. The *parallelization* procedure partitions a *sequential application* described in a high-level programming language (e.g., C/C++) into various connected tasks. The connections between the tasks reflect the dependencies in the corresponding sequential application. The example partitioned application is shown as *Application Task Graph*

[1]By multi-/many-core system, we mean a system on a multi-/many-core chip or several multi-/many-core chips connected with each other, where each chip contains a set of connected cores.

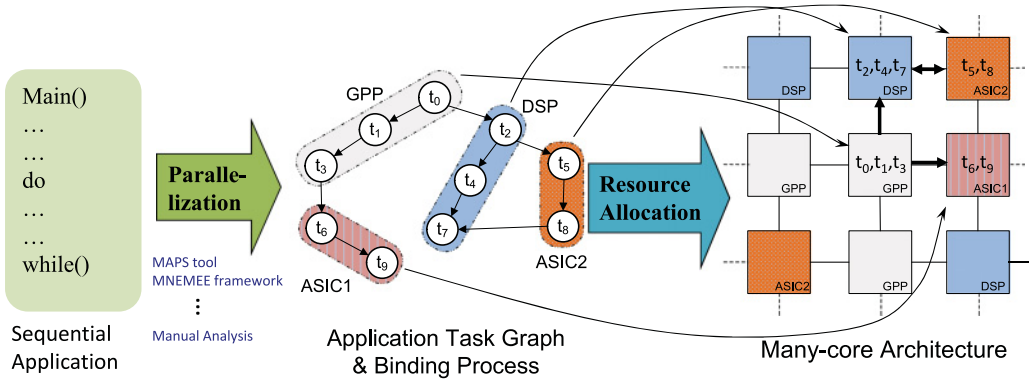

Fig. 1. Resource allocation on a many-core system.

that consists of 10 tasks (t_0,t_1, . . . , t_9). For the *Application Task Graph*, the *binding* process has specified different core types for various tasks based on the implementation costs, for example, GPP for tasks t_0, t_1, and t_3. The *resource allocation* process assigns tasks and their communications on part of a heterogeneous many-core system. The communicating tasks are mapped (allocated) on the same core or close to each other to optimize for the communication delay and energy. Differently from functional partitioning, data partitioning can also be employed to perform parallel processing of the data by the same function, for example, applying a filter function on partitioned segments of an image.

The resource allocation process is carried out either at design time (statically) or runtime (dynamically). Most of the existing literature for resource allocation falls under static resource allocation (e.g., Murali et al. [2006], Hu and Marculescu [2003], Javaid and Parameswaran [2009], Marcon et al. [2008], Thiele et al. [2011], Meyer et al. [2010], Choi et al. [2012], and Castrillon et al. [2012]). However, they cannot handle dynamic workloads and changing environments, for example, adding a new application into the system at runtime.

Dynamic resource allocation approaches can handle the aforementioned issues, as the assignment of tasks and their communications on the multi-/many-core system resources is done at runtime. In addition, they offer several other advantages, such as adaptability to the available resources over time (in case performance requirements of a running application are changed or the current allocation is not sufficiently close to optimal) and the ability to avoid the defective parts of multi-/many-core systems and to enable foreseeable upgrades [Singh et al. 2013b]. The allocation has been handled either by performing all the processing at runtime, that is, on-the-fly processing, or by using previously analyzed results [Singh et al. 2013b; Indrusiak et al. 2016]. The results have been analyzed by employing efficient design-time design space exploration (DSE) strategies to encounter different runtime scenarios [Xue et al. 2006; Zamora et al. 2007; Stuijk et al. 2010; Schranzhofer et al. 2010; Mariani et al. 2010; Ykman-Couvreur et al. 2011; Piscitelli and Pimentel 2012; Singh et al. 2013a]. For *on-the-fly* processing, efficient heuristics have been devised to assign new arriving tasks on the system resources [Moreira et al. 2007; Nollet et al. 2008; Schranzhofer et al. 2009; Carvalho et al. 2010; Wang et al. 2011; Kaushik et al. 2011; Chen et al. 2012]. These heuristics do not use any prior analysis results and thus provide rather low-quality resource allocations. However, since they do not use precomputed platform-specific analysis results, they cope well in allocating unknown applications on any platform. In contrast, better-quality resource allocation is achieved by using previously analyzed

results, but the applications to be supported on a platform should be known in advance to perform analysis.

Real-time dynamic resource allocation is desired in systems where performance (timing) constraints need to be satisfied to fulfill safe system operations (e.g., in automotive engine management, operating medical equipment, and flight control software) and end-user demands (frame rate in video processing). This necessitates the development of efficient resource allocation strategies that take an application model, a multi-/many-core platform model, the constraints (e.g., timing and power), and the performance model of inter-process communication (e.g., execution time and energy consumption) and estimate of the worst-case execution time (WCET) of the process implementations on different cores (e.g., GPP, DSP, application-specific integrated circuit (ASIC)) as input and provide real-time performance guaranteeing resource allocations or optimized resource allocations. A significant amount of research for real-time dynamic resource allocation on single-core system was done in the 1980s and 1990s [Audsley et al. 1995]. For multi-/many-core systems, it started at the same time, but even more attention was paid after the release of multi-/many-core processors, such as the dual-core POWER4 processor by IBM in 2001. Despite the fact that several articles have been published and significant progress has been made for real-time allocation on multi-/many-core systems, there still remain many open questions and research challenges.

1.2. Dynamic Resource Allocation Problem and Challenges

It has been well proven that resource allocation is one of the most complex problems in large many-core and distributed systems, and, in general, it is considered NP-hard [Garey and Johnson 1979]. It has also been identified as one of the most urgent problems to be solved for implementing multi-/many-core-based embedded systems [Marculescu et al. 2009; Marwedel et al. 2011]. A well-tuned search algorithm may need to evaluate hundreds of thousands of distinct allocations before it finds one solution that meets the systems performance requirements [Mariani et al. 2010; Piscitelli and Pimentel 2012]. Since such evaluation is expected to take a long time, maybe hours to days, it cannot be applied to find the solution quickly, which is desired within the context of dynamic resource allocation. Further, it is difficult and challenging to identify the ways that can help to *achieve accurate status of resources* during runtime. This status may be utilization or memory usage of different cores into the system. An inaccurate status of resources may result in an allocation that might not be efficient at runtime.

It is also challenging to satisfy performance requirements of each application when various combinations of simultaneously active applications, referred to as use-cases, need to be supported into the system at runtime. For each use-case, since optimal solution cannot be explored at runtime due to limited computation power and evaluation time, it needs to be explored by advanced design-time DSE approaches and then to be used at runtime. However, an explosion in the number of use-cases has been witnessed with increasing number of applications, for example, 2^n use-cases for n applications. This makes analysis of all the possible use-cases infeasible for a large number of applications. In order to handle these situations, dynamic resource allocation employing on-the-fly processing needs to be applied even though optimal solutions are not guaranteed.

The resource allocation problem is being addressed by several research groups across the globe, which is evident from the massive available literature in this direction. Applications from various domains (e.g., automotive and video processing) are being targeted to allocate them on multi-/many-core-based systems in order to exploit their parallel processing capability. For a given domain, the knowledge of applications is used to devise an efficient resource allocation strategy. However, getting application domain knowledge might be time consuming and challenging. Research progress in

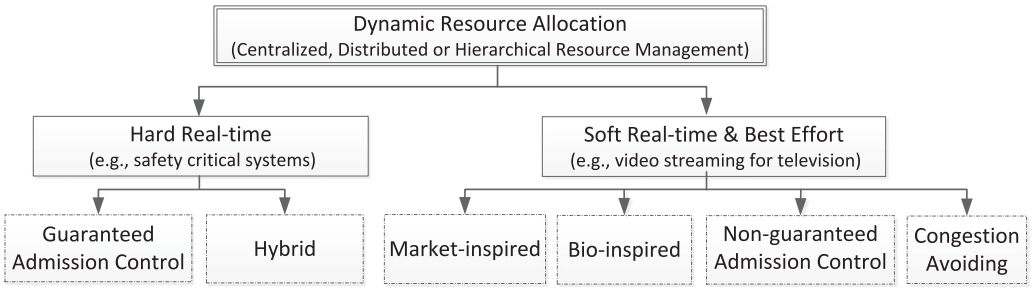

Fig. 2. A taxonomy of dynamic resource allocation strategies.

the direction of dynamic resource allocation is being published in several top-ranked conferences/journals to move beyond state of the art.

1.3. Classification of Dynamic Resource Allocation Strategies

The dynamic resource allocation (DRA) strategies can be classified with a number of taxonomies, which could be based on optimization criteria (performance or energy), target architecture (homogeneous or heterogeneous), criticality (hard or soft real-time), and so on. Broadly, the classification can be done based on criticality, and other taxonomies can be included at some hierarchy in the criticality-based classification. For example, hard real-time dynamic resource allocation can target homogeneous or heterogeneous systems and carry out optimization for performance or energy. Figure 2 shows a classification of the DRA strategies based on the criticality of the systems. The systems that need to guarantee the timing constraints (e.g., automotive engine management, operating medical equipment, and flight control software, realized on a multi-/many-core architecture) require *hard real-time* resource allocation approaches, whereas *soft real-time and best-effort* resource allocation approaches are desired where deadline miss can be tolerated (e.g., video streaming for television and HPC systems).

For *hard real-time resource allocation*, the existing works reported in the literature can be classified into several categories. However, a careful observation of these works has led them to categorize broadly into *Guaranteed Admission Control* and *Hybrid* approaches, as shown in Figure 2. In guaranteed admission control, concepts from scheduling theory are used to ensure that only requirement-satisfying applications are entered into the system during the course of resource allocation. The hybrid approaches utilize design-time computations in order to identify a timing constraint satisfying allocation at runtime.

The existing *soft real-time and best-effort resource allocation* approaches can be broadly categorized into *Market-Inspired*, *Bio-Inspired*, *Non-Guaranteed Admission Control*, and *Congestion-Avoiding* strategies after observing the basic employed principles, as shown in Figure 2. The strategies into different categories utilize some basic principles to optimize for one or several performance metrics in order to fulfill the end-user demands.

The strategies under both the above categories perform computation either at both design time and runtime or only at runtime depending on the known system status at design time. For example, if the applications to be executed in the platform are fixed and known at design time, for example, workloads of an application-specific HPC data center, the allocations can be computed at design time and used at runtime; otherwise, the allocations need to be computed at runtime by applying best-effort heuristics.

The dynamic resource management process is carried out by employing a *centralized* [Nollet et al. 2008; Carvalho et al. 2010; Ng et al. 2015], *distributed* [Peter et al. 2009;

Kobbe et al. 2011; Castilhos et al. 2013], or *hierarchical* [Götzinger et al. 2016; Quan and Pimentel 2016] approach. In centralized management, one core of the platform is used as the manager that handles the allocation process. For distributed management, the platform is divided into regions (clusters) and one core in each cluster manages the allocation process inside the cluster. The cluster managers communicate with each other through a global manager to find the best cluster for allocating an application. The hierarchical approach exploits the features of both the centralized and distributed approaches.

There are some resource allocation surveys reported in the literature, but they have several limitations; for example, they focus only on hard real-time resource allocation [Davis and Burns 2011], cover strategies based on only one basic principle [Yeo and Buyya 2006], focus on a specific domain [Hussain et al. 2013; Hameed et al. 2014] or optimization criteria [Henkel et al. 2013], and do not explicitly and extensively cover hard and soft real-time aspects [Pop and Kumar 2004; Lombardi and Milano 2012; Zhuravlev et al. 2012; Sahu and Chattopadhyay 2013; Singh et al. 2013b]. Based on the aforementioned taxonomy, this article presents an in-depth *survey* and *comparative study* of dynamic resource allocation strategies, which have been reported in the literature since the mid-1990s. The strategies that try to specifically optimize *compute performance*, *energy consumption*, or *both of them* have been considered towards focusing on the most important metrics of real-time multi-/many-core-based systems. They also might optimize for some other performance metrics such as reliability and temperature, but they are not considered to limit the scope of comparison. Optimizing for compute performance is of paramount importance in order to meet the timing deadlines or to minimize the time taken to finish some applications. The compute performance may refer to total *execution time*, *latency*, *delay*, *period*, *throughput*, *exploration time*, worst-case response time (WCRT), and so on, which are related to timing information. Optimizing for the energy consumption of modern computing systems, for example, embedded and high-performance centers, is important, as they are usually operated by a standalone power supply, like a battery, or a huge amount of energy is required to operate such systems. The energy optimization needs to be performed to increase the operational time of the systems and reduce energy costs.

To include aspects of other taxonomies, for example, target architecture (homogeneous or heterogeneous) and resource control mechanisms (centralized, distributed, or hierarchical), the strategies classified based on criticality (hard or soft real time, as shown in Figure 2) are analyzed to highlight the considered *optimization goal*, *target architecture type*, or *control mechanism* and their consideration for *only computation or both computation and communication optimization*. The strategies have also been compared and analyzed to highlight their strengths and weaknesses. The above investigations have also enabled us to observe the trend followed by the strategies and identify significant open issues and promising future research directions.

Paper Organization: Section 2 and Section 3 cover analysis and elaboration of hard real-time and soft real-time and best-effort resource allocation strategies, respectively. A comparative study of strategies falling into different categories is presented in Section 4. Section 5 provides the upcoming trends that could be followed as future research and open research challenges. Finally, Section 6 provides some concluding remarks.

2. HARD REAL-TIME RESOURCE ALLOCATION

The majority of works reported in the literature for hard real-time resource allocation assume the workloads to be known in advance, that is, the allocation decisions by taking the timing constraints are computed at design time [Han and Lin 1989; Saifullah et al. 2011; Bonifaci et al. 2013; Jahr et al. 2014; de Matos Pedro et al. 2015]. However, the main focus in this article has been on dynamic resource allocation that needs to

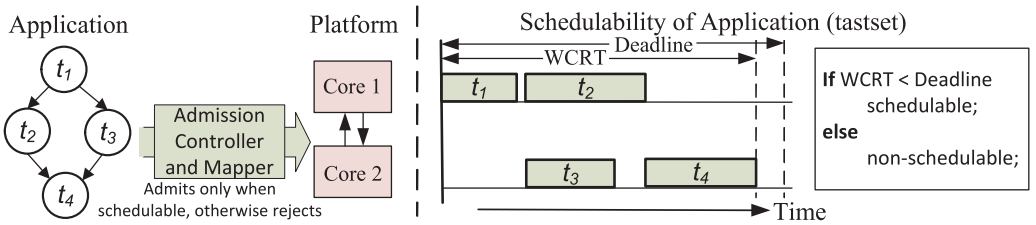

Fig. 3. Guaranteed admission control-based resource allocation.

compute allocations at runtime for dynamically arriving workloads. Based on the earlier classification as in Figure 2, hard real-time dynamic resource allocation approaches are described next along with the advantages and drawbacks of approaches in each category. Further, a combined discussion of all the approaches is provided.

2.1. Guaranteed Admission Control-Based Resource Allocation

A guaranteed admission control ensures that all the admitted applications or tasks in a system will meet their respective deadlines without forcing other running applications/tasks to miss theirs. To achieve such a control, an admission controller that fully guarantees the schedulability of the admitted applications or tasks is desired. To ensure the schedulability of the application, schedulability analysis has been extensively used to determine whether the application or taskset is schedulable or not when to be allocated on platform core(s) [Leontyev and Anderson 2008]. Various kinds of schedulability tests have been employed in the literature, for example, response time tests [Audsley et al. 1993; Davis and Burns 2011; Indrusiak 2014] and utilization tests [Lopez et al. 2004; Baker and Baruah 2007]. An application is schedulable if its end-to-end worst-case response time is less than or equal to its deadline [Dertouzos and Mok 1989]. Similarly, a taskset is schedulable if all its tasks are schedulable. The response time for each task can be estimated by employing widely available standard techniques that takes the interference of the tasks with higher priority into account. Figure 3 provides an overview of the admission controller-based resource allocation and schedulability analysis that determines the end-to-end WCRT by taking the interference with running tasks into account to identify if the taskset meets the deadline. First, tasks are mapped by following an allocation policy. Then, schedulability analysis is performed to determine WCRT. If the WCRT is less than the application deadline, then it is considered to be schedulable, and only then admission controller admits the application into the system. Once admitted, it is mapped onto the platform cores by following the allocation policy used during the schedulability analysis.

The schedulability analysis is performed by taking the allocations of tasks and scheduling algorithm into account, where the former determines tasks to platform cores assignment and latter defines execution order of tasks assigned to a core. Therefore, the schedulability of a task or application (taskset) depends on the employed allocation and scheduling algorithm. At runtime, the schedulability of a task considers the entire knowledge on the working scenario at that time, for example, slack and utilization of cores [Dziurzanski et al. 2016b]. The slack of a core is usually defined as the difference between worst-case execution time and actual execution time of the task currently running on the core. In case of a taskset, the scedulability analysis for each task is done when it arrives into the system. One can also perform worst-case schedulability tests that may be conservative, but safe, as a form of online admission test. In these instances, the real resource usage is unknown, but only the worst-case timing properties are used. For hard real-time systems, the schedulability tests ensuring timing guarantees, for example, a worst-case response time test [Audsley et al.

Table I. Guaranteed Admission Control-Based Resource Allocation Approaches That Consider Computation (Comp.) and/or Communication (Comm.) for Optimization

References	Optimization Goal	Comp. and Comm. Consideration	Architecture	Control
[Lauzac et al. 1998]	Execution time	Comp.	Homogeneous	Centralized
[Isovic and Fohler 2004]	Execution time	Comp.	Homogeneous	Centralized
[Moreira et al. 2005]	Execution time	Comp. & Comm.	Homogeneous	Centralized
[Moreira et al. 2007]	Execution time	Comp. & Comm.	Heterogeneous	Centralized
[Mendis et al. 2014]	Execution time	Comp. & Comm.	Homogeneous	Centralized
[Melani et al. 2015]	Execution time	Comp. & Comm.	Homogeneous	Centralized
[Dziurzanski et al. 2016b]	Execution time	Comp. & Comm.	Homogeneous	Centralized

1993; Mendis et al. 2014] and an exact schedulability test [Davis et al. 2008; Baruah et al. 2010; Dziurzanski et al. 2015], can be employed. A feedback-based admission controller originated from control theory can also be used to facilitate hard real-time allocation as more accurate system status is known by feedback [Lu et al. 2002; Zhu and Mueller 2005; Dziurzanski et al. 2016b]. Feedback mechanisms monitor the capacity of computing resources and quality-of-service levels in order to guarantee a bounded time response and stability even if the exact knowledge of a system workload and service capacity is not available *a priori* [Stankovic et al. 1999]. Thus, by careful fine-tuning of control parameters, they can be successfully applied even to systems with real-time constraints.

2.1.1. Existing Works. Table I lists works employing guaranteed (hard real-time) admission control-based resource allocation of applications on multi-/many-core systems. Such dynamic resource allocation works for multi-/many-core systems are limited, unlike for single-core systems, for example, as in Zhu and Mueller [2005]. There have also been claims that some costly schedulability tests can be used for small task sets in on-line admission control [Bertogna et al. 2005]. These approaches try to find a schedulable allocation at runtime by taking limited available platform resources into account. They use assorted allocation and scheduling algorithms, for example, highest priority task first and earliest deadline task first, and try to optimize only for the response time, which has also been referred to as execution time. While optimizing for the execution time, they also try to optimize other real-time metrics, like number of schedulable applications, deadline misses, utilization, and so on. In these approaches, all the computations are performed at runtime. A few works have been recently reported that utilize design-time computed results for dynamic resource allocation while applying schedulability analysis, but these fall in the category of hybrid resource allocations and are listed in the next subsection.

2.1.2. Advantages and Drawbacks. The schedulability analysis can be used to allocate time-critical applications (e.g., avionics and medical) on required systems. Since such analysis can provide the WCRT of an application in a relatively short amount of time, it can be known whether the application is schedulable. In case the application is not schedulable (WCRT $>$ deadline), a different allocation and scheduling algorithm can be tried. Further, if an application is rejected by an admission controller, then the resource working time is not wasted with the application that will probably violate its deadline, and a possibility of early signalling the lack of admittance can be employed by an outer system to perform an appropriate action (e.g., perform computation on a cloud), minimizing the negative impact of the task rejection. However, the drawback is that users need to have some knowledge of the application domain such that they can analyse the applications and define the allocation and scheduling algorithms to be applied in order to meet the deadline. The scheduling analysis can also incur large

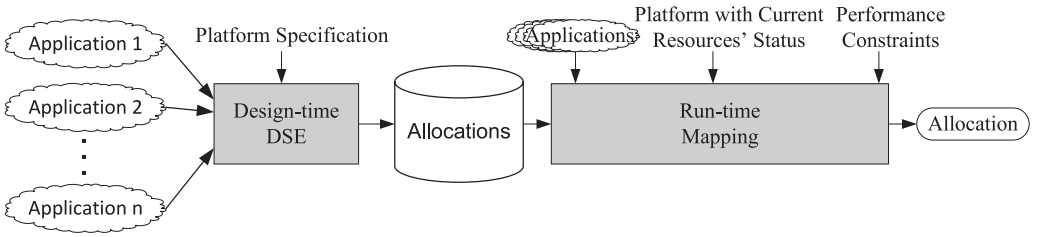

Fig. 4. Hybrid resource allocation: Design-time allocations computation and runtime selection.

computation (timing) overhead for a complex application/platform containing large number of tasks/cores and complex dependencies to manage lots of data flows. Therefore, possibilities to reduce the analysis overhead for such complex scenarios can be explored [Kuo et al. 2003; Davis et al. 2008]. Another drawback is that the platform resources might be very little utilized as only deadline meeting applications are admitted, and their number might be quite low, as hard real-time schedulability tests are pessimistic in that they take only the worst-case conditions into account.

2.2. Hybrid Resource Allocation

Depending on the amount of computation involved at design time and runtime, the existing literature for hybrid approaches can be classified into two categories: *(1) Design-Time Allocations Computation and Runtime Selection* and *(2) Design-Time Deadline Distribution and Runtime Allocation*. The details of these class of approaches are provided subsequently.

2.2.1. Design-Time Allocations Computation and Runtime Selection. This kind of hybrid approach utilizes design-time computed allocations to identify a timing constraint satisfying allocation at runtime. In this approach, the applications to be supported on a platform should be known at design-time in order to perform advance DSE. Figure 4 provides an overview of this hybrid resource allocation approach, which takes advantage of both design-time and runtime computations. The heavy computations pertaining to mappings exploration are performed at design time so only light ones are left for runtime. At design time, for each application, DSE is performed by taking the application and architecture specifications as input in order to explore *allocations* with some design objectives. The explored allocations are stored and used as guidelines to efficiently allocate the applications at runtime [Kwok et al. 2006; Singh et al. 2013a]. The same DSE strategy can be applied to all the applications (Application 1 to Application n) one after another. For the applications to be allocated at runtime, lightweight heuristics are required to select the most efficient allocation for each of them from the storage (precomputed set) of allocations. The selection of an allocation is based on the user demands representing real-time performance constraints of the application and current status (availability) of the platform resources. The selected allocation is used to configure the platform in order to execute the application.

Existing Works. Within the focus of this article, the performance and energy optimizing existing resource allocation approaches have been considered. Table II lists the relevant approaches under this category that optimize performance, energy, and both of them. For an application, all the approaches employ DSE to generate multiple allocations (operating points), which are used at runtime based on the platform status and user requirements. There has also been effort just to develop DSE approaches optimizing for various metrics, such as execution time [Angiolini et al. 2006; Jia et al. 2010; Wildermann et al. 2011; Piscitelli and Pimentel 2012], execution time and energy

Table II. Comparison of Hybrid Resource Allocation Approaches That Consider Computation (Comp.) and/or Communication (Comm.) for Optimizing Execution Time (ET) and/or Energy Consumption (EC)

References	Optimization Goal	Comp. and Comm. Consideration	Architecture	Control
[Huang and Xu 2010]	ET	Comp. & Comm.	Heterogeneous	Centralized
[Stuijk et al. 2010]	ET	Comp. & Comm.	Homogeneous	Centralized
[Weichslgartner et al. 2014]	ET	Comp. & Comm.	Heterogeneous	Centralized
[Dziurzanski et al. 2015]	ET	Comp. & Comm.	Homogeneous	Centralized
[Choudhury et al. 2007]	EC	Comp. & Comm.	Homogeneous	Centralized
[Cong and Gururaj 2009]	EC	Comp. & Comm.	Homogeneous	Centralized
[Schranzhofer et al. 2010]	EC	Comp.	Heterogeneous	Centralized
[Huang et al. 2011b]	EC	Comp. & Comm.	Heterogeneous	Centralized
[Singh et al. 2013a]	EC	Comp. & Comm.	Homogeneous	Centralized
[Javaid et al. 2014]	EC	Comp. & Comm.	Homogeneous	Centralized
[Dziurzanski et al. 2016a]	EC	Comp. & Comm.	Homogeneous	Centralized
[Yang et al. 2002]	ET & EC	Comp. & Comm.	Heterogeneous	Centralized
[Ykman-Couvreur et al. 2011]	ET & EC	Comp. & Comm.	Heterogeneous	Centralized
[Singh et al. 2013a]	ET & EC	Comp. & Comm.	Heterogeneous	Centralized
[Jung et al. 2014]	ET & EC	Comp. & Comm.	Homogeneous	Centralized
[Quan and Pimentel 2015]	ET & EC	Comp. & Comm.	Heterogeneous	Centralized
[Singh et al. 2016b]	ET & EC	Comp. & Comm.	Homogeneous	Centralized

consumption [Zamora et al. 2007; Giovanni et al. 2010], and resource utilization [Xue et al. 2006; Stuijk et al. 2010]. These efforts do not explore the ways to use the DSE results at runtime. However, they can be used to generate allocations to be used at runtime.

The DSE takes a very long time if the application/platform size is large, and hence the exploration may not finish within a limited time. Further, the exploration time increases with the heterogeneity in the platform (different types of cores) and the number of optimization goals to find efficient allocations. To overcome the exploration time bottleneck, there have been efforts to accelerate the DSE process by incorporating estimations along with time-consuming simulations to evaluate the allocations [Piscitelli and Pimentel 2012; Herrera and Sander 2013; Singh et al. 2013b]. The accuracy of the results by these approaches depends on the number of employed simulations. The exploration time could be further reduced by employing pure estimations [Mohanty et al. 2002; Kim and Orshansky 2006], but the evaluation results will not be accurate.

Along with efficient resource allocation, dynamic voltage and frequency scaling (DVFS) potential of cores has also been exploited to optimize energy consumption [Choudhury et al. 2007; Cong and Gururaj 2009; Singh et al. 2013a]. DVFS can be applied both at design time and runtime based on the available time slack. At design time, the slack is defined as the difference between the applications timing constraint and achieved execution time. At runtime, the slack is dynamically created due to tasks finishing earlier than their WCETs.

Advantages and Drawbacks. At runtime, since only selection of the allocation from the storage is required, a lightweight runtime platform manager can be employed to configure the applications efficiently. The hybrid approach allocates applications more efficiently than on-the-fly heuristics that perform all the computations at runtime. However, flexibility in these approaches is limited, since all potential applications must be known in their entirety at design time, and analysis results will be applicable only to the analyzed platform. Therefore, design-time analysis needs to be repeated when the application set or platform changes. Further, storing analysis results introduces additional memory overhead.

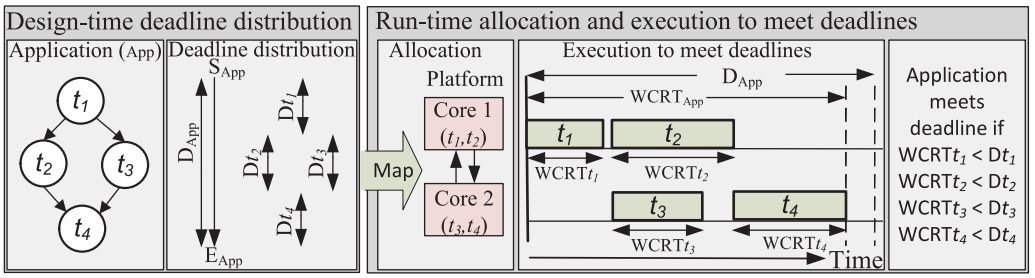

Fig. 5. Design-time deadline distribution and runtime resource management.

Table III. Deadline Distribution-Based Resource Allocation Approaches That Consider Computation (Comp.) and/or Communication (Comm.) for Optimization

References	Optimization Goal	Comp. and Comm. Consideration	Architecture	Control
[Saksena and Hong 1996]	Execution time	Comp.	Homogeneous	Centralized
[Jonsson and Shin 2002]	Execution time	Comp.	Homogeneous	Centralized
[Jayachandran and Abdelzaher 2008]	Execution time	Comp.	Homogeneous	Centralized
[Serreli et al. 2009]	Execution time	Comp.	Homogeneous	Centralized
[Buttazzo et al. 2011]	Execution time	Comp.	Homogeneous	Centralized
[Lee et al. 2012]	Execution time	Comp.	Homogeneous	Centralized
[Hong et al. 2015]	Execution time	Comp.	Homogeneous	Centralized

2.2.2. Design-Time Deadline Distribution and Runtime Allocation. To meet a hard real-time (HRT) deadline, some researchers have tried to distribute deadlines to each task of the application at design time, and the resource management approach aims to meet these deadlines at runtime during the resource allocation and application execution process. There has also been effort to distribute a task's deadline further to its functional blocks [Ahmed et al. 2011]. Figure 5 provides an overview of the design-time deadline assignments to different tasks of an application and runtime resource allocation to meet the deadlines. Based on the application deadline (D_{App}), the computation complexity of its tasks are evaluated, and then a deadline is assigned to each task based on its computation complexity. At runtime, the tasks are allocated to platform cores to perform execution. If individual tasks meet their deadlines, that is, the WCRT of each task is less than its respective deadline (D), then the application meets its deadline.

Existing Works. Table III lists deadline distribution-based resource allocation approaches reported in the literature. We highlight that even though some of the reviewed techniques were not originally developed targeting multi-/many-core systems, they are based on similar assumptions and can or have been applied to the multi-/many-core allocation problem. Similarly to guaranteed-admission control-based resource allocation approaches, these approaches also optimize only the execution time (response time), as their main focus is to meet deadlines. The deadline distribution considers mainly two types of deadlines, global and local, which define deadlines for a taskset and individual tasks, respectively. With deadlines available for each task, the listed approaches apply various kinds of allocation and scheduling algorithms, for example, earliest deadline first.

Advantages and Drawbacks. Deadline distribution enables us to consider the allocation of individual application tasks independently by considering their deadlines and scheduling order. This facilitates optimization for each individual task in order to meet its local deadline towards meeting the global deadline of the application or taskset. Thus, only local deadlines need to be taken into account. However, this might lead

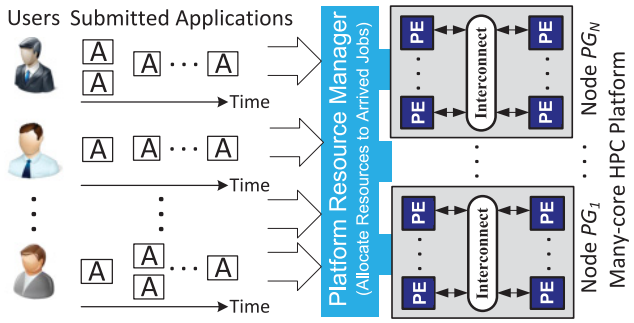

Fig. 6. Market-inspired resource allocation in a cloud data center that contains different nodes (servers) with dedicated cores (or Processing Elements (PEs)) to execute applications submitted by multiple users.

to a situation where all the local deadlines are not met, but the end-to-end deadline might be met. A few drawbacks might be observed when the exact task properties are not known; for example, a subtask deadline calculation might be wrong if the exact execution time of a task is not known, and the calculation might be too pessimistic if only WCET is known. Further, in the case of a complex application containing a large number of tasks, the deadline distribution to individual tasks considering dependencies among the tasks might be quite cumbersome. Additionally, the complexity of the runtime resource management to meet the deadlines also increases.

2.3. Discussions and Summary

These hard real-time approaches have been extensively studied in the literature and applied to meet the deadlines for time-critical applications. The concepts from one category of approaches can be used in another one as well in order to accomplish the aim in a particular category. For example, deadline distribution and schedulability analysis have been jointly exploited in Rivas et al. [2010], deadline distribution and admission control in Marinca et al. [2004], and hybrid and schedulability analysis is employed in Dziurzanski et al. [2015]. For all the categories, it can also be deduced that complexity of the resource allocation increases with the complexity of the considered application. From the above categories, since hybrid resource allocation needs to perform computations at design time, the taskset needs to be known in advance. However, most of the guaranteed admission control-based approaches perform allocation and schedulability analysis directly at runtime as the analysis can be performed in a short amount of time [Buttazzo 2011].

3. SOFT REAL-TIME AND BEST EFFORT RESOURCE ALLOCATION

Based on the classification as shown in Figure 2, soft real-time and best-effort resource allocation approaches are described subsequently. Advantages and drawbacks of approaches in each category are also described. Further, a combined discussion of all the approaches is also provided.

3.1. Market-Inspired Resource Allocation

Market-inspired resource allocation mechanisms use market concepts to perform the allocation. These mechanisms use available platform capacity measured by low-level heuristics as bids within an auctionlike allocation process to find the allocation that can provide guarantees to satisfy the required level of quality of service (QoS) and can maximize the overall system utility (profit). Figure 6 demonstrates the process of market-inspired dynamic resource allocation where different applications need to be allocated into a many-core system representing a typical HPC data center. The shown

Table IV. Market-Inspired Resource Allocation Approaches That Consider Computation (Comp.) and/or Communication (Comm.) for Optimizing Execution Time (ET) and/or Energy Consumption (EC)

References	Optimization Goal	Comp. and Comm. Consideration	Architecture	Control
[Theocharides et al. 2010]	ET (Value)	Comp.	Heterogeneous	Centralized
[Bansal and Pruhs 2010]	ET (Value)	Comp.	Homogeneous	Centralized
[Burkimsher 2014]	ET (Value)	Comp. & Comm.	Homogeneous	Centralized
[Singh et al. 2015a]	ET (Value)	Comp. & Comm.	Homogeneous	Centralized
[Aksanli and Rosing 2014]	EC	Comp.	Homogeneous	Centralized
[Calheiros and Buyya 2014]	EC	Comp.	Homogeneous	Centralized
[Wang et al. 2015]	EC	Comp.	Homogeneous	Centralized
[Khemka et al. 2015]	Value & EC	Comp.	Homogeneous	Centralized
[Singh et al. 2015b]	Value & EC	Comp. & Comm.	Homogeneous	Centralized
[Singh et al. 2016]	Value & EC	Comp. & Comm.	Homogeneous	Centralized

system executes a set of *applications* submitted by various *users* at different moments of time. The applications are submitted to the *platform resource manager* that allocates resources to them. To incorporate market and value concepts in the allocation process, applications are assigned values, and bids from resources are placed in the allocation engine (Manager Processor) to maximize the value/profit returned by a many-core system. The values of applications represent their importance level.

3.1.1. Existing Works. Table IV lists some market-inspired resource allocation approaches that optimize for execution time, energy consumption, or both of them. The main goal of the execution-time-optimizing approaches is to maximize value (profit) by early completion of applications. Some researchers assume a fixed value of an application [Theocharides et al. 2010], whereas others consider values that can change with time, described with a so-called value curve of the application [Burns et al. 2000; Khemka et al. 2015; Irwin et al. 2004; Chen and Muhlethaler 1996]. The changing value over time reflects the impact of the computation over the business processes and adds complexity to the allocation process. With changing value over time, an early completion leads to a high value, whereas late completion results in a low value. An enormous amount of literature exists for optimizing value, as the main focus of cloud data centers have been to maximize the profit. However, since the energy consumption of data centers is quite huge (around 1.5% of the worldwide electricity consumption [Koomey 2011]) and rapidly increasing, optimizing for energy consumption along with the value is of paramount importance and recently has been a focus point of various researchers. Along with allocation, DVFS potential has been exploited to achieve energy savings [Calheiros and Buyya 2014; Singh et al. 2016].

3.1.2. Advantages and Drawbacks. These resource allocation approaches are proven to provide promising results in the overload situation where demand for available resources is higher than the supply. Such a situation is normally encountered in HPC data centers [Yeo and Buyya 2006]. The notion of values of applications and auction process facilitate in deciding to hold the low-value applications for late allocation and allocating limited resources to the high-value applications. However, with a value curve associated with each application, resource allocation becomes complex while considering an enormous number of applications that arrive at the same time due to an overload situation. Depending on the arrival rate and value curve pattern of applications, an appropriate resource allocation approach needs to be identified to perform optimization for desired metrics.

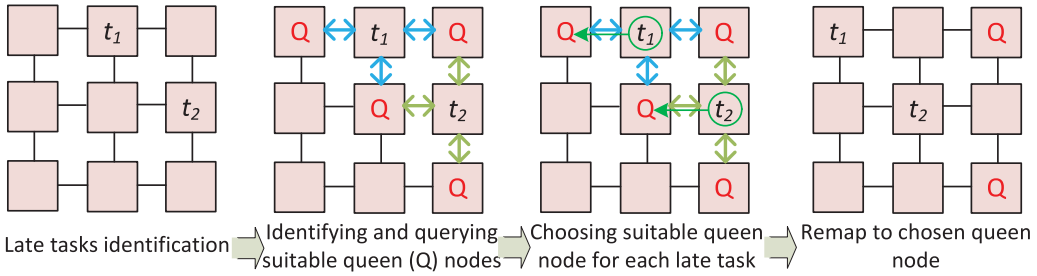

Fig. 7. Bio-inspired resource allocation. An example showing remapping of late tasks t_1 and t_2 on suitable queen nodes (cores) having higher resource availability.

3.2. Bio-Inspired Resource Allocation

Biologically inspired resource allocation approaches draw inspiration from nature and apply the observed characteristics when solving specific computational problems [Heiss and Schmitz 1995]. They are often based on the characteristics of self-organizing biological systems where global patterns emerge from interactions at a lower level in the system [Camazine et al. 2001; Babaoglu et al. 2006]. This indicates that the resource allocations are changed, that is, reallocations are performed based on observed biological phenomena. They have been well explored to balance communication loads in networks and distributed systems [Nishitha and Reddy 2012; da Silva Rego et al. 2012], communication loads in many-core systems [Rowlings et al. 2015], and both computation and communication loads in embedded systems [Mendis et al. 2015]. These approaches usually employ distributed resource management to overcome the limitations of centralized and clustered (hierarchical distributed) management for dynamic applications and large-scale many-core systems. Figure 7 illustrates an example to reallocate some late tasks in a 3×3 many-core system by employing a pheromone signalling mechanism seen in social insects (e.g., honey bees) to improve the overall performance [Mendis et al. 2015]. At each reallocation interval, *late task identification* is done on each node to find late tasks in their task queues. Then, for late tasks, for example, t_1 and t_2, the process of *identifying and querying suitable queen (Q) nodes* is carried out, where each Q node represents its pheromone level in terms of processing capability that is obtained periodically by executing a lightweight set of rules on each node. Only the nodes having pheromone levels greater than a threshold contribute to the Q nodes, and suitable Q nodes are the ones in close proximity. Thereafter, the suitability of each Q node is determined to reallocate the late tasks in order to *choose the most suitable queen node for each late task*. The neighbouring Q nodes are chosen to reallocate (remap) the late tasks such that they are still allocated close to each other in order to maintain low communication overhead in case the tasks communicate with each other.

3.2.1. Existing Works. Table V lists bio-inspired resource allocation approaches for many-core systems. It has been observed that they have been less explored for many-core systems than networks and distributed systems [Nishitha and Reddy 2012; da Silva Rego et al. 2012]. These approaches have performed optimizations by considering several biological phenomena, for example, particle swarm optimization [Jha et al. 2014], pheromone signalling [Mendis et al. 2015], and so on. In Mendis et al. [2015], energy consumption is not directly optimized, but communication overhead is reduced, which leads to optimized communication energy consumption. These approaches have been studied to implement a fully distributed resource allocation in order to overcome the limitations of centralized and clustered (hierarchical) resource managements and have shown promising results in several scenarios.

Table V. Bio-Inspired Resource Allocation Approaches That Consider Computation (Comp.) and/or Communication (Comm.) for Optimizing Execution Time (ET) and/or Energy Consumption (EC)

References	Optimization Goal	Comp. and Comm. Consideration	Architecture	Control
[Heiss and Schmitz 1995]	ET	Comp.	Homogeneous	Distributed
[Brinkschulte et al. 2007]	ET	Comp.	Heterogeneous	Distributed
[Mudry and Tempesti 2009]	ET	Comp.	Homogeneous	Distributed
[Nayak et al. 2012]	ET	Comp.	Heterogeneous	Distributed
[Betting and Brinkschulte 2014]	ET	Comp.	Homogeneous	Distributed
[Barbagallo et al. 2010]	EC	Comp.	Homogeneous	Distributed
[Jha et al. 2014]	ET & EC	Comp. & Comm.	Homogeneous	Distributed
[Mendis et al. 2015]	ET & EC	Comp. & Comm.	Homogeneous	Distributed

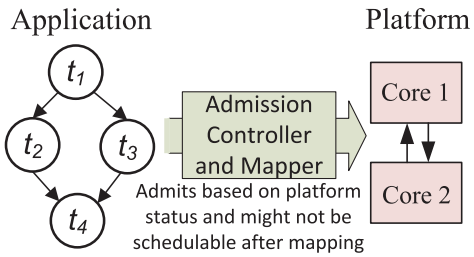

Fig. 8. Non-guaranteed admission control-based resource allocation.

3.2.2. Advantages and Drawbacks. The bio-inspired approaches perform allocation in a completely distributed and self-organizing way. Thus, they alleviate the limitations of centralized and hierarchical (mixture of centralized and distributed) resource allocation approaches that suffer from the issues of scalability, monitoring large amounts of traffic, hot spots, and so on. The main drawbacks of these approaches lie in the fact that they are difficult to implement, study, and calibrate in real systems due to several assumptions on the running system. Additionally, in static environments with a small number of cores, they might lead to bad results as compared to other design-time (static) optimization approaches. Further, for a system of small size, they might perform worse than the centralized resource management approaches that have a better view of the system resources. For various system sizes, in Kobbe et al. [2011], it has been shown that applications achieve better performance with the centralized approach than the distributed approach.

3.3. Non-guaranteed Admission Control-Based Resource Allocation

A non-guaranteed admission controller does not guarantee the schedulability of the admitted applications or tasks. Therefore, some of the admitted applications might not meet their deadlines. For guaranteed admission control, computation costly schedulability analysis (e.g., exact schedulability test) is employed, which takes several measures into account, for example, computation/communication requirements of tasks and interference among tasks, such that timing guarantees are always fulfilled for the admitted and running tasks. The costly analysis might incur long delays between the application release (arrival) and its allocation process. Further, there might be very low utilization of system resources as the tasks are admitted only when the system can allocate a necessary resource budget to meet timing requirements. To overcome the issues of costly schedulability analysis and low utilization, low-cost tests can be employed, which would let a higher number of tasks enter the system to increase resource utilization, but some of them might miss their deadlines. Figure 8 shows such a process,

Table VI. Non-Guaranteed Admission Control-Based Resource Allocation Approaches That Consider Computation (Comp.) and/or Communication (Comm.) for Optimization

References	Optimization Goal	Comp. and Comm. Consideration	Architecture	Control
[Lakshmanan et al. 2009]	Execution time	Comp.	Homogeneous	Centralized
[Kumar et al. 2010]	Execution time	Comp. & Comm.	Homogeneous	Centralized
[Mendis et al. 2014]	Execution time	Comp. & Comm.	Homogeneous	Centralized
[Lin et al. 2010]	Energy consumption	Comp.	Homogeneous	Centralized

representing non-guaranteed admission control-based resource allocation, where some of the admitted applications might miss their deadlines as lightweight schedulability tests are employed that are usually based on the status of the platform resources and thus does not ensure schedulability. Feedback-based admission controller can also be used to provide monitored system information in order to facilitate for better admission control decisions.

3.3.1. Existing Works. Table VI lists non-guaranteed admission control-based resource allocation approaches/works applying lightweight schedulability tests for soft real-time systems. Some of the approaches also employ feedback concepts originated from control theory. For overall execution time optimization, some approaches admit and allocate the tasks such that a load balance is achieved in the multi-/many-core system to achieve high performance. The energy optimization is mainly based on the feedback utilized to perform DVFS. Monitoring (feedback) information has also been used to perform adaptive resource allocation to improve performance [Huang et al. 2011a; Lee et al. 2013]. It has been observed that several works studying feedback admission control exist for single-core systems, but we focus on multi-/many-core systems. Additionally, admission control-based approaches to jointly optimize both execution time and energy consumption are lacking.

3.3.2. Advantages and Drawbacks. The non-guaranteed admission control enables high system utilization by admitting a higher number of applications. Additionally, lightweight schedulability analysis reduces the delay between the start of schedulability test and the time of admission into the system. This might lead to lower response time. However, in the case of using feedback that is some monitored activity in the system, fast and accurate runtime monitoring is desired, which might not always be easily achievable. Further, monitoring activities can flood the network and increase energy consumption.

3.4. Congestion-Avoiding Resource Allocation

The congestion-avoiding heuristics assign new arriving tasks or application tasks in many-core system resources such that the congestion in the cores and links is minimized. This might lead to optimized overall execution time and energy consumption. These heuristics do not guarantee a QoS requirement but can be used to meet soft real-time requirements as they optimize congestion, leading to optimized execution time. Figure 9 shows an example mapping of the computations (tasks) and communications (edges) of an arrived application on a many-core system's cores and links, respectively, while trying to reduce congestion in the cores/links. Tasks t_1 and t_2 are mapped onto one core, assuming that they are highly communicating tasks (huge amounts of data need to be transferred between them) and can fit on the core, that is, the core has sufficient memory to allocate them. This reduces the communication overhead between the tasks, as data transfer happens via shared local memory on the core. The other tasks are mapped in the close proximity, assuming that higher amounts of data need to be transferred between t_3 and t_4 as compared to t_2 and t_4, that is, t_3 and t_4 are mapped

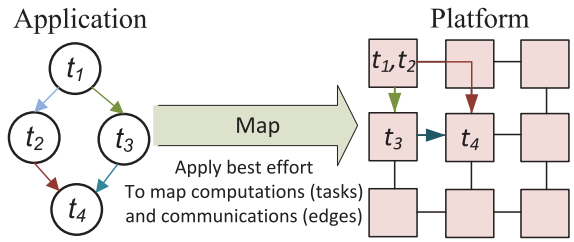

Fig. 9. Congestion-avoiding resource allocation in a many-core system for an arrived application.

Table VII. Congestion-Avoiding Resource Allocation Approaches That Consider Computation (Comp.) and/or Communication (Comm.) for Optimizing Execution Time (ET) and/or Energy Consumption (EC)

References	Optimization Goal	Comp. and Comm. Consideration	Architecture	Control
[Moreira et al. 2007]	ET	Comp. & Comm.	Homogeneous	Centralized
[Nollet et al. 2008]	ET	Comp. & Comm.	Heterogeneous	Centralized
[Al Faruque et al. 2008]	ET	Comp. & Comm.	Heterogeneous	Distributed
[Hong et al. 2009]	ET	Comp.	Homogeneous	Centralized
[Shojaei et al. 2009]	ET	Comp.	Homogeneous	Centralized
[Peter et al. 2009]	ET	Comp. & Comm.	Homogeneous	Distributed
[Theocharides et al. 2009]	ET	Comp.	Heterogeneous	Centralized
[Wang et al. 2011]	ET	Comp. & Comm.	Heterogeneous	Centralized
[Blanch et al. 2011]	ET	Comp.	Heterogeneous	Centralized
[Huang et al. 2011a]	ET	Comp. & Comm.	Heterogeneous	Centralized
[Kobbe et al. 2011]	ET	Comp. & Comm.	Heterogeneous	Distributed
[Chen et al. 2012]	ET	Comp.	Heterogeneous	Centralized
[Chou and Marculescu 2008]	EC	Comp. & Comm.	Homogeneous	Centralized
[Mandelli et al. 2011]	EC	Comp. & Comm.	Homogeneous	Centralized
[Sun et al. 2010]	EC	Comp. & Comm.	Homogeneous	Centralized
[Ost et al. 2013]	EC	Comp. & Comm.	Homogeneous	Centralized
[Smit et al. 2004]	ET & EC	Comp. & Comm.	Heterogeneous	Centralized
[Mehran et al. 2008]	ET & EC	Comp. & Comm.	Homogeneous	Centralized
[Briáo et al. 2008]	ET & EC	Comp. & Comm.	Homogeneous	Centralized
[Chou et al. 2008]	ET & EC	Comp. & Comm.	Homogeneous	Centralized
[Schranzhofer et al. 2009]	ET & EC	Comp. & Comm.	Heterogeneous	Centralized
[Carvalho et al. 2010]	ET & EC	Comp. & Comm.	Heterogeneous	Centralized
[ter Braak et al. 2010]	ET & EC	Comp. & Comm.	Heterogeneous	Centralized
[Castilhos et al. 2013]	ET & EC	Comp. & Comm.	Homogeneous	Distributed
[Modarressi et al. 2013]	ET & EC	Comp. & Comm.	Homogeneous	Centralized
[Fattah et al. 2014a]	ET & EC	Comp. & Comm.	Homogeneous	Centralized
[Ng et al. 2015]	ET & EC	Comp. & Comm.	Homogeneous	Centralized

close to each other while trying to reduce the communication distance between t_2 and t_4 as well. The congestion on the cores is also reduced by homogeneously distributing loads of tasks on the cores.

3.4.1. Existing Works. Table VII lists recent works applying congestion-avoiding heuristics to allocate tasks/edges on the resources at runtime. The heuristics employ various fundamental principles based on the characteristics of the application and current status of system resources, for example, by mapping highly communicating tasks on the same core or neighbouring cores, edges on least-utilized (highly available) links, and tasks on least-utilized cores. These works optimize for one or several performance

metrics. Further, these heuristics perform all the computations at runtime and thus can handle highly dynamic workloads.

3.4.2. Advantages and Drawbacks. The congestion-avoiding heuristics cope well to map unknown applications (not available at design time) on any platform, as they do not use any platform-specific analysis results computed in advance. However, these heuristics may not be able to guarantee for schedulability, that is, for strict timing deadlines due to lack of any prior analysis and limited computational power at runtime.

3.5. Discussions and Summary

The soft real-time and best-effort resource allocation approaches can be applied to perform optimization for one or several performance metrics depending on the need of the application domain. For example, in battery-operated embedded systems, the timing constraints need to be fulfilled while optimizing for the energy consumption. These approaches employ different kinds of resource management, for example, centralized, distributed, or a mix of both (hierarchical) depending on the size of the many-core system and the workload to be executed on it. It has been observed that distributed management leads to better results than centralized when many-core systems of large sizes are considered [Kobbe et al. 2011]. However, in a relatively smaller system, the centralized management might perform better as the distributed approach incurs additional communication overhead among the several agents without offering significant advantages [Kadin et al. 2009; Kobbe et al. 2011]. In the case where the performance needs to be improved at runtime by employing adaptive resource allocation, task migrations are performed, where the tasks are migrated without completely stopping and restarting on the destination core [Briáo et al. 2008; Peter et al. 2009]. These approaches can also be modified to perform optimizations for other performance metrics, for example, reliability, fault tolerance, temperature, security, and so on, by taking appropriate measures into account.

4. COMPARATIVE STUDY AND SUMMARY

This section shows comparative results of various resource allocation approaches falling under the hard real-time and soft real-time and best-effort categories. We show some example comparisons, where mainly considered approaches are those that employ the same application and multi-/many-core system model or application models that can be easily converted into a unified model such that the same evaluation tool can be used. It should be noted that a single tool chain is not used to evaluate approaches from various categories, but the tool chain to evaluate a set of approaches within a category is the same and taken from in-house/open-source tool chains. This enables a fair comparison of approaches within a category. Further, approaches within each category are compared separately; that is, approaches across different categories are not compared, as they try to achieve objectives by following different principles. This enables comparison of approaches following similar principles. It should also be noted that some approaches are not considered for comparison due to their well-proven inferiority and lack of availability of application models, architecture models, and tool chains. The used application model, architecture model, and the tool chain are listed while presenting the respective results in the next subsections.

4.1. Hard Real-Time Approaches

4.1.1. Guaranteed Admission Control-Based Approaches. The recently reported admission control-based approaches that can be applied to applications represented as task graphs are considered for comparison and listed in Table VIII. For the task graph, these approaches can use outputs from controllers to choose the core for critical-path jobs or

Table VIII. Admission Control-Based Approaches Considered for Comparison with Various Open-Loop (OL) and Closed-Loop (CL) Combinations

References	Approaches	Abbreviation
[Mendis et al. 2014]	Open Loop for critical-path jobs and Open Loop for other jobs	OLOL
[Dziurzanski et al. 2016b]	Open Loop for critical-path jobs and Closed Loop for other jobs	OLCL
[Dziurzanski et al. 2016b]	Closed Loop for critical-path jobs and Open Loop for other jobs	CLOL
[Dziurzanski et al. 2016b]	Closed Loop for critical-path jobs and Closed Loop for other jobs	CLCL

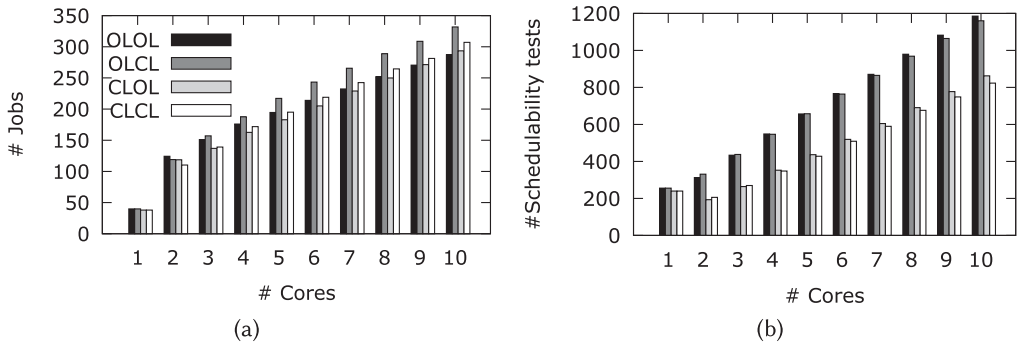

Fig. 10. Number of executed jobs and number of schedulability test executions.

cores for the remaining jobs. The decision of whether to use outputs from controllers for critical paths and other jobs leads to four possible alternatives listed in Table VIII under the column "Approaches." We abbreviate them with four-letter acronyms, where the two first letters denote whether the core selection for critical-path jobs is done without (open-loop (OL)) or with (closed-loop (CL)) controllers and, similarly, the two remaining letters denote if the core selection for jobs outside the critical path is performed without (OL) or with (CL) controllers. Specifically, proportional-integral-derivative controllers are used to perform the comparative study. A Transaction-Level Modelling simulation model developed in the SystemC language has been used to evaluate the efficiency of the approaches. The controller components (proportional, integral, and derivative) are tuned by analysing the corresponding open-loop system response to a bursty workload.

Number of Executed Jobs and Number of Schedulability Test Executions. Figures 10(a) and (b) show the number of jobs executed before their deadlines and number of schedulability tests when multi-core systems with different numbers of cores are considered. The shown results are for grid workload of an engineering design department of a large aircraft manufacturer that contains 100 tasks of 827 to 962 jobs in total, where job execution time varies from 1ms to 99ms, which has been scaled down for faster execution and was originally on the orders of hours. In the OL configuration, cores are scanned in a lexicographical order as long as the first one capable of executing the job satisfying its timing constraints is not found, whereas in the CL configurations, the tasks are checked with regards to the decreasing value of the corresponding controller outputs. A couple of observations can be made from Figure 10. (1) For the number of executed jobs, an OLOL configuration approach seems to be particularly beneficial in systems with lower numbers of cores (more heavily loaded with jobs). However, in systems with more than two cores, the OLCL configuration leads to the best results. Its superiority in comparison with CLCL stems from the fact that an over-pessimistic

Table IX. Hybrid Approaches (Design-Time Allocations Computation and Runtime Selection) Considered for Comparison

References	Approaches	Abbreviation
[Stuijk et al. 2010]	Hybrid Execution Time Optimization	HETOpt
[Choudhury et al. 2007]	Hybrid Energy Consumption Optimization 1	HECOpt1
[Singh et al. 2013a]	Hybrid Energy Consumption Optimization 2	HECOpt2
[Yang et al. 2002]	Hybrid Execution Time & Energy Consumption Optimization 1	HETECOpt1
[Singh et al. 2013a]	Hybrid Execution Time & Energy Consumption Optimization 2	HETECOpt2
[Singh et al. 2016b]	Hybrid Execution Time & Energy Consumption Optimization 3	HETECOpt3

rejection of critical-path jobs leads to fast rejection of the whole task. Thus, the cost of a false-negative estimation is rather high. The OLCL configuration admits 11% more jobs than OLOL, whereas CLCL is only slightly (about 1.5%) better than the baseline OLOL. (2) For the number of schedulability tests, the difference between OLOL and OLCL is almost unnoticeable, but the configurations with control-theory-aided selection of a core for the critical-path jobs, that is, CLCL, leads to a significant, over 30%, reduction. This indicates the benefits of using the controller outputs. From the results, it follows that two configurations, OLCL and CLCL, dominate each other, the former in terms of the number of executed jobs and the latter in terms of the number of schedulability tests. Depending on which goal is more important, one of them is advised to be selected.

4.1.2. Hybrid Approaches: Design-Time Allocations Computation and Runtime Selection. The compared hybrid approaches that perform design-time allocations computation and runtime selection are listed in Table IX. Some of these approaches optimize only execution time or energy consumption, and some of them optimize both metrics. HETOpt tries to find a load-balanced allocation to optimize execution time. The energy consumption optimization in HECOpt1 and HECOpt2 is performed by exploiting expected future slack and current slack, respectively. Appropriate voltage/frequency levels of used cores are identified to exploit the slack. In HETECOpt1, at design time, exhaustive exploration is performed to identify the best allocation in terms of execution time and energy consumption, whereas HETECOpt2 prunes the design space to perform the exploration within a limited time. HETECOpt3 identifies the best allocation at runtime by exploiting the design-time extracted execution traces. To evaluate these approaches, they are implemented using the publicly available synchronous data-flow for free (SDF^3) tool set [Stuijk et al. 2006] along with the required application and architecture models.

Execution Time and Energy Consumption Comparison. Figures 11(a) and (b) show a comparison of worst-case execution time and energy consumption for various streaming multimedia applications when different approaches tabulated in Table IX are employed to map them onto a 4×4 2D mesh multi-core chip. The multimedia applications are characterized by throughput constraints [Bamakhrama and Stefanov 2012]. The results obtained by different approaches are normalized w.r.t. the result obtained by HETECOpt1. A couple of observations can be made from Figures 11(a) and (b). (1) HETOpt leads to the worst result in terms of both execution time and energy consumption, as it tries to perform load-balanced resource allocation of tasks on cores without taking parallelism of tasks and their communication overhead into account. (2) Both execution time and energy consumption optimization approaches HETECOpt1, HETECOpt2, and HETECOpt3 lead to similar results for most of the applications, as all of them compute efficient allocations at design time. (3) Energy consumption by HECOpt1 and HECOpt2

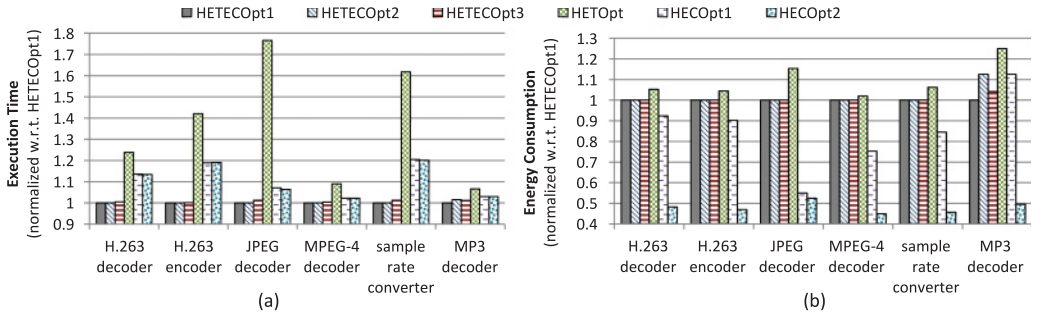

Fig. 11. Execution time and energy consumption comparison.

Table X. Hybrid Approaches (Design-Time Deadline Distribution and Runtime Allocation) Considered for Comparison

References	Approaches	Abbreviation
[Jayachandran and Abdelzaher 2008]	Pipeline Delay Composition	PDC
[Buttazzo et al. 2011]	Automatic Partitioning	AP
[Hong et al. 2015]	Local-Deadline Assignment	LDA

is lower than that of other approaches. The reason lies in the fact that HECOpt1 and HECOpt2 employ DVFS on cores to reduce the energy consumption while respecting the application deadlines, whereas other approaches just try to find the best allocation and do not employ DVFS. (4) HECOpt2 leads to minimum energy consumption as DVFS is applied both at design time and runtime while taking DVFS overhead and deadline into account.

4.1.3. Hybrid Approaches: Design-Time Deadline Distribution and Runtime Allocation. The compared hybrid approaches that perform design-time deadline distribution and runtime allocation are listed in Table X. These approaches distribute the end-to-end deadline of an application/job to its sub-functions and then perform resource allocation in order to meet the deadline. In PDC, end-to-end delay of a job in a multi-stage pipeline is bounded as a function of job execution times on different stages. The AP approach partitions a parallel real-time application into a set of sequential flows. In LDA, a locally optimal algorithm is employed to assign local deadlines to the jobs. The evaluation results for these approaches are employed from Hong et al. [2015], where they are implemented in C++ and consider stream-type and general-type workloads to emulate different kinds of application scenarios.

Number of Executed Task Sets. Figures 12(a) and (b) compare feasible task sets at various utilization levels for balanced and imbalanced workloads when assorted approaches are employed. Each workload contains a total of 100 task sets of 50 tasks each, where individual instances of a task and sub-task are referred to as jobs and sub-jobs, respectively. A total of eight cores are considered. In a balanced workload, the execution time of a job is randomly distributed along its execution path to balance the core loads, whereas cores' loads are imbalanced in an imbalanced workload. A couple of observations can be made from the figure. (1) LDA finds far more feasible sets than the other two methods for both the balanced and imbalanced workloads. (2) LDA performs much better than PDC and AP at high utilization levels where there are more jobs in the system. It should also be noted that sometimes LDA may not be able to find a feasible solution even though such solutions exist, since LDA finds local sub-job deadlines for each core independently instead of using a global approach.

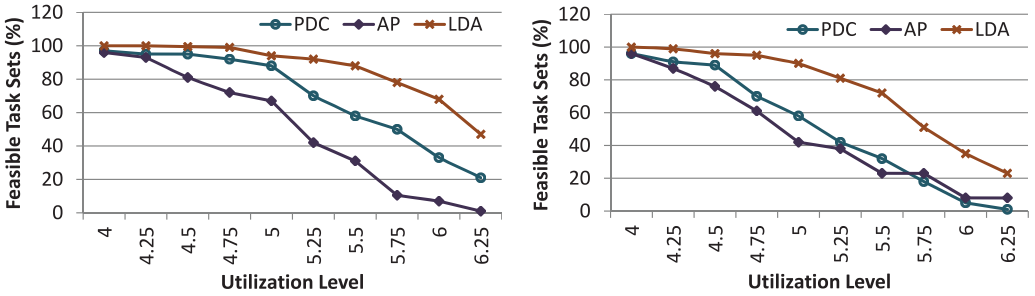

Fig. 12. Percentage of feasible task sets for (a) balanced and (b) imbalanced workloads.

Table XI. Market-Inspired and Value-Based Approaches Considered for Comparison

References	Approaches	Abbreviation
[Theocharides et al. 2010]	Value Optimization 1	ValOpt1
[Bansal and Pruhs 2010]	Value Optimization 2	ValOpt2
[Burkimsher 2014]	Value Optimization 3	ValOpt3
[Singh et al. 2013a]	Value and Energy Separate Optimization	ValEnSepOpt
[Singh et al. 2015b]	Value and Energy Joint Optimization	ValEnJoinOpt
[Singh et al. 2016]	Value and Energy Adaptive Optimization	ValEnAdaptOpt

4.2. Soft Real-Time Approaches

4.2.1. Market-Inspired Approaches. We have considered mainly market-inspired and value-based approaches for comparison. Table XI lists the compared approaches that have been applied to applications containing dependent tasks. The value optimization approaches optimize only for value while applying various principles, for example, ValOpt1 chooses the highest value application first, ValOpt2 first chooses the application having maximum value density computed as the value over the amount of required computational resources, and ValOpt3 chooses the application with the minimum remaining value first. The approach in Singh et al. [2013a] optimizes only for energy and has been extended to optimize both value and energy consumption for a fair comparison, where first a value-optimizing allocation is found and then DVFS is applied to optimize energy consumption. In ValEnJoinOpt, value and energy consumption are jointly optimized by employing a genetic algorithm. In ValEnAdaptOpt, in addition to joint optimization, adaptation (reallocation) is also performed based on the execution status of running applications and available system resources to explore the scope for further optimization. These approaches are implemented in a C++ prototype and integrated with a SystemC functional simulator. They are evaluated by considering job models from historical data of an industrial HPC data center at the High Performance Computing Center Stuttgart.

Value and Energy Consumption Comparison. Figures 13(a) and (b) show the influence of the number of available nodes (servers) on the overall value and energy consumption when various approaches are employed. Each node contains a total of 10 cores. To sufficiently stress the platform, we consider all the applications arriving over a month. Further, to remain close to a real-world scenario, it is considered that a higher number of applications arrives at peak times, that is, weekdays and day times, as compared to off-peak times, that is, weekends and night times. The value and energy consumption results are normalized w.r.t. the value and energy by the ValOpt1 approach at two nodes. A couple of observations can be made from Figure 13. (1) Overall value with

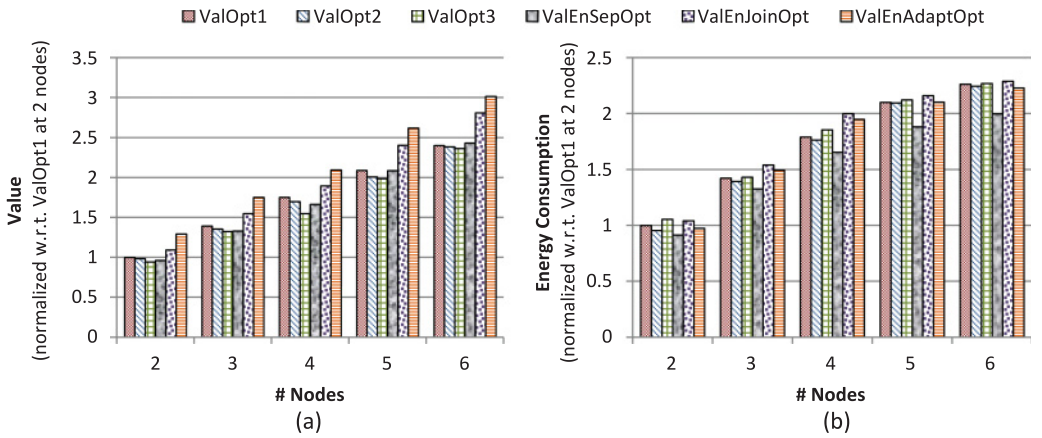

Fig. 13. Value and energy consumption comparison.

Table XII. Bio-Inspired Approaches Considered for Comparison

References	Approaches	Abbreviation
Reference implementation	Random Reallocation	RR
[Castilhos et al. 2013]	Multiple Cluster-Based Reallocation	MCR
[Mendis et al. 2015]	Single Cluster-Based Reallocation	SCR
[Mendis et al. 2015]	Pheromone Signalling-Based Reallocation	PSR

all of the approaches increases with the number of nodes due to increased processing capability leading to completion of a higher number of applications before their value becomes zero. (2) The ValEnAdaptOpt approach achieves a higher overall value than other approaches. This is due to the fact that adaptation leads to early completion of executing applications and thus higher values for them. Further, earlier completion leaves resources for the queued applications to be allocated and completed sooner, leading to higher values. (3) The energy consumption is not the lowest by ValEnAdaptOpt, as it completes execution of higher numbers of applications and energy is consumed for executing them. Further, if both the value and the energy consumption metrics are to be jointly optimized as value achieved per unit of energy consumption, that is, value divided by energy, then ValEnAdaptOpt leads to the best results.

4.2.2. Bio-Inspired Approaches. Table XII lists the bio-inspired approaches considered for comparison. These approaches are applied to perform reallocation on top of an initial allocation done based on the least-utilized heuristic in order to improve average execution time and energy consumption. In RR, at every remapping interval, each core selects the latest task from its task queue and randomly selects another core to remap it. MCR partitions the whole many-core chip into virtual clusters, and each cluster is managed by a local manager that receives states of its cores every time a task completes execution and performs mapping of tasks. In case the cluster has no available cores, it sends a request to neighboring clusters and loans the closest available core to map the task. To have a fair comparison, the MCR approach of Castilhos et al. [2013] has been modified to take the following aspects into account: relocation to a maximum two-hop distance, remapping a late task to a PE with positive slack, and placing the cluster manager in the center of the cluster [Mendis et al. 2015]. SCR is essentially MCR with only one cluster. In PSR, the reallocation is performed based on the pheromone level of the cores. These approaches are evaluated by a discrete-event abstract simulator to map a video stream workload on a 10×10 mesh of cores [Mendis et al. 2015], where

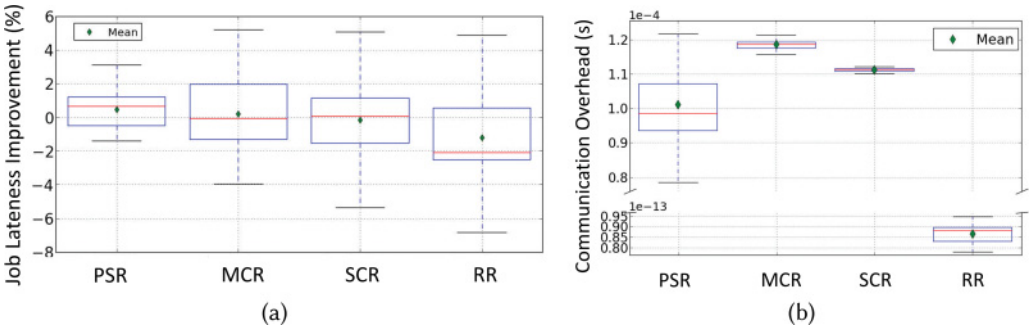

Fig. 14. Job lateness and communication overhead comparison.

cluster size of 2×5 (i.e., 10 clusters) is considered for MCR and the video stream contains a set of jobs arriving at different moments of time.

Execution Time and Energy Consumption Comparison. To evaluate average execution time and energy consumption, the relevant metrics' job lateness and communication overhead are considered. By optimizing the job lateness and communication overhead, average execution time and communication energy can be optimized, where job lateness is computed as the difference between the job deadline and its response time. Figures 14(a) and (b) compare distribution of cumulative job lateness and communication overhead, respectively, when various approaches are employed. For job lateness, all the approaches show positive and negative results. Therefore, the remapping techniques have failed to improve lateness of jobs under certain workload situations. However, the majority of the distribution for PSR and MCR is in the positive region. In over 60% of the workload scenarios, PSR produces positive improvement to the job lateness of the video streams. Further, MCR shows better job lateness improvement over SCR as the monitoring traffic is shorter in route length and hence is less disruptive to the data communication. The random remapper shows the worst results. It is interesting to note that there are a few scenarios where random remapping produce significant job lateness improvements. For communication overhead, PSR shows a significant reduction when compared to some other approaches. The maximum overhead of PSR is comparable to that of MCR. Both the MCR and SCR show a higher and narrower distribution of communication overhead than PSR. A higher upper whisker in PSR shows that under certain workload scenarios the overhead can be costly and similar to the MCR. The lower communication overhead distribution of the SCR when compared to MCR is due to the lack of inter-cluster communication. In SCR, communicating tasks mapped at the middle of the system will suffer due to the network congestion caused by the incoming monitoring traffic and thus communication overhead issues will become severe for larger system sizes. The RR has the lowest communication overhead, as it only incurs overhead when notifying the task dispatcher regarding remapping decisions.

4.2.3. Non-Guaranteed Admission Control Approaches. Table XIII lists the relevant approaches considered for comparison. NAC is considered to analyse the effects of admission controls over no admission control. In GAC, it is ensured that all the admitted tasks/jobs will meet their deadlines. The EQF approach divides the total remaining slack among the subtasks in proportion to their estimated execution times in the hope of reducing individual task deadline miss rates. In HAC, a heuristic-based admission control is employed in order to achieve tradeoff between predictability and utilization. These approaches are evaluated by a discrete-event abstract simulator to map a

Table XIII. Non-Guaranteed Admission Control Approaches Considered for Comparison

References	Approaches	Abbreviation
Reference implementation	No Admission Control	NAC
[Leontyev and Anderson 2008]	Guaranteed Admission Control	GAC
[Kao and Garcia-Molina 1997]	Equal Flexibility	EQF
[Mendis et al. 2014]	Heuristic Admission Control	HAC

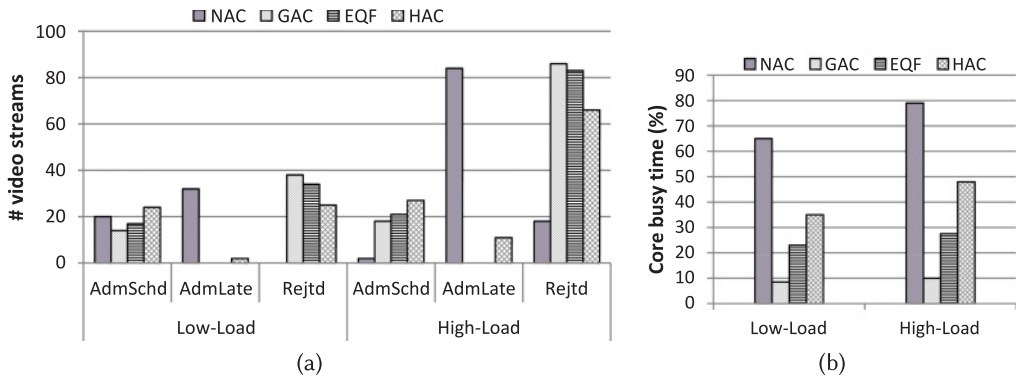

Fig. 15. Video stream admission and utilization comparison.

number of video streams on a 3×3 mesh of cores [Mendis et al. 2014], where each video stream contains a set of jobs arriving at different moments of time.

Admitted Video Streams (Relates to Predictability) and Energy Consumption (Relates to System Utilization) Comparison. Figure 15(a) shows the number of video streams that were admitted and successfully schedulable (AdmSchd), admitted but late (AdmLate), and rejected (Rejtd) when difference approaches are employed under light (Low) and heavy (High) load conditions. The HAC is employed by setting fixed values of constants in the adopted heuristic [Mendis et al. 2014]. A couple of observations can be made from Figure 15(a). (1) For both low- and high-load conditions, when GAC is employed, none of the admitted video streams incur any lateness, that is, AdmLate is zero. (2) In NAC, all the incoming video streams are admitted unless the global input buffers do not have available free space to hold the new tasks. Hence, a large number of admitted video streams miss their deadlines, especially under the high-load situation, and only a few video streams are rejected. (3) The rejected jobs are higher in the high-load situation as the task-queues and input buffers become saturated earlier. (4) EQF and GAC have similar service guarantees, where no admitted streams incur lateness (AdmLate = 0); however, on average, the rejection rate of EQF is lower than GAC, giving a tighter guaranteed decision. Figure 15(b) shows the percentage busy time of all the cores for different approaches under low- and high-load conditions. The percentage is computed by the ratio between the total core busy time and total simulation time. A couple of observations can be made by looking Figures 15(a) and (b). (1) Admitting more streams into the system (whether they are late or schedulable) improves system utilization. In the high-load condition, NAC has peak utilization of about 80%, after which the buffers begin to overflow and streams are rejected. Further, admitting a few high-resolution video streams may cause the system to be more busy than admitting relatively low-resolution ones. (2) GAC has the lowest system busy time, which corresponds well with the high number of stream rejections shown in Figure 15(a). (3)

Table XIV. Congestion-Avoiding Approaches Considered for Comparison

References	Approaches	Abbreviation
[Carvalho et al. 2010]	Non-contiguous Nearest Neighbour	NonContNN
[Modarressi et al. 2013]	Non-contiguous with Task Migration	NonContTM
[Ng et al. 2015]	NonContNN with Defragmentation	NonContNNDefrag
[Chou et al. 2008]	Incremental Contiguous	IncCont
[Sun et al. 2010]	Circular Contiguous	CirCont
[Ng et al. 2015]	CirCont with Defragmentation	CirContDefrag

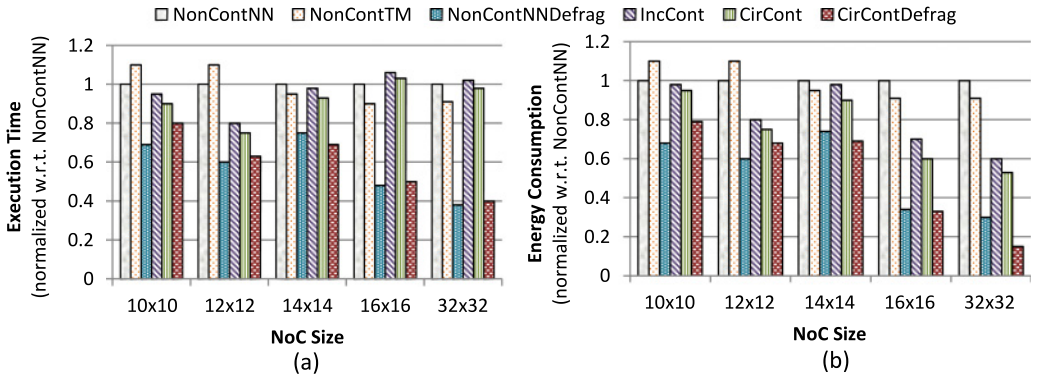

Fig. 16. Execution time and energy consumption comparison.

The system busy time is better by EQF over GAC because of extra admitted streams. Further, HAC shows even higher busy time due to more admitted video streams.

4.2.4. Congestion-Avoiding Approaches. Table XIV lists the promising congestion-avoiding resource allocation approaches considered for comparison. NonContNN tries to find the allocation for a task on the nearest core that is close to the core containing its communicating task. NonContTM employs task migration to obtain a better allocation. NonContNNDefrag employs the defragmentation approach proposed in Ng et al. [2015] on top of NonContNN. In IncCont, for the tasks of an incoming application, an incremental allocation is performed in a selected region that may be long and thin in shape, which may incur a higher communication energy. In contrast, CirCont selects a region that is nearly circular in shape, which leads to small average distance between any pair of cores inside the selected region and thus saves energy consumption. CirContDefrag employs the defragmentation approach of Ng et al. [2015] on top of CirCont. These approaches are implemented by extending Noxim [Fazzino et al. 2008], a SystemC-based Network-on-Chip simulator. Their evaluation is done by considering applications represented as task graphs that have varying arrival times [Ng et al. 2015].

Execution Time and Energy Consumption Comparison. Figures 16(a) and (b) show overall execution time and energy consumption when 100 random applications are allocated on systems of various sizes by employing the congestion-avoiding approaches reported in Table XIV. The inter-application arrival period is set randomly, and the average number of tasks in each application is 8. In the case of non-contiguous mapping, on average, NonContNNDefrag reduces execution time by 35.6% and 34% when compared to NonContNN and NonContTM, respectively. Further, NonContNNDefrag reduces energy consumption by 40.6% and 39% over NonContNN and NonContTM, respectively. In NonContNN and NonContTM, free cores might be scattered, and thus

tasks of the incoming application are allocated to non-contiguous regions. This increases the inter-task communication overhead and thus degrades the performance. On the other hand, NonContNNDefrag leads to better performance as inter-task communication distance is reduced by grouping the free cores into one contiguous region before allocating the incoming application. In the case of contiguous mapping, on average, CirContDefrag shows a 28% reduction in execution time when compared to CirCont. Moreover, on average, CirContDefrag reduced energy consumption by 28% when compared to CirCont. CirContDefrag improves the CirCont approach by reshaping the irregular free core region into a regular and contiguous one with low communication distance; otherwise, CirCont has to wait until a regular free core region is not available even if sufficient scattered cores are available after multiple allocations and deallocations.

4.3. Discussions and Summary

The compared approaches under various categories indicate their superiority for performing different kinds of resource allocations. Therefore, based on the kind of application domain and runtime scenario, one category of approaches is recommended to be employed. The references to recommended previous studies for some common scenarios/applications are as follows. In case the platform is overloaded (i.e., demand for available resources is higher than the supply) in a centralized control environment, market-inspired approaches are suitable, as more profitable jobs can be allocated to the limited resources while holding the low-value jobs for late allocation [Theocharides et al. 2010; Aksanli and Rosing 2014; Calheiros and Buyya 2014; Khemka et al. 2015; Singh et al. 2016]. In a distributed control environment where reallocations need to be performed based on observed biological phenomenon and related platform/application-level metrics (e.g., throughput and lateness), bio-inspired approaches can be employed to perform efficient resource allocation [Brinkschulte et al. 2007; Barbagallo et al. 2010; Nayak et al. 2012; Jha et al. 2014; Mendis et al. 2015]. For typical embedded systems where the applications are known in advance, efficient allocations for each application or use-case (combination of active applications) can be computed at design time with well-defined search heuristics. Such measures can be taken to achieve better results as compared to on-the-fly processing approaches [Huang and Xu 2010; Ykman-Couvreur et al. 2011; Singh et al. 2013b; Weichslgartner et al. 2014; Quan and Pimentel 2015; Singh et al. 2016b]. However, on-the-fly approaches are recommended to be applied when the applications are not known in advance, as they lend well in such scenarios [Moreira et al. 2007; Shojaei et al. 2009; Carvalho et al. 2010; Castilhos et al. 2013; Fattah et al. 2014a; Ng et al. 2015].

Further, depending on the size of the system, a decision to choose between centralized and distributed control can be made by performing experiments with various system sizes. Usually, distributed control leads to better results for large-scale systems, as it alleviates the monitoring traffic problem around the centralized manager [Kadin et al. 2009; Kobbe et al. 2011; Castilhos et al. 2013].

5. UPCOMING TRENDS FOR FUTURE RESEARCH AND OPEN CHALLENGES

This section addresses some of the upcoming trends and challenges to be faced to take the mapping methodologies into the next era.

5.1. Hybrid Resource Allocation

The analysis of dynamic resource allocation approaches indicate that hybrid strategies that combine design space exploration of design-time techniques with runtime management to select mapping configurations that are best suited to newly arriving applications lead to better results than on-the-fly strategies. The already-known

mapping configurations also improve predictability. Further, since they involve minimum computation at runtime, they are appropriate for a lightweight runtime platform manager. This speeds up the mapping process, that is, mapping time, significantly over on-the-fly strategies.

The trend for hybrid resource allocation was introduced earlier [Singh et al. 2013b]. Since then, because of its potential, significant advances have taken place for embedded systems [Weichslgartner et al. 2014; Javaid et al. 2014; Jung et al. 2014; Quan and Pimentel 2015; Dziurzanski et al. 2015; Singh et al. 2016b]. It has also shown its potential for managing data center resources [Singh et al. 2015b, 2016]. With advancement, these approaches have tried to address the involved challenges such as reducing exploration time for large-scale applications/platforms and storage overheads for explored designs. However, since the problem size is continuously increasing because of the additional complexity of applications and the way they share increasingly sophisticated platforms, further studies are required.

Although the advantages of hybrid strategy seem promising, it comes with its own tradeoffs due to inherent pseudo-dynamic nature and inability to handle new applications without available design-time exploration. With no doubt, hybrid strategies seem to be followed in the field of mapping methodologies, but because of their nascent development and lack of in-depth examination, further development of design-time and on-the-fly mapping methodologies will continue hand in hand with hybrid strategies.

5.2. Large-Scale Many-Core Architectures

It is evident that technological enhancement will enable integration of hundreds and even thousands of cores in a single chip [Borkar 2007]. Recently, some large-scale architectures have been introduced, for example, Angstrom [Hoffmann et al. 2012], Kalray's MPPA [De Dinechin et al. 2014], and the KiloCore chip [Bohnenstiehl et al. 2016]. These architectures impose a big challenge by managing their resources at runtime in a scalable manner. Some researchers have tried to address the scalability concern by developing distributed resource management strategies [Kadin et al. 2009; Al Faruque et al. 2008; Kobbe et al. 2011; Ebi et al. 2011]. These distributed strategies need to be further investigated such that the systems can be made more predictable and real-time challenges of foreseeable future can be addressed.

The resource allocation for large-scale many-core architectures has already been a trend [Singh et al. 2013b]. However, due to the integration of higher number of cores in recent years [De Dinechin et al. 2014; Bohnenstiehl et al. 2016] and its expected increase in upcoming years, resource allocation approaches need to advance to address the challenge of efficient exploitation of the abundant amount of cores.

To overcome the issues of large-scale two-dimensional (2D) many-core chips such as large areas, power, and signal transmission delays, integration of multiple layers of cores into a single device is taking place in order to realize 3D many-core architectures [Coskun et al. 2009; Zhou et al. 2010; Cox et al. 2013]. Despite having several advantages of 3D integration, the 3D high integration density brings major concern in the temperature increase that causes thermal hot spots and high temperature gradients. This might lead to an unreliable system and degraded performance. Efficient thermal management of 3D architectures is challenging and requires investigation of efficient resource allocation strategies to cater to the reliability and performance concerns. The performance concern is specially important for real-time applications where predictable execution is desired.

Resource allocation studies for 3D architectures have also advanced. These advanced studies have tried to exploit the 3D architecture through various heuristics. For example, in Cheng et al. [2013], interconnect energy optimization is achieved by allocating heavily communicating edges to fast vertical links, and in Singh et al. [2016a], potential

of 3D-neighborhood correlation available in spatial and temporal domains for 3D videos is exploited.

Further, these large-scale architectures are expected to contain several types of cores, for example, GPP cores, DSP cores, graphic processing unit (GPU) cores, and field programmable gate array (FPGA), to meet functional and non-functional demands by exploiting their distinct features. Such heterogeneous integration will further increase resource management challenges. Recently, small-scale heterogeneous architectures have been released, for example, Samsung Exynos 5422 System-on-Chip [Samsung 2014] that powers the popular Samsung Galaxy S5 smartphone; the chip contains 4 ARM Holdings plc (ARM) Cortex-A15 cores, 4 ARM Cortex-A7 cores and a six-core ARM Mali T628 MP6 GPU. Additionally, some development chips have been released, for example, MediaTek's Helio X20 chip [MediaTek 2016], containing Cortex-A72 cores, Cortex-A53 cores, and Mali-T880 MP4 GPU cores; Xilinx's Zynq UltraScale+ EG MPSoC devices [Xilinx 2016] containing ARM Cortex-A53 cores, Cortex-R5 cores, and a Mali-400 MP2 GPU; and Intel's Xeon-FPGA hybrid chip [Intel 2016] containing a 12-core Intel microprocessor with an Altera Arria10 FPGA. The number of heterogeneous cores in future architectures is going to increase towards the realization of large-scale heterogeneous architectures. Further, academia is also investigating large-scale heterogeneous architectures, for example, Invasive architecture containing thousands of cores [Henkel et al. 2012] and the Loki many-core architecture [Bates et al. 2015]. To exploit these architectures, new frameworks such as OpenCL [OpenCL 2016] have been developed for writing programs that can execute across heterogeneous processing cores, for example, CPU, GPU, DSP, and FPGAs. Existing works have explored the possibility of executing OpenCL programs across CPUs and GPUs [Luk et al. 2009; Grewe et al. 2013; Prakash et al. 2015]. In the future, efficient resource allocation strategies will need to be developed by taking an abundant amount of cores and heterogeneity into account [Ding et al. 2014]. This will address the need to satisfy the ever-increasing performance requirements of modern and future applications that can efficiently utilize large numbers of processor cores.

5.3. Joint Consideration of Computation and Communication Loads

Earlier research on multi-/many-core systems indicates that only computation loads were considered, and optimizations used to be performed only for such loads. Some of these works can be seen in Tables I to VII, where computation (comp.) is entered in the column computation and communication (comp. and comm.) consideration. These works consider loads as a set of independent tasks, which used to be true for single-core systems where each task is executed sequentially. However, to efficiently exploit multi-/many-core systems, the applications are partitioned into multiple tasks that usually have execution order and data dependencies, and such partitioned applications are represented as task graphs, dataflow graphs, and so on. Due to the dependency, once computation of a task is finished, it sends data (representing communication load) to another task that starts execution after receiving the data. This necessitates joint consideration of computation and communication loads during the process of resource allocation.

5.4. Multi-Objective Resource Allocation

The resource allocation problem for multi-/many-core systems has focused a lot on optimizing execution time to meet timing constraints, which is one of the important objectives to achieve safe operation in time-critical systems or better user experience in embedded systems. However, modern multi-/many-core systems need to be optimized for several other metrics along with the execution time due to various reasons such as increasing demand of energy, shrinking transistor sizes leading to unreliable systems

due to leakage power and thermal issues, and security threats due to interactions with untrusted devices.

5.4.1. Jointly Optimizing Execution Time and Energy Consumption. The joint optimization of execution time and energy consumption is of paramount importance in battery-operated systems [Jha et al. 2014; Fattah et al. 2014a; Quan and Pimentel 2015; Singh et al. 2016b] and the limited availability of power for remote devices and HPC data centers [Singh et al. 2015b; Khemka et al. 2015]. This helps to enhance the operational time while still meeting the timing requirements. As mentioned in Tables I to VII, recently there has been extensive focus on such joint optimization. However, with the increase of application/platform complexity and the power requirements of many-core systems, the developments of efficient resource allocation approaches to jointly optimize these two important metrics is expected to continue in the foreseeable future.

5.4.2. Consideration for Multiple Objectives. As mentioned earlier, the requirements to jointly optimize for several performance metrics will grow in the future. Along with execution time and energy consumption, other important metrics are temperature, reliability, fault tolerance, and security. For example, the metrics execution time, energy consumption, and temperature are optimized in Sheikh and Ahmad [2014]. The temperature is to be optimized to surmount its effect on other metrics such as execution time [Murali et al. 2008] and reliability [Wang and Chen 2010] and to reduce the cooling cost of many-core-based HPC data centers [Chaudhry et al. 2015]. Reliability-aware resource allocation is desired to increase the mean time to failure of a system [Singh et al. 2013b]. However, in the case where a fault has occurred, resource reallocation is desired during system operation to make the system fault tolerant [Lee et al. 2010; Derin et al. 2011; Huang et al. 2011b; Schor et al. 2012; Singh et al. 2013b; Das et al. 2013; Fattah et al. 2014b; Sahoo et al. 2016; Tzilis et al. 2016; Zeng et al. 2016]. The demand to support increasing numbers of applications within a device and then their communication with the applications in other devices has created security threats due to possible attacks in the communication channels and interaction with untrusted devices. This indicates that resource allocation approaches need to optimize for security as well, which can be achieved by developing security-aware resource allocation approaches that take into account other metrics.

Optimization for multiple objectives also increases the design space and thus the exploration time. Therefore, in the case where the design space exploration can be performed at design time, for example, for a known application set, the design space needs to be efficiently pruned so Pareto-fronts for several conflicting objectives can be derived within a limited time. The design space increases even further with the heterogeneity of systems, and thus exploration strategies might need to establish an upper limit on the heterogeneity to maintain low complexity.

5.5. Additional Challenges

Some additional challenges also need to be addressed to take the mapping methodologies into the next era, and this includes, for example, the development of efficient programming models for large-scale and 3D architectures, efficient synchronization and control of concurrently executing tasks on such architectures, and debugging of several concurrent executions if the results are not as expected.

6. CONCLUSION

This article provides a survey and comparative study of dynamic resource allocation strategies for multi-/many-core systems. Specially, hard and soft real-time strategies optimizing for execution time and energy consumption are surveyed and compared. The kind of target architecture, for example, homogeneous or heterogeneous, and resource

control mechanism, for example, centralized or distributed, have also been identified. A comparative study of these strategies has shown the superiority of some of them. Based on the analysis of the surveyed and compared resource allocation strategies, upcoming trends and open challenges are identified.

The research directions highlighted in this survey are expected to advance in the future to tackle the identified open challenges. These advances will require us to develop efficient resource allocation mechanisms to cater to the needs of future hard and soft real-time multi-/many-core systems.

REFERENCES

Waheed Ahmed, Muhammad Shafique, Lars Bauer, and Jörg Henkel. 2011. Adaptive resource management for simultaneous multitasking in mixed-grained reconfigurable multi-core processors. In *Proceedings of IEEE/ACM/IFIP Conference on Hardware/Software Codesign and System Synthesis (ISSS+CODES)*. ACM, 365–374.

B. Aksanli and T. Rosing. 2014. Providing regulation services and managing data center peak power budgets. In *Proceedings of IEEE Conference on Design, Automation and Test in Europe (DATE)*. 1–4.

Mohammad Abdullah Al Faruque, Rudolf Krist, and Jörg Henkel. 2008. ADAM: Run-time agent-based distributed application mapping for on-chip communication. In *Proceedings of ACM Design Automation Conference (DAC)*. 760–765.

AMD. 2011. AMD Opteron 6000 series processors. Retrieved February 12, 2016 from http://www.amd.com/en-us/products/server/opteron/6000.

F. Angiolini, Jianjiang Ceng, R. Leupers, F. Ferrari, C. Ferri, and L. Benini. 2006. An integrated open framework for heterogeneous MPSoC design space exploration. In *Proceedings of IEEE Conference on Design, Automation and Test in Europe (DATE)*. 1–6.

N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings. 1993. Applying new scheduling theory to static priority pre-emptive scheduling. *Softw. Eng. J.* 8, 5 (Sep 1993), 284–292.

Neil C. Audsley, Alan Burns, Robert I. Davis, Ken W. Tindell, and Andy J. Wellings. 1995. Fixed priority pre-emptive scheduling: An historical perspective. *Real-Time Syst.* 8, 2–3 (1995), 173–198.

Ozalp Babaoglu, Geoffrey Canright, Andreas Deutsch, Gianni A. Di Caro, Frederick Ducatelle, Luca M. Gambardella, Niloy Ganguly, Márk Jelasity, Roberto Montemanni, Alberto Montresor, et al. 2006. Design patterns from biology for distributed computing. *ACM Trans. Auton. Adapt. Syst.* 1, 1 (2006), 26–66.

Theodore P. Baker and Sanjoy K. Baruah. 2007. Schedulability analysis of multiprocessor sporadic task systems. *Handbook of Real-Time and Embedded Systems* (2007). CRC Press.

Mohamed A. Bamakhrama and Todor Stefanov. 2012. Managing latency in embedded streaming applications under hard-real-time scheduling. In *Proceedings of IEEE/ACM/IFIP Conference on Hardware/Software Codesign and System Synthesis (ISSS+CODES)*. 83–92.

Nikhil Bansal and Kirk R. Pruhs. 2010. Server scheduling to balance priorities, fairness, and average quality of service. *SIAM J. Comput.* 39, 7 (2010), 3311–3335.

Donato Barbagallo, Elisabetta Di Nitto, Daniel J Dubois, and Raffaela Mirandola. 2010. A bio-inspired algorithm for energy optimization in a self-organizing data center. In *Self-Organizing Architectures*. Springer, 127–151.

Sanjoy Baruah, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, and Sebastian Stiller. 2010. Improved multiprocessor global schedulability analysis. *Real-Time Syst.* 46, 1 (2010), 3–24.

Daniel Bates, Alex Bradbury, Andreas Koltes, and Robert Mullins. 2015. Exploiting tightly-coupled cores. *J. Sign. Process. Syst.* 80, 1 (2015), 103–120.

L. Benini and G. De Micheli. 2002. Networks on chips: A new SoC paradigm. *Computer* 1 (2002), 70–78.

M. Bertogna, M. Cirinei, and G. Lipari. 2005. Improved schedulability analysis of EDF on multiprocessor platforms. In *Proceedings of IEEE Euromicro Conference on Real-Time Systems (ECTRS)*. 209–218.

Benjamin Betting and Uwe Brinkschulte. 2014. Analyzing the overhead of self-optimization through task migration within a decentralized task control mechanism for dependable system-on-chip architectures. In *Proceedings of the IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC)*. IEEE, 84–91.

Tobias Bjerregaard and Shankar Mahadevan. 2006. A survey of research and practices of network-on-chip. *ACM Comput. Surv.* 1 (2006).

Carolina Blanch, Rogier Baert, Paul Coene, Maja D'Hondt, Zhe Ma, and Roel Wuyts. 2011. Runtime scheduling for video decoding on heterogeneous architectures. In *Proceedings of the International Conference on Real Time and Networks Systems (RTNS)*. Citeseer, 195–204.

B. Bohnenstiehl, A. Stillmaker, J. Pimentel, T. Andreas, B. Liu, A. Tran, E. Adeagbo, and B. Baa. 2016. A 5.8 pJ/Op 115 billion Ops/sec, to 1.78 trillion Ops/sec 32nm 1000 processor array. In *IEEE Symposia on VLSI Technology and Circuits*.

Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, Sebastian Stiller, and Andreas Wiese. 2013. Feasibility analysis in the sporadic DAG task model. In *Proceedings of IEEE Euromicro Conference on Real-Time Systems (ECRTS)*. IEEE, 225–233.

Shekhar Borkar. 2007. Thousand core chips: A technology perspective. In *Proceedings of ACM Design Automation Conference (DAC)*. 746–749.

Eduardo Wenzel Briáo, Daniel Barcelos, and Flávio Rech Wagner. 2008. Dynamic task allocation strategies in MPSoC for soft real-time applications. In *Proceedings of IEEE Conference on Design, Automation and Test in Europe (DATE)*. 1386–1389.

Uwe Brinkschulte, Mathias Pacher, and Alexander Von Renteln. 2007. Towards an artificial hormone system for self-organizing real-time task allocation. In *Software Technologies for Embedded and Ubiquitous Systems*. Springer, 339–347.

Andrew Marc Burkimsher. 2014. *Fair, Responsive Scheduling of Engineering Workflows on Computing Grids*. Ph.D. Dissertation. University of Southampton, UK.

A. Burns, D. Prasad, A. Bondavalli, F. Di Giandomenico, K. Ramamritham, J. Stankovic, and L. Strigini. 2000. The meaning and role of value in scheduling flexible real-time systems. *J. Syst. Arch.* 46, 4 (2000), 305–325.

Giorgio Buttazzo, Enrico Bini, and Yifan Wu. 2011. Partitioning real-time applications over multicore reservations. *IEEE Trans. Industr. Inform.* 7, 2 (2011), 302–315.

Giorgio C. Buttazzo. 2011. *Hard Real-Time Computing Systems*. Real-Time Systems Series, Vol. 24. Springer.

Rodrigo N. Calheiros and Rajkumar Buyya. 2014. Energy-efficient scheduling of urgent bag-of-tasks applications in clouds through DVFS. In *Proceedings of IEEE International Conference on Cloud Computing Technology and Science (CLOUDCOM)*. 342–349.

Scott Camazine, Nigel R. Franks, James Sneyd, Eric Bonabeau, Jean-Louis Deneubourg, and Guy Theraula. 2001. *Self-Organization in Biological Systems*. Princeton University Press, Princeton, NJ.

Ewerson Luiz de Souza Carvalho, Ney Laert Vilar Calazans, and Fernando Gehm Moraes. 2010. Dynamic task mapping for MPSoCs. *IEEE Des. Test* (2010), 26–35.

Guilherme Castilhos, Marcelo Mandelli, Guilherme Madalozzo, and Filipe Moraes. 2013. Distributed resource management in NoC-based MPSoCs with dynamic cluster sizes. In *Proceedings of IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 153–158.

Jeronimo Castrillon, Andreas Tretter, Rainer Leupers, and Gerd Ascheid. 2012. Communication-aware mapping of KPN applications onto heterogeneous MPSoCs. In *Proceedings of ACM Design Automation Conference (DAC)*. 1266–1271.

J. Ceng, J. Castrillon, W. Sheng, H. Scharwächter, R. Leupers, G. Ascheid, H. Meyr, T. Isshiki, and H. Kunieda. 2008. MAPS: An integrated framework for MPSoC application parallelization. In *Proceedings of ACM Design Automation Conference (DAC)*. 754–759.

Muhammad Tayyab Chaudhry, Teck Chaw Ling, Atif Manzoor, Syed Asad Hussain, and Jongwon Kim. 2015. Thermal-aware scheduling in green data centers. *ACM Comput. Surv.* 47, 3 (2015), 39.

Ken Chen and Paul Muhlethaler. 1996. A scheduling algorithm for tasks described by time value function. *Real-Time Syst.* 10, 3 (1996), 293–312.

Liang Chen, T. Marconi, and T. Mitra. 2012. Online scheduling for multi-core shared reconfigurable fabric. In *Proceedings of IEEE Conference on Design, Automation and Test in Europe (DATE)*. 582–585.

Yuanqing Cheng, Lei Zhang, Yinhe Han, and Xiaowei Li. 2013. Thermal-constrained task allocation for interconnect energy reduction in 3-D homogeneous MPSoCs. *IEEE Trans. VLSI* 21, 2 (2013), 239–249.

Junchul Choi, Hyunok Oh, Sungchan Kim, and Soonhoi Ha. 2012. Executing synchronous dataflow graphs on a SPM-based multicore architecture. In *Proceedings of ACM Design Automation Conference (DAC)*. 664–671.

Chen-Ling Chou and Radu Marculescu. 2008. User-aware dynamic task allocation in networks-on-chip. In *Proceedings of IEEE Conference on Design, Automation and Test in Europe (DATE)*. 1232–1237.

Chen-Ling Chou, U. Y. Ogras, and R. Marculescu. 2008. Energy- and performance-aware incremental mapping for networks on chip with multiple voltage levels. *Trans. Comp.-Aided Des. Integ. Cir. Sys.* (Oct. 2008), 1866–1879.

Pravanjan Choudhury, P. P. Chakrabarti, and Rajeev Kumar. 2007. Online dynamic voltage scaling using task graph mapping analysis for multiprocessors. In *Proceedings of the International Conference on VLSI Design (VLSID)*. 89–94.

Jason Cong and Karthik Gururaj. 2009. Energy efficient multiprocessor task scheduling under input-dependent variation. In *Proceedings of IEEE Conference on Design, Automation and Test in Europe (DATE)*. 411–416.

Ayse K. Coskun, Jose L. Ayala, David Atienza, Tajana Simunic Rosing, and Yusuf Leblebici. 2009. Dynamic thermal management in 3D multicore architectures. In *Proceedings of IEEE Conference on Design, Automation and Test in Europe (DATE)*. 1410–1415.

Marco Cox, Amit Kumar Singh, Akash Kumar, and Henk Corporaal. 2013. Thermal-aware mapping of streaming applications on 3D Multi-Processor Systems. In *Proceedings of IEEE/ACM/IFIP Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia)*. 11–20.

Antoniel da Silva Rego, Joaquim Celestino, Andre Dos Santos, Eduardo Coelho Cerqueira, Anup Patel, and Mehdi Taghavi. 2012. BEE-C: A bio-inspired energy efficient cluster-based algorithm for data continuous dissemination in Wireless Sensor Networks. In *Proceedings of the IEEE International Conference on Networks (ICON)*. IEEE, 405–410.

Anup Das, Amit Kumar Singh, and Akash Kumar. 2013. Energy-aware dynamic reconfiguration of communication-centric applications for reliable MPSoCs. In *Proceedings of the IEEE International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*. 1–7.

Robert I. Davis and Alan Burns. 2011. A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv.* 43, 4, Article 35 (Oct. 2011), 44 pages.

Robert I. Davis, Attila Zabos, and Alan Burns. 2008. Efficient exact schedulability tests for fixed priority real-time systems. *IEEE Trans. Comput.* 57, 9 (2008), 1261–1276.

Benoît Dupont De Dinechin, Duco Van Amstel, Marc Poulhiès, and Guillaume Lager. 2014. Time-critical computing on a single-chip massively parallel processor. In *Proceedings of IEEE Conference on Design, Automation and Test in Europe (DATE)*. 1–6.

André de Matos Pedro, David Pereira, Luís Miguel Pinho, and Jorge Sousa Pinto. 2015. Logic-based schedulability analysis for compositional hard real-time embedded systems. *ACM SIGBED Rev.* 12, 1 (2015), 56–64.

O. Derin, D. Kabakci, and L. Fiorin. 2011. Online task remapping strategies for fault-tolerant network-on-chip multiprocessors. In *Proceedings of the IEEE/ACM Symposium on Networks on Chip (NoCS)*. 129–136.

Michael L Dertouzos and Aloysius Ka-Lau Mok. 1989. Multiprocessor online scheduling of hard-real-time tasks. *IEEE Trans. Softw. Eng.* 15, 12 (1989), 1497–1506.

Jiun-Hung Ding, Ya-Ting Chang, Zhou-dong Guo, Kuan-Ching Li, and Yeh-Ching Chung. 2014. An efficient and comprehensive scheduler on asymmetric multicore architecture systems. *J. Syst. Arch.* 60, 3 (2014), 305–314.

Piotr Dziurzanski, Amit Kumar Singh, and Leandro Soares Indrusiak. 2016a. Energy-aware resource allocation in multi-mode automotive applications with hard real-time constraints. In *Proceedings of the IEEE International Symposium on Real-Time Distributed Computing (ISORC)*. 100–107.

Piotr Dziurzanski, Amit Kumar Singh, and Leandro Soares Indrusiak. 2016b. Feedback-based admission control for hard real-time task allocation under dynamic workload on many-core systems. In *Proceedings of the International Conference on Architecture of Computing Systems (ARCS)*. 157–169.

Piotr Dziurzanski, Amit Kumar Singh, Leandro Soares Indrusiak, and Björn Saballus. 2015. Hard real-time guarantee of automotive applications during mode changes. In *Proceedings of the International Conference on Real Time and Networks Systems (RTNS)*. 161–170.

Thomas Ebi, David Kramer, Wolfgang Karl, and Jörg Henkel. 2011. Economic learning for thermal-aware power budgeting in many-core architectures. In *Proceedings of IEEE/ACM/IFIP Conference on Hardware/Software Codesign and System Synthesis (ISSS+CODES)*. 189–196.

M. Fattah, P. Liljeberg, J. Plosila, and H. Tenhunen. 2014a. Adjustable contiguity of run-time task allocation in networked many-core systems. In *Proceedings of IEEE Asia and South Pacific Design Automation Conference (ASP-DAC)*. 349–354.

Mohammad Fattah, Maurizio Palesi, Pasi Liljeberg, Juha Plosila, and Hannu Tenhunen. 2014b. SHiFA: System-level hierarchy in run-time fault-aware management of many-core systems. In *Proceedings of ACM Design Automation Conference (DAC)*. 101:1–101:6.

Fabrizio Fazzino, Maurizio Palesi, and David Patti. 2008. Noxim: Network-on-chip simulator. Retrieved from http://sourceforge. net/projects/noxim.

Michael R. Garey and David S. Johnson. 1979. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co.

Beltra Giovanni, Luca Fossati, and Donatella Sciuto. 2010. Decision-theoretic design space exploration of multiprocessor platforms. *Trans. Comp.-Aided Des. Integ. Cir. Sys.* (2010), 1083–1095. Issue 7.

Maximilian Götzinger, Amir M. Rahmani, Martin Pongratz, Pasi Liljeberg, Axel Jantsch, and Hannu Tenhunen. 2016. The role of self-awareness and hierarchical agents in resource management for many-core systems. In *IEEE International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*. IEEE, 53–60.

Dominik Grewe, Zheng Wang, and Michael F. P. OBoyle. 2013. OpenCL task partitioning in the presence of GPU contention. In *International Workshop on Languages and Compilers for Parallel Computing*. 87–101.

Abdul Hameed, Alireza Khoshkbarforoushha, Rajiv Ranjan, Prem Prakash Jayaraman, Joanna Kolodziej, Pavan Balaji, Sherali Zeadally, Qutaibah Marwan Malluhi, Nikos Tziritas, Abhinav Vishnu, and others. 2014. A survey and taxonomy on energy efficient resource allocation techniques for cloud computing systems. *Computing* (2014), 1–24.

Ching-Chih Han and Kwei-Jay Lin. 1989. Scheduling parallelizable jobs on multiprocessors. In *Proceedings Real Time Systems Symposium (RTSS)*. IEEE, 59–67.

Hans-Ulrich Heiss and Michael Schmitz. 1995. Decentralized dynamic load balancing: The particles approach. *Inf. Sci.* 84, 1 (1995), 115–128.

Jörg Henkel, Lars Bauer, Nikil Dutt, Puneet Gupta, Sani Nassif, Muhammad Shafique, Mehdi Tahoori, and Norbert Wehn. 2013. Reliable On-chip systems in the nano-era: Lessons learnt and future trends. In *Proceedings of ACM Design Automation Conference (DAC)*. 99:1–99:10.

J. Henkel, A. Herkersdorf, L. Bauer, T. Wild, M. Hubner, R. K. Pujari, A. Grudnitsky, J. Heisswolf, A. Zaib, B. Vogel, V. Lari, and S. Kobbe. 2012. Invasive manycore architectures. In *Proceedings of IEEE Asia and South Pacific Design Automation Conference (ASP-DAC)*. 193–200.

Fernando Herrera and Ingo Sander. 2013. Combining analytical and simulation-based design space exploration for time-critical systems. In *Forum on Specification Design Languages (FDL)*. 1–8.

Henry Hoffmann, Jim Holt, George Kurian, Eric Lau, Martina Maggio, Jason E. Miller, Sabrina M. Neuman, Mahmut Sinangil, Yildiz Sinangil, Anant Agarwal, Anantha P. Chandrakasan, and Srinivas Devadas. 2012. Self-aware computing in the angstrom processor. In *Proceedings of ACM Design Automation Conference (DAC)*. 259–264.

Shengyan Hong, T. Chantem, and Xiaobo Sharon Hu. 2015. Local-deadline assignment for distributed real-time systems. *IEEE Trans. Comput.* 64, 7 (July 2015), 1983–1997.

S. Hong, S. H. K. Narayanan, M. Kandemir, and Ö. Özturk. 2009. Process variation aware thread mapping for chip multiprocessors. In *Proceedings of IEEE Conference on Design, Automation and Test in Europe (DATE)*. 821–826.

Jingcao Hu and Radu Marculescu. 2003. Energy-aware mapping for tile-based NoC architectures under performance constraints. In *Proceedings of IEEE Asia and South Pacific Design Automation Conference (ASP-DAC)*. 233–239.

Jia Huang, Andreas Raabe, Christian Buckl, and Alois Knoll. 2011a. A workflow for runtime adaptive task allocation on heterogeneous mpsocs. In *Proceedings of IEEE Conference on Design, Automation and Test in Europe (DATE)*. 1–6.

Lin Huang and Qiang Xu. 2010. Performance yield-driven task allocation and scheduling for MPSoCs under process variation. In *Proceedings of ACM Design Automation Conference (DAC)*. 326–331.

Lin Huang, Rong Ye, and Qiang Xu. 2011b. Customer-aware task allocation and scheduling for multi-mode MPSoCs. In *Proceedings of ACM Design Automation Conference (DAC)*. 387–392.

Hameed Hussain, Saif Ur Rehman Malik, Abdul Hameed, Samee Ullah Khan, Gage Bickler, Nasro Min-Allah, Muhammad Bilal Qureshi, Limin Zhang, Wang Yongji, Nasir Ghani, Joanna Kolodziej, Albert Y. Zomaya, Cheng-Zhong Xu, Pavan Balaji, Abhinav Vishnu, Fredric Pinel, Johnatan E. Pecero, Dzmitry Kliazovich, Pascal Bouvry, Hongxiang Li, Lizhe Wang, Dan Chen, and Ammar Rayes. 2013. Review: A survey on resource allocation in high performance distributed computing systems. *Parallel Comput.* 39, 11 (Nov. 2013), 709–736.

Leandro Soares Indrusiak. 2014. End-to-end schedulability tests for multiprocessor embedded systems based on networks-on-chip with priority-preemptive arbitration. *J. Syst. Arch.* 60, 7 (2014), 553–561.

Leandro Soares Indrusiak, Piotr Dziurzanski, and Amit Kumar Singh. 2016. *Dynamic Resource Allocation in Embedded, High-Performance and Cloud Computing*. River Publishers.

Intel. 2016. Intel Hardware Accelerator Research Program. Retrieved from https://www.nextplatform.com/2016/03/14/intel-marrying-fpga-beefy-broadwell-open-compute-future/.

David E. Irwin, Laura E. Grit, and Jeffrey S. Chase. 2004. Balancing risk and reward in a market-based task service. In *Proceedings of the IEEE International Symposium on High Performance Distributed Computing (HPDC)*. 160–169.

Damir Isovic and Gerhard Fohler. 2004. Quality aware MPEG-2 stream adaptation in resource constrained systems. In *Proceedings of IEEE Euromicro Conference on Real-Time Systems (ECRTS)*. 23–32.

Ralf Jahr, Martin Frieb, Mike Gerdes, Theo Ungerer, Andreas Hugl, and Hans Regler. 2014. Paving the way for multi-cores in industrial hard real-time control applications. In *Proceedings of the IEEE International Symposium on Industrial Embedded Systems (SIES)*. 1–4.

Haris Javaid and Sri Parameswaran. 2009. A design flow for application specific heterogeneous pipelined multiprocessor systems. In *Proceedings of ACM Design Automation Conference (DAC)*. 250–253.

Haris Javaid, Muhammad Shafique, Jorg Henkel, and Sri Parameswaran. 2014. Energy-efficient adaptive pipelined MPSoCs for multimedia applications. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst.* 33, 5 (2014), 663–676.

Praveen Jayachandran and Tarek Abdelzaher. 2008. Delay composition in preemptive and non-preemptive real-time pipelines. *Real-Time Syst.* 40, 3 (2008), 290–320.

Ahmed Jerraya, Hannu Tenhunen, and Wayne Wolf. 2005. Guest editors' introduction: Multiprocessor systems-on-chips. *Computer* 7 (2005), 36–40.

Vaibhav Jha, Mohit Jha, and G. K. Sharma. 2014. Estimation of optimized energy and latency constraints for task allocation in 3d network on chip. *arXiv Preprint arXiv:1405.0109* (2014).

Zai Jian Jia, A. D. Pimentel, M. Thompson, T. Bautista, and A. Nunez. 2010. NASA: A generic infrastructure for system-level MP-SoC design space exploration. In *Proceedings of IEEE/ACM/IFIP Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia)*. 41–50.

Jan Jonsson and Kang G. Shin. 2002. Robust adaptive metrics for deadline assignment in distributed hard real-time systems. *Real-Time Syst.* 23, 3 (2002), 239–271.

Hanwoong Jung, Chanhee Lee, Shin-Haeng Kang, Sungchan Kim, Hyunok Oh, and Soonhoi Ha. 2014. Dynamic behavior specification and dynamic mapping for real-time embedded systems: HOPES approach. *ACM Trans. Embed. Comput. Syst.* 13, 4s, Article 135 (2014), 26 pages.

Michael Kadin, Sherief Reda, and Augustus Uht. 2009. Central vs. distributed dynamic thermal management for multi-core processors: Which one is better? In *Proceedings of ACM Great Lakes symposium on VLSI (GLSVLSI)*. 137–140.

B. Kao and H. Garcia-Molina. 1997. Deadline assignment in a distributed soft real-time system. *IEEE Trans. Parallel Distrib. Syst.* 8, 12 (Dec 1997), 1268–1274.

Samarth Kaushik, Amit Kumar Singh, Wu Jigang, and Thambipillai Srikanthan. 2011. Run-time computation and communication aware mapping heuristic for NoC-based heterogeneous MPSoC platforms. In *Proceedings of the IEEE International Symposium on Parallel Architectures, Algorithms and Programming (PAAP)*. 203–207.

Bhavesh Khemka, Ryan Friese, Sudeep Pasricha, Anthony A. Maciejewski, Howard Jay Siegel, Gregory A Koenig, Sarah Powers, Marcia Hilton, Rajendra Rambharos, and Steve Poole. 2015. Utility maximizing dynamic resource management in an oversubscribed energy-constrained heterogeneous computing system. *Sustainable Computing: Informatics and Systems* 5 (2015), 14–30.

Joonsoo Kim and Michael Orshansky. 2006. Towards formal probabilistic power-performance design space exploration. In *Proceedings of ACM Great Lakes symposium on VLSI (GLSVLSI)*. 229–234.

Sebastian Kobbe, Lars Bauer, Daniel Lohmann, Wolfgang Schröder-Preikschat, and Jörg Henkel. 2011. DistRM: Distributed resource management for on-chip many-core systems. In *Proceedings of IEEE/ACM/IFIP Conference on Hardware/Software Codesign and System Synthesis (ISSS+CODES)*. 119–128.

Jonathan Koomey. 2011. Growth in data center electricity use 2005 to 2010. A report by Analytical Press, completed at the request of The New York Times (2011).

Akash Kumar, Henk Corporaal, Bart Mesman, and Yajun Ha. 2010. *Multimedia Multiprocessor Systems: Analysis, Design and Management*. Springer Science & Business Media.

Tei-Wei Kuo, Li-Pin Chang, Yu-Hua Liu, and Kwei-Jay Lin. 2003. Efficient online schedulability tests for real-time systems. *IEEE Trans. Softw. Eng.* 29, 8 (2003), 734–751.

Yu-Kwong Kwok et al. 2006. A semi-static approach to mapping dynamic iterative tasks onto heterogeneous computing systems. *J. Parallel Distrib. Comput.* 66, 1 (2006), 77–98.

Karthik Lakshmanan, Dionisio de Niz, and Ragunathan Rajkumar. 2009. Coordinated task scheduling, allocation and synchronization on multiprocessors. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*. 469–478.

Sylvain Lauzac, Rami Melhem, and Daniel Mosse. 1998. An efficient RMS admission control and its application to multiprocessor scheduling. In *Proceedings of the IEEE Symposium on Parallel and Distributed Processing (SPDP)*. 511–518.

Chanhee Lee, Hokeun Kim, Hae-woo Park, Sungchan Kim, Hyunok Oh, and Soonhoi Ha. 2010. A task remapping technique for reliable multi-core embedded systems. In *Proceedings of IEEE/ACM/IFIP Conference on Hardware/Software Codesign and System Synthesis (ISSS+CODES)*. 307–316.

Chanhee Lee, Sungchan Kim, and Soonhoi Ha. 2013. Efficient run-time resource management of a manycore accelerator for stream-based applications. In *Proceedings of IEEE/ACM/IFIP Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia)*. 51–60.

Jinkyu Lee, Insik Shin, and Arvind Easwaran. 2012. Convex optimization framework for intermediate deadline assignment in soft and hard real-time distributed systems. *J. Syst. Softw.* 85, 10 (2012), 2331–2339.

Hennadiy Leontyev and James H Anderson. 2008. A unified hard/soft real-time schedulability test for global EDF multiprocessor scheduling. In *IEEE Real-Time Systems Symposium (RTSS)*. 375–384.

Yu-Chia Lin, Chuan-Yue Yang, Che-Wei Chang, Yuan-Hao Chang, Tei-Wei Kuo, and Chi-Sheng Shih. 2010. Energy-efficient mapping technique for virtual cores. In *Proceedings of IEEE Euromicro Conference on Real-Time Systems (ECRTS)*. 66–75.

Michele Lombardi and Michela Milano. 2012. Optimal methods for resource allocation and scheduling: a cross-disciplinary survey. *Constraints* 17, 1 (2012), 51–85.

Jose M. Lopez, Jose L. Diaz, and Daniel F. Garcia. 2004. Minimum and maximum utilization bounds for multiprocessor rate monotonic scheduling. *IEEE Trans. Parallel Distrib. Syst.* 15, 7 (2004), 642–653.

Chenyang Lu, John A. Stankovic, Sang H. Son, and Gang Tao. 2002. Feedback control real-time scheduling: Framework, modeling, and algorithms*. *Real-Time Syst.* 23, 1–2 (2002), 85–126.

Chi-Keung Luk, Sunpyo Hong, and Hyesoon Kim. 2009. Qilin: Exploiting parallelism on heterogeneous multiprocessors with adaptive mapping. In *Proceedings of IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 45–55.

A. Mallik et al. 2011. MNEMEE - An automated toolflow for parallelization and memory management in MPSoC platforms. In *Proceedings of ACM Design Automation Conference (DAC)*.

Marcelo Mandelli, Luciano Ost, Everton Carara, Guilherme Guindani, Thiago Gouvea, Guilherme Medeiros, and Fernando G. Moraes. 2011. Energy-aware dynamic task mapping for NoC-based MPSoCs. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*. 1676–1679.

G. Manimaran, C. Siva Ram Murthy, and Krithi Ramamritham. 1998. A new approach for scheduling of parallelizable tasks in real-time multiprocessor systems. *Real-Time Syst.* 15, 1 (1998), 39–60.

C. A. M. Marcon, E. I. Moreno, N. L. V. Calazans, and F. G. Moraes. 2008. Comparison of network-on-chip mapping algorithms targeting low energy consumption. *IET Comput. Dig. Techn.* (2008), 471–482.

R. Marculescu, U. Y. Ogras, Li-Shiuan Peh, N. E. Jerger, and Y. Hoskote. 2009. Outstanding research problems in NoC design: System, microarchitecture, and circuit perspectives. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst.* 1 (2009), 3–21.

G. Mariani, P. Avasare, G. Vanmeerbeeck, C. Ykman-Couvreur, G. Palermo, C. Silvano, and V. Zaccaria. 2010. An industrial design space exploration framework for supporting run-time resource management on multi-core systems. In *Proceedings of IEEE Conference on Design, Automation and Test in Europe (DATE)*. 196–201.

Dana Marinca, Pascale Minet, and Laurent George. 2004. Analysis of deadline assignment methods in distributed real-time systems. *Comput. Commun.* 27, 15 (2004), 1412–1423.

G. Martin. 2006. Overview of the MPSoC design challenge. In *Proceedings of ACM Design Automation Conference (DAC)*. 274–279.

Peter Marwedel, Jürgen Teich, Georgia Kouveli, Iuliana Bacivarov, Lothar Thiele, Soonhoi Ha, Chanhee Lee, Qiang Xu, and Lin Huang. 2011. Mapping of applications to MPSoCs. In *Proceedings of IEEE/ACM/IFIP Conference on Hardware/Software Codesign and System Synthesis (ISSS+CODES)*. 109–118.

MediaTek. 2016. MediaTek's Helio X20 Chip. Retrieved from http://www.mediatek.com/products/mediatek-helio.

A. Mehran, A. Khademzadeh, and S. Saeidi. 2008. DSM: A heuristic dynamic spiral mapping algorithm for network on chip. *IEICE Electron. Expr.* 13 (2008), 464–471.

A. Melani, M. Bertogna, V. Bonifaci, A. Marchetti-Spaccamela, and G. C. Buttazzo. 2015. Response-time analysis of conditional DAG tasks in multiprocessor systems. In *Proceedings of IEEE Euromicro Conference on Real-Time Systems (ECRTS)*. 211–221.

Hashan Roshantha Mendis, Leandro Soares Indrusiak, and Neil C. Audsley. 2014. Predictability and utilisation trade-off in the dynamic management of multiple video stream decoding on network-on-chip based homogeneous embedded multi-cores. In *Proceedings of the International Conference on Real-Time Networks and Systems (RTNS)*. 161–170.

Hashan R. Mendis, Leandro Soares Indrusiak, and Neil C. Audsley. 2015. Bio-inspired distributed task remapping for multiple video stream decoding on homogeneous NoCs. In *Proceedings of IEEE/ACM/IFIP Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia)*. 1–10.

Brett H. Meyer, Adam S. Hartman, and Donald E. Thomas. 2010. Cost-effective slack allocation for lifetime improvement in NoC-based MPSoCs. In *Proceedings of IEEE Conference on Design, Automation and Test in Europe (DATE)*. 1596–1601.

Mehdi Modarressi, Marjan Asadinia, and Hamid Sarbazi-Azad. 2013. Using task migration to improve non-contiguous processor allocation in NoC-based CMPs. *J. Syst. Arch.* 59, 7 (2013), 468–481.

S. Mohanty, V. K. Prasanna, S. Neema, and J. Davis. 2002. Rapid design space exploration of heterogeneous embedded systems using symbolic search and multi-granular simulation. *SIGPLAN Not.* 37, 7 (2002), 18–27.

Orlando Moreira, Jan-David Mol, Marco Bekooij, and Jef Van Meerbergen. 2005. Multiprocessor resource allocation for hard-real-time streaming with a dynamic job-mix. In *Proceedings of the IEEE Real Time and Embedded Technology and Applications Symposium (RTAS)*. 332–341.

Orlando Moreira, Jacob Jan-David Mol, and Marco Bekooij. 2007. Online resource management in a multiprocessor with a network-on-chip. In *Proceedings of ACM Symposium on Applied Computing (SAC)*. 1557–1564.

Orlando Moreira, Frederico Valente, and Marco Bekooij. 2007. Scheduling multiple independent hard-real-time jobs on a heterogeneous multiprocessor. In *Proceedings of ACM International Conference on Embedded Software (EMSOFT)*. 57–66.

Pierre-André Mudry and Gianluca Tempesti. 2009. Self-scaling stream processing: A bio-inspired approach to resource allocation through dynamic task replication. In *Proceedings of the IEEE NASA / ESA Conference on Adaptive Hardware and Systems (AHS)*. 353–360.

Srinivasan Murali, Martijn Coenen, Andrei Radulescu, Kees Goossens, and Giovanni De Micheli. 2006. A methodology for mapping multiple use-cases onto networks on chips. In *Proceedings of IEEE Conference on Design, Automation and Test in Europe (DATE)*. 118–123.

Srinivasan Murali, Almir Mutapcic, David Atienza, Rajesh Gupta, Stephen Boyd, Luca Benini, and Giovanni De Micheli. 2008. Temperature control of high-performance multi-core platforms using convex optimization. In *Proceedings of IEEE Conference on Design, Automation and Test in Europe (DATE)*. 110–115.

Sasmita Kumari Nayak, Sasmita Kumari Padhy, and Siba Prasada Panigrahi. 2012. A novel algorithm for dynamic task scheduling. *Future Gen. Comput. Syst.* 28, 5 (2012), 709–717.

J. Ng, X. Wang, A. K. Singh, and T. Mak. 2015. DeFrag: Defragmentation for efficient runtime resource allocation in NoC-based many-core systems. In *Proceedings of the Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*. 345–352.

T. Nishitha and P. Chenna Reddy. 2012. Performance evaluation Of AntHocNet routing algorithm in Ad Hoc networks. In *Proceedings of the IEEE International Conference on Computing Sciences (ICCS)*. 207–211.

Vincent Nollet, Prabhat Avasare, Hendrik Eeckhaut, Diederik Verkest, and Henk Corporaal. 2008. Run-time management of a MPSoC containing FPGA fabric tiles. *IEEE Trans. VLSI Syst.* (2008), 24–33.

OpenCL. 2016. Open Computing Language (OpenCL) - The open standard for parallel programming of heterogeneous systems. Retrieved from https://goo.gl/A9wXRJ.

Luciano Ost, Marcelo Mandelli, Gabriel Marchesan Almeida, Leandro Moller, Leandro Soares Indrusiak, Gilles Sassatelli, Pascal Benoit, Manfred Glesner, Michel Robert, and Fernando Moraes. 2013. Power-aware dynamic mapping heuristics for NoC-based MPSoCs using a unified model-based approach. *ACM Trans. Embed. Comput. Syst.* 12, 3 (2013), 75:1–75:22.

Z. Peter, S. Gilles, U. Nurten, S. J. Nicolas, B. Pascal, and G. Manfred. 2009. A decentralised task mapping approach for homogeneous multiprocessor network-on-chips. *Int. J. Reconfig. Comput.* (2009), 3:1–3:14.

R. Piscitelli and A. D. Pimentel. 2012. Design space pruning through hybrid analysis in system-level design space exploration. In *Proceedings of IEEE Conference on Design, Automation and Test in Europe (DATE)*. 781–786.

Ruxandra Pop and Shashi Kumar. 2004. *A survey of techniques for mapping and scheduling applications to network on chip systems*. School of Engineering, Jonkoping University, Research Report 4 (2004), 4.

Alok Prakash, Siqi Wang, Alexandru Eugen Irimiea, and Tulika Mitra. 2015. Energy-efficient execution of data-parallel applications on heterogeneous mobile platforms. In *Proceedings of IEEE International Conference on Computer Design (ICCD)*. 208–215.

Wei Quan and Andy D. Pimentel. 2015. A hybrid task mapping algorithm for heterogeneous MPSoCs. *ACM Trans. Embed. Comput. Syst.* 14, 1 (2015), 14:1–14:25.

Wei Quan and Andy D. Pimentel. 2016. A hierarchical run-time adaptive resource allocation framework for large-scale MPSoC systems. *Des. Autom. Embed. Syst.* 20, 4 (2016), 311–339.

Juan Maria Rivas, José Javier Gutiérrez García, José C. Palencia Gutiérrez, and Michael González Harbour. 2010. Optimized deadline assignment and schedulability analysis for distributed real-time systems with local EDF scheduling. In *ESA*. 150–156.

Matthew Rowlings, Andy Tyrrell, and Martin Trefzer. 2015. Social-insect-inspired networking for autonomous load optimisation. *Proc. CIRP* 38 (2015), 259–264.

Siva Satyendra Sahoo, Akash Kumar, and Bharadwaj Veeravalli. 2016. Design and evaluation of reliability-oriented task re-mapping in MPSoCs using time-series analysis of intermittent faults. In *Proceedings of IEEE Conference on Design, Automation and Test in Europe (DATE)*. 798–803.

Pradip Kumar Sahu and Santanu Chattopadhyay. 2013. A survey on application mapping strategies for network-on-chip design. *J. Syst. Arch.* 59, 1 (2013), 60–76.

A. Saifullah, K. Agrawal, Chenyang Lu, and C. Gill. 2011. Multi-core real-time scheduling for generalized parallel task models. In *IEEE Real-Time Systems Symposium (RTSS)*. 217–226.

Manas Saksena and Seongsoo Hong. 1996. An engineering approach to decomposing end-to-end delays on a distributed real-time system. In *IEEE International Workshop on Parallel and Distributed Real-Time Systems*. 244–251.

Samsung. 2014. Samsung Exynos 5422. (2014). Retrieved from www.samsung.com/exynos/.

Lars Schor, Iuliana Bacivarov, Devendra Rai, Hoeseok Yang, Shin-Haeng Kang, and Lothar Thiele. 2012. Scenario-based design flow for mapping streaming applications onto on-chip many-core systems. In *Proceedings of ACM Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES)*. 71–80.

Andreas Schranzhofer, Jian-Jia Chen, and Lothar Thiele. 2009. Power-aware mapping of probabilistic applications onto heterogeneous MPSoC platforms. In *IEEE Real Time and Embedded Technology and Applications Symposium (RTAS)*. 151–160.

A. Schranzhofer, Jian-Jian Chen, and L. Thiele. 2010. Dynamic power-aware mapping of applications onto heterogeneous MPSoC platforms. *IEEE Trans. Industr. Inf.* 4 (2010), 692–707.

Nicola Serreli, Giuseppe Lipari, and Enrico Bini. 2009. Deadline assignment for component-based analysis of real-time transactions. In *Workshop on Compositional Real-Time Systems*. Citeseer.

Hafiz Fahad Sheikh and Ishfaq Ahmad. 2014. Efficient heuristics for joint optimization of performance, energy, and temperature in allocating tasks to multi-core processors. In *IEEE International Green Computing Conference (IGCC)*. 1–8.

Hamid Shojaei, AmirHossein Ghamarian, Twan Basten, Marc Geilen, Sander Stuijk, and Rob Hoes. 2009. A parameterized compositional multi-dimensional multiple-choice knapsack heuristic for CMP run-time management. In *Proceedings of ACM Design Automation Conference (DAC)*. 917–922.

Amit Kumar Singh, Anup Das, and Akash Kumar. 2013a. Energy optimization by exploiting execution slacks in streaming applications on multiprocessor systems. In *Proceedings of ACM Design Automation Conference (DAC)*. Article 115, 7 pages.

Amit Kumar Singh, Anup Das, and Akash Kumar. 2013b. RAPIDITAS: RAPId design-space-exploration incorporating trace-based analysis and simulation. In *Proceedings of IEEE Euromicro Conference on Digital System Design (DSD)*. 836–843.

Amit Kumar Singh, Piotr Dziurzanski, and Leandro Soares Indrusiak. 2015a. Market-inspired dynamic resource allocation in many-core high-performance computing systems. In *IEEE International Conference on High Performance Computing & Simulation (HPCS)*. 413–420.

Amit Kumar Singh, Piotr Dziurzanski, and Leandro Soares Indrusiak. 2015b. Value and energy optimizing dynamic resource allocation in many-core HPC systems. In *IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. 180–185.

Amit Kumar Singh, Piotr Dziurzanski, and Leandro Soares Indrusiak. 2016. Value and energy aware adaptive resource allocation of soft real-time jobs on many-core HPC data centers. In *IEEE International Symposium on Real-Time Computing (ISORC)*.

Amit Kumar Singh, Akash Kumar, and Thambipillai Srikanthan. 2013a. Accelerating throughput-aware runtime mapping for heterogeneous MPSoCs. *ACM Trans. Des. Autom. Electron. Syst.* 18, 1, Article 9 (January 2013), 1–29.

Amit Kumar Singh, Muhammad Shafique, Akash Kumar, and Jörg Henkel. 2013b. Mapping on multi-/many-core systems: Survey of current and emerging trends. In *Proceedings of ACM Design Automation Conference (DAC)*. Article 1.

A. K. Singh, M. Shafique, A. Kumar, and J. Henkel. 2016a. Analysis and mapping for thermal and energy efficiency of 3-D video processing on 3-D multicore processors. *IEEE Trans. VLSI* 24, 8 (2016), 2745–2758.

Amit Kumar Singh, Muhammad Shafique, Akash Kumar, and Jorg Henkel. 2016b. Resource and throughput aware execution trace analysis for efficient run-time mapping on MPSoCs. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst.* 35, 1 (2016), 72–85.

L. T. Smit, G. J. M. Smit, J. L. Hurink, H. Broersma, D. Paulusma, and P. T. Wolkotte. 2004. Run-time mapping of applications to a heterogeneous reconfigurable tiled system on chip architecture. In *Proceedings of IEEE International Conference on Field-Programmable Technology (FPT)*. 421–424.

John A. Stankovic, Chenyang Lu, Sang H. Son, and Gang Tao. 1999. The case for feedback control real-time scheduling. In *Proceedings of IEEE Euromicro Conference on Real-Time Systems (ECRTS)*. 11–20.

S. Stuijk, M. C. W. Geilen, and T. Basten. 2006. SDF^3: SDF For Free. In *Proceedings of IEEE Conference on Application of Concurrency to System Design (ACSD)*. 276–278.

Sander Stuijk, Marc Geilen, and Twan Basten. 2010. A predictable multiprocessor design flow for streaming applications with dynamic behaviour. In *Proceedings of IEEE Euromicro Conference on Digital System Design (DSD)*. 548–555.

Guang Sun, Yong Li, Yuanyuan Zhang, Li Su, Depeng Jin, and Lieguang Zeng. 2010. Energy-aware run-time mapping for homogeneous NoC. In *IEEE International Symposium on System on Chip (SoC)*. 8–11.

Timon D. ter Braak, Philip K. F. Hölzenspies, Jan Kuper, Johann L. Hurink, and Gerard J. M. Smit. 2010. Run-time spatial resource management for real-time applications on heterogeneous MPSoCs. In *Proceedings of IEEE Conference on Design, Automation and Test in Europe (DATE)*. 357–362.

Theocharis Theocharides, Maria K. Michael, Marios Polycarpou, and Ajit Dingankar. 2009. Towards embedded runtime system level optimization for MPSoCs: On-chip task allocation. In *Proceedings of ACM Great Lakes symposium on VLSI (GLSVLSI)*. 121–124.

Theocharis Theocharides, Maria K. Michael, Marios Polycarpou, and Ajit Dingankar. 2010. Hardware-enabled dynamic resource allocation for manycore systems using bidding-based system feedback. *EURASIP J. Embedded Syst.* 2010, Article 3 (2010), 21 pages.

Lothar Thiele, Lars Schor, Hoeseok Yang, and Iuliana Bacivarov. 2011. Thermal-aware system analysis and software synthesis for embedded multi-processors. In *Proceedings of ACM Design Automation Conference (DAC)*. 268–273.

TILE-Gx. 2009. First 100-core Processor with the New TILE-Gx Family. Retrieved February 12, 2016 http://www.tilera.com/.

Stavros Tzilis, Ioannis Sourdis, Vasileios Vasilikos, Dimitrios Rodopoulos, and Dimitrios Soudris. 2016. Runtime management of adaptive MPSoCs for graceful degradation. In *Proceedings of ACM Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES)*. 1–10.

S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob, S. Jain, S. Venkataraman, Y. Hoskote, and N. Borkar. 2007. An 80-Tile 1.28TFLOPS network-on-chip in 65nm CMOS. In *Proceedings of IEEE International Solid-State Circuits Conference (ISSCC)*. 98–589.

Feng Wang, Yibo Chen, C. Nicopoulos, X. Wu, Y. Xie, and N. Vijaykrishnan. 2011. Variation-aware task and communication mapping for MPSoC architecture. *IEEE Trans. Comput.-Aid. Des. Integr. Cir. Syst.* 2 (2011), 295–307.

Shengquan Wang and Jian-Jia Chen. 2010. Thermal-aware lifetime reliability in multicore systems. In *Proceedings of International Symposium on Quality Electronic Design (ISQED)*. 399–405.

Xiaohang Wang, Baoxin Zhao, Terrence Mak, Mei Yang, Yingtao Jiang, and Masoud Daneshtalab. 2015. An efficient runtime power allocation scheme for many-core systems inspired from auction theory. *VLSI J.* 50 (2015), 147–157.

Andreas Weichslgartner, Deepak Gangadharan, Stefan Wildermann, Michael Glaß, and Jürgen Teich. 2014. DAARM: Design-time application analysis and run-time mapping for predictable execution in many-core systems. In *Proceedings of IEEE/ACM/IFIP Conference on Hardware/Software Codesign and System Synthesis (ISSS+CODES)*. 34:1–34:10.

Stefan Wildermann, Felix Reimann, Daniel Ziener, and Jürgen Teich. 2011. Symbolic design space exploration for multi-mode reconfigurable systems. In *Proceedings of IEEE/ACM/IFIP Conference on Hardware/Software Codesign and System Synthesis (ISSS+CODES)*. 129–138.

Dong Hyuk Woo and Hsien-Hsin S. Lee. 2008. Extending Amdahl's law for energy-efficient computing in the many-core era. *Computer* 41, 12 (2008).

Frédéric Worm, Paolo Ienne, Patrick Thiran, and Giovanni De Micheli. 2002. An adaptive low-power transmission scheme for on-chip networks. In *Proceedings of IEEE/ACM/IFIP Conference on Hardware/Software Codesign and System Synthesis (ISSS+CODES)*. 92–100.

Xilinx. 2016. Zynq UltraScale+ EG MPSoC. Retrieved from https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html.

Liping Xue, Ozcan ozturk, Feihui Li, Mahmut Kandemir, and I. Kolcu. 2006. Dynamic partitioning of processing and memory resources in embedded MPSoC architectures. In *Proceedings of IEEE Conference on Design, Automation and Test in Europe (DATE)*. 690–695.

Peng Yang, Paul Marchal, Chun Wong, Stefaan Himpe, Francky Catthoor, Patrick David, Johan Vounckx, and Rudy Lauwereins. 2002. Managing dynamic concurrent tasks in embedded real-time multimedia systems. In *Proceedings of IEEE/ACM/IFIP Conference on Hardware/Software Codesign and System Synthesis (ISSS+CODES)*. 112–119.

Chee Shin Yeo and Rajkumar Buyya. 2006. A taxonomy of market-based resource management systems for utility-driven cluster computing. *Softw. Pract. Exper.* 36, 13 (Nov. 2006), 1381–1419.

C. Ykman-Couvreur, P. Avasare, G. Mariani, G. Palermo, C. Silvano, and V. Zaccaria. 2011. Linking run-time resource management of embedded multi-core platforms with automated design-time exploration. *IET Comput. Dig. Techn.* 2 (2011), 123–135.

Nicholas H. Zamora, Xiaoping Hu, and Radu Marculescu. 2007. System-level performance/power analysis for platform-based design of multimedia applications. *ACM Trans. Des. Autom. Electron. Syst.* 2, 1 (2007).

Luyuan Zeng, Pengcheng Huang, and Lothar Thiele. 2016. Towards the design of fault-tolerant mixed-criticality systems on multicores. In *Proceedings of ACM Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES)*. 6:1–6:10.

Xiuyi Zhou, Jun Yang, Yi Xu, Youtao Zhang, and Jianhua Zhao. 2010. Thermal-aware task scheduling for 3D multicore processors. *IEEE Trans. Parallel Distrib. Syst.* (2010), 60–71.

Yifan Zhu and Frank Mueller. 2005. Feedback EDF scheduling of real-time tasks exploiting dynamic voltage scaling. *Real-Time Syst.* 31, 1-3 (2005), 33–63.

Sergey Zhuravlev, Juan Carlos Saez, Sergey Blagodurov, Alexandra Fedorova, and Manuel Prieto. 2012. Survey of scheduling techniques for addressing shared resources in multicore processors. *ACM Comput. Surv.* 45, 1 (2012).

Received August 2016; revised February 2017; accepted February 2017