# HDGM: Hierarchical Dynamic Goal Management for Many-Core Resource Allocation

Amir M. Rahmani, *Senior Member, IEEE,* Axel Jantsch, *Member, IEEE,* and Nikil Dutt, *Fellow, IEEE*

***Abstract*—Many-core systems are highly complex and require thorough orchestration of different goals across the computing abstraction stack to satisfy embedded system constraints. Contemporary resource management approaches typically focus on a fixed objective, while neglecting the need for replanning (i.e., updating the objective function). This trend is particularly observable in existing resource allocation and application mapping approaches that allocate a task to a tile to maximize a fixed objective (e.g., the cores' and network's performance), while minimizing others (e.g., latency and power consumption). However, embedded system goals typically vary over time, and also over abstraction levels, requiring a new approach to orchestrate these varying goals. We motivate the problem by showcasing conflicts resulting from state-of-the-art fixed-objective resource allocation approaches, and highlight the need to incorporate dynamic goal management from the very early stages of design. We then present the concept of a Hierarchical Dynamic Goal Manager (HDGM) that considers the priority, significance, and constraints of each application, while holistically coupling the overlapping and/or contradicting goals of different applications to satisfy embedded system constraints.**

***Index Terms*—Goal Management, Many-cores, Resource Allocation, Dynamic Mapping, Dark Silicon, Lifetime Balancing**

## I. Introduction

Aggressive technology scaling has exacerbated challenges resulting from process variation, failure of Dennard's law, emergence of dark silicon, poor energy proportionality, faults manifesting from thermal issues, aging [1], etc. Thus, hardware platforms for embedded systems are becoming increasingly inefficient and vulnerable to diverse conflicting system constraints such as limited power resources, temperature accumulation, silicon meltdown, faster aging, etc. *We define **Constraint** as a parameter that controls what we are interested in to satisfy for instance by keeping it within particular limits.*

In this context, embedded systems face extreme challenges in ensuring their objective such as QoS, energy efficiency, reliability, etc., over a diverse set of applications, in the face of multiple, possibly conflicting constraints. *We define **Objective** as a cost function that linearly combines different constraints through different weights (e.g., priorities) for achieving a specific result within these constraints. In this context, being **Multi-objective** indicates having an objective function with more than one constraint.* The radar chart in Figure 1 illustrates these challenges. State-of-the-art performance-driven resource allocation approaches have the fixed objective of exclusively focusing on power-performance optimization (blue overlay), while they do not adapt at runtime to consider other critical requirements such as error detection, life-time balancing, and temperature accumulation [2]–[6]. The same applies to reliability, thermally-efficient, and life-time balancing oriented designs (grey overlay), where considerations on QoS and performance surges are not addressed dynamically [7]–[9]. Similarly, throughput driven approaches focus on maximizing the system throughput while sacrificing the latency constraints of individual applications (orange overlay) [10].

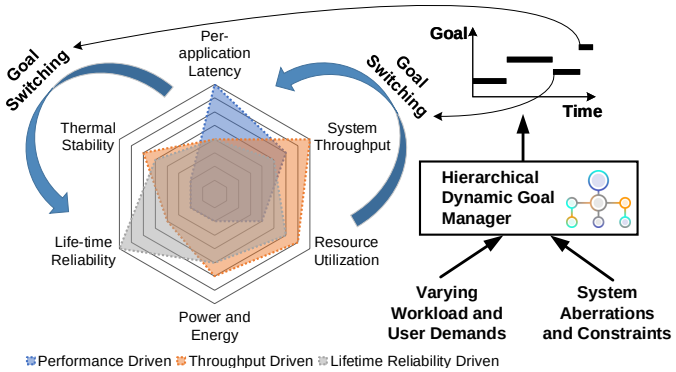



Figure 1. Hierarchical Dynamic Goal Management for Goal Switching

Thus there is a growing need for a systematic scheme to manage system goals through coordinated orchestration. *We define **Goal** as high-level loosely defined plans of the system that can be selected, prioritized (i.e., managed) at runtime due to changes in the environment or the system's own state, and would consequently result in updating the weights in the objective function (i.e., replanning).* Therefore, objectives are more specific and easier to measure than goals. We posit that a Hierarchical Dynamic Goal Manager (HDGM – right side of Figure 1) can effectively manage a pyramid of goals by considering varying goal priority, as well as the significance and requirements of each application to serve application requests. HDGM could improve efficiency in resource utilization and decision making, since the goals of different applications may be fully or partially exclusive, overlapping, or contradicting. For instance, an NVIDIA study reports that most mobile devices (deploying embedded multicore platforms) are typically in standby state 80% of the time, and execute compute-intensive mobile applications only 20% of the time [11]. However, the usage of individual devices varies greatly, based on user activity and preferences. Similarly, a study of server utilization in public clouds reports that the workload at night is half the workload during the day [12]. Intuitively, *performance* is of higher priority during the active mode for mobile processors or during the day for cloud providers due to heavier workloads. With reduced workload during the standby mode or the night and lower requirement on performance, available/idle resources can be utilized to manage other goals such as *life-time balancing* by updating their priorities.

In this article, we motivate the problem by showcasing some of the conflicts arising from state-of-the-art resource allocation approaches that are typically focused on a fixed objective. We then present a high-level architecture for Hierarchical Dynamic Goal Management (HDGM) that can handle the significance, constraints, and requirements of each application, while holistically considering the overlapping and/or contradicting goals of different applications to satisfy system-level goals.

Amir M. Rahmani is with the Department Computer Science, University of California Irvine, CA, 92697 USA & Institute of Computer Technology, TU Wien, 1040 Vienna, Austria, e-mail: amirr1@uci.edu.

Axel Jantsch is with the Institute of Computer Technology, TU Wien, 1040 Vienna, Austria, e-mail: axel.jantsch@tuwien.ac.at.

Nikil Dutt is with the Department Computer Science, University of California Irvine, CA, 92697 USA, e-mail: dutt@uci.edu.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LES.2017.2751522, IEEE Embedded Systems Letters



## II. Background on Dynamic Task Mapping

Dynamic task mapping is split into two phases viz., finding a suitable region to map an application, followed by mapping tasks of the application in the selected region [3]. Each application in the system is represented by a directed task graph $Ap = TG(T, E)$ (an example application is shown in Figure 2). Each vertex $t_i \in T$ represents one task of the application, while the edge $e_{i,j} \in E$ represents a communication between the source task $t_i$, and the destination task $t_j$ [3]. Each task $t_i$ is defined as 3-tuple $tnr_i = \langle id_i, ex_i, pr_i \rangle$, where $id_i$ stands for the task identification, $ex_i$ represents the task execution time, and $pr_i$ denotes the task priority. An application is defined as a set of tasks having inter-dependencies. Therefore, application mapping is a one-to-many function. It should be noted that the discussion made in the following is also relevant to shared-memory based multi-threaded many-core systems.

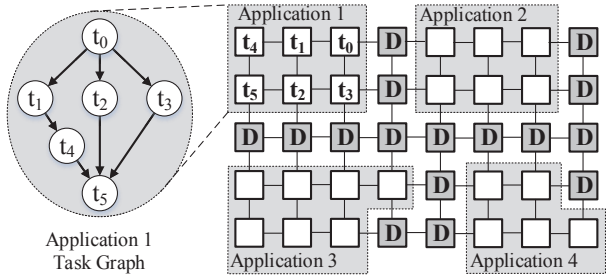



Figure 2. Mesh-Based platform with an application mapped onto it (the highlighted region.) where some cores are dark (D) [1]

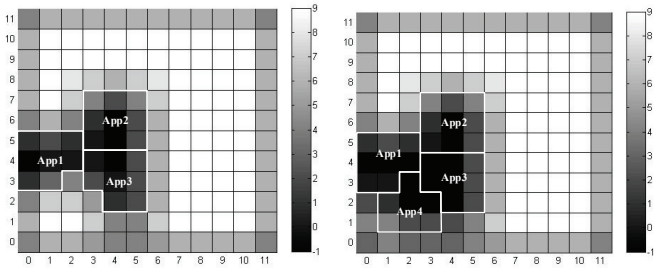
## III. Conflicting Goals in Fixed-Objective Resource Allocation Approaches

To motivate the need for coordinated, dynamic goal management, we now highlight conflicting goals resulting from examples of typical contemporary fixed-objective resource allocators for many-core systems. It should be noted that each of the works discussed in the following is representatives of a rich class of resource allocation techniques. A comprehensive survey of the field can be found for instance in [13]. Moreover, there exists a rich body of literature on resource allocators considering other constraints such as faults, QoS, process variation, etc.

### A. Performance-Driven Resource Allocation

**Problem Definition and Objective:** Given a new application commencing execution at runtime, map it to an optimal region on the system which results in the lowest per-application execution time, i.e., lowest average Manhattan distance, lowest average packet latency, lowest internal and external congestion, and lowest mapped region dispersion. To meet this objective, the selected region needs to have the following attributes: i) *Spatially available* - enough number of free nodes around it to map the application, ii) *Contiguous* - free nodes that are contiguous to minimize internal congestion, and iii) *Near Convex* - free nodes forming a near convex geometrical shape (square shaped) to minimize dispersion and external congestion.

**Solution:** We use MapPro [2] as an exemplar of a contemporary technique from the class of approaches addressing this objective. MapPro's preferred mapping solutions result in lower congestion, dispersion, power consumption and higher performance, which is realized by **mapping dense and contiguously**. As it is assumed that the applications are in the form of data flow graph and there are dependencies among their comprising tasks, contiguous mapping can considerably mitigate congestion, in particular when considerable portion of the applications running on the system are communication intensive. It deploys a proactive region selection strategy which quantitatively represents the propagated impact of spatial availability and dispersion on the network with every new mapped application. Figure 3(a) shows a scenario where three applications are already mapped and running on a 12×12 system, and there is an application with a size of 7 tasks (*App4*) issuing an execution request. Figure 3(b) shows how MapPro maps this application onto the system and updates the probability of the unoccupied cores around *App4* for the future mappings. MapPro enhances congestion and dispersion in the system by up to 28% and 21%, respectively, compared to other state-of-the-art region selection methods in the same class such as [4]. We observe that the main goal of these dynamic resource allocation approaches is fixed on *enhancing per-application execution time (i.e., latency) and cannot adapt to focus on improving the overall system utilization and throughput*.




(a) System state before mapping *App4* (b) System state after mapping *App4*

Figure 3. Mapping dense and contiguously by using MapPro [2]

### B. Throughput- and Power-Constrained Resource Allocation

**Problem Definition:** Another class of resource allocation problems addresses orthogonal issues in managing the higher power density and potential hotspots, that in turn result in reducing the chip's utilizable power budget. With contiguous mapping of applications, all the active cores are tightly packed, resulting in the heat dissipated by every active core affecting its neighbors; consequently, cores can reach their critical temperature even when on a lower power budget. On the other hand if the cores are **spread out**, the heating effect of one active core on the others is minimized, thereby allowing cores to consume more power before reaching their critical temperature. Furthermore, dark silicon constraints will force a variable number of cores to be inactive (dark). This results in the upper bound on power consumption largely varying at runtime. A sensible way to avoid this is to use a variable and realistic upper bound on power consumption, Thermal Saturation Power (TSP), as proposed in [14]. For instance, mapping of the 3 applications contiguously on a NoC-based many-core system with 144 cores, under safe peak operating temperature (80°C), results in a power budget of 66W (Figure 4(a)), while a non-contiguous and spread-out mapping of the same applications (as well tasks) leads to a power budget of 74.6W (Figure 4(b)).

**Objective:** Given an incoming application, find a region at runtime that has minimal effect in terms of temperature on other regions, and align (i.e., map) active cores in a way that results in a higher power budget over some other mapping. Subsequently, the surplus budget gained through mapping can be utilized to power up additional, otherwise dark cores, thereby maximizing overall system utilization and throughput.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LES.2017.2751522, IEEE Embedded Systems Letters



**Solution:** A representative approach by Kanduri *et al.* [10] deploys a closed-loop feedback Budget Manager (the upper left side of Figure 5) interacting with the runtime mapping unit (RMU). The Budget Manager's TSP Calculator computes the TSP using the the current mapping configuration of the system. The RMU considers the spatial distribution of applications and sparsity among tasks of each individual application in the selected region to perform the allocation.

In this approach, *inter-core communication latency and per-application latency are traded off to improve maximum utilizable power budget and thus gain system throughput*. This results in increased average application execution time due to longer communication latency, while the surplus power budget allows more incoming applications to be executed (i.e., more parallelism and less dark silicon). We note that these goals are contradictory to the previous performance-driven allocator, and as the objective is fixed, the system cannot adapt to possible changes.

### C. Lifetime-Reliability-Driven Resource Allocation

**Problem Definition:** A third class of resource allocators focuses on lifetime reliability: uneven resource allocation stresses components of a chip in a non-uniform fashion, resulting in imbalanced aging and reliability of the chip. Recent studies have shown that a $10-15°$C difference in operating temperature may result in a $2\times$ difference in the overall lifespan of a device [15].

**Objective:** These resource allocation methods aim to optimize system performance, while maximizing the overall system lifetime by **choosing less stressed resources** and providing long term recovery periods for highly stressed cores.

**Solution:** A representative approach [8], [9] achieves lifetime-reliability balancing by deploying a centralized controller composed of a long-term runtime reliability analysis unit and a short-term runtime mapping unit (the lower left side of Figure 5). The *Reliability Analysis Unit* is the long-term controller responsible for monitoring the aging status of the various cores. The unit analyzes the current reliability value of each core w.r.t. the target aging reference to compute a specific reliability metric describing the aging trend. The *Runtime Mapping Unit* as the short-term controller dispatches the applications onto the system considering the provided reliability information.

Hagyhbayan et al. [8], [9] showed that assuming the goal is to provide at least a 30% reliability for each single core at the end of the target lifetime of a $12\times12$ many-core system, performance-centric and reliability-agnostic dynamic mapping strategies such as MapPro miss the requirement before 6 years (Figure 6(a)). However, with reliability-aware dynamic resource allocation it is possible to guarantee the specified reliability constraint (as shown in Figure 6(b) where only after 12 years the first core has a reliability lower than 30%).

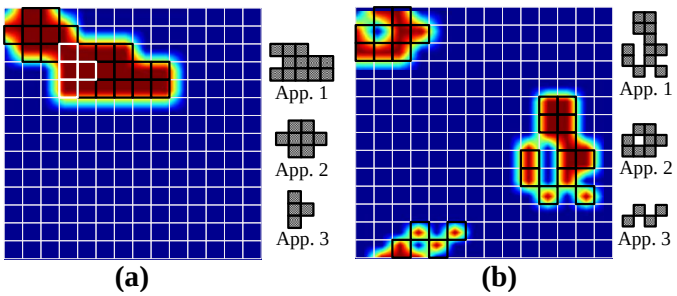



Figure 4. Effect of mapping on temperature accumulation [10]

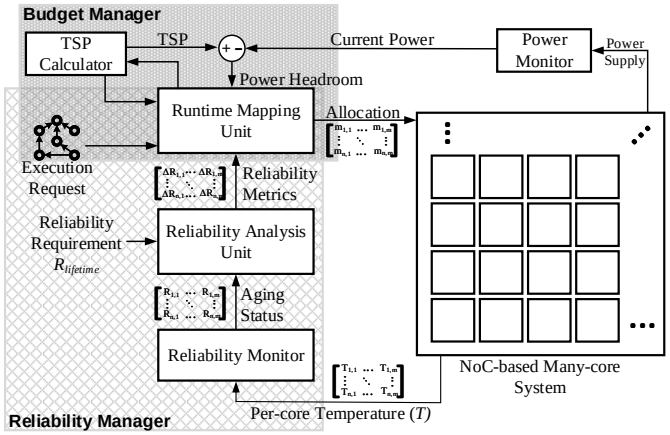
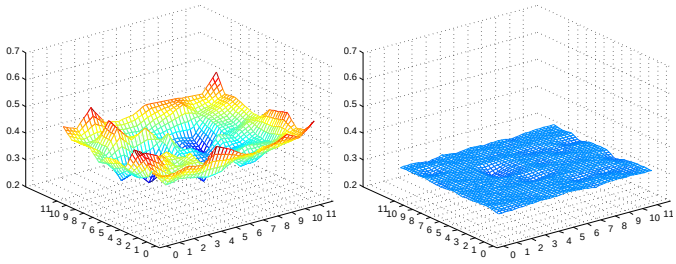



Figure 5. High-level architecture implementing throughput-driven vs. reliability-driven mapping strategy



(a) Using MapPro (lifetime=5.52 years) (b) Using reliability aware mapping (life time=12 years)

Figure 6. The effect of different runtime mapping approached on overall system lifetime (until the first core's reliability reaches 30%) [9]

Here, the chip's lifetime was considerably improved while imposing a 7% average performance slowdown (with considerably higher worst case penalty).

Once again, this approach presents a fixed objective function as well as partially overlapping and contradicting goals on resource allocation viz., *to enhance power-performance characteristics vs. to enhance balanced allocation*.

To summarize these three case studies: (a) performance driven allocation leads to dense mappings that are not optimal for temperature dissipation, utilization and a long lifetime; (b) Power-constraint driven allocation leads to lower performance and also ignores life-time concerns; (c) life-time maximizing allocation leads to non-optimal performance and ignores temperature dissipation concerns. While sometimes these different goals can be reconciled, very often one has to be compromised for the benefit of another goal through prioritization of different constraints w.r.t. changes in the user preferences, environment, or system's own state. The challenge is how to decide when to compromise which goal.

## IV. Hierarchical Dynamic Goal Management

We argue that the pursuit of isolated goals is not effective for handling the complexity of modern many-cores. There is a critical need to coordinate scores of runtime parameters for sustained efficiency at the hierarchy of these overlapping and contradicting goals. We believe that a Hierarchical Dynamic Goal Management (HDGM) approach will enable improved abstraction to effectively handle complexity, manage on-chip resources, and establish synergistic actuations to meet various (conflicting and complementary) QoS guarantees. HDGM can effectively fuse sensory data across layers and deploy a hierarchy of goals and commands from the application layer, to orchestrate effective system execution, while meeting the

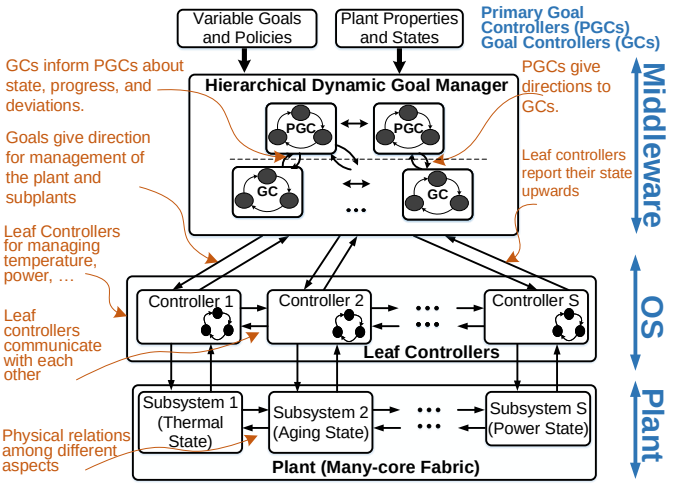



Figure 7. High-Level architecture for hierarchical dynamic goal management.

dynamic changes in the priorities of goals and sub-goals at runtime. It needs to also to adapt and evolve over time to optimize itself. Such a framework can generalize the existing management policies and simplify them by *separating the many-core platform from the goal management system*.

A HDGM needs to navigate through a space with: i) a set of *goals/subgoals*, including a predefined subset of goals, that can be dynamically generated and eliminated at runtime (e.g., the case when an incoming realtime application requests execution while there is not any other realtime application running on the system), ii) a set of *priorities* for changing the significance of goals at runtime (e.g., life-time balancing gets a higher priority when a system is in the active standby mode), iii) a notion of *time* as an input to the goal function to update priority of goals over time, to be used for goal tracking epochs with different timescales (e.g., overboosting can be a short-term goal while lifetime balancing is a long-term one), etc., iv) an *inspection function* to audit how the current resource allocation is effective in meeting the goals, and v) a *goal function* capable of categorization and holistic orchestration of application-level and system-level goals over time considering the inputs from the inspection function, the system, and the application layer.

An example high-level architecture of a HDGM is shown in Figure 7. HDGM manages a pyramid of goals in a hierarchical fashion. One possible implementation is to consider a two-tier hierarchy of controllers, denoted as Primary Goal Controllers (PGCs) and Goal Controllers (GCs), respectively, as shown in Figure 7. In this architecture, HDGM receives primary goals from the application layer while having access to properties and initial states of the manycore fabric. Primary goals are highest up in the goal hierarchy. They include application specific goals but also basic, generic goals concerning survival and harm avoidance. Goal controllers manage application relevant, partial goals such as frame rate, survival time, audio quality, etc. An important challenge in the realm of goals is to arbitrate between conflicting and competing goals. In this architecture, PGCs give directions to GCs, and GCs inform PGCs about state, progress, and deviations. Leaf Controllers at the bottom layer interact with the HDGM and the controlled system. They are responsible for managing low-level parameters of the many-core fabric such as temperature, power, etc. A leaf controller can be for example a power budget or reliability manager (e.g., the controllers shown in Figure 5). In the same fashion, leaf controllers report their state upwards to the HDGM, and based on the assessment, directions to the leaf controllers are given by the HDGM.

While this architecture can be a good initial exemplar for a HDGM, it does not consider the required theoretical models, and implementation and verification complexity/challenges of realizing such a manager, however it can lay the ground for future work to address many of the open challenges we describe next.

## V. Conclusions and Challenges

In this article, we illustrated some of the conflicts resulting from state-of-the-art many-core resource allocation approaches when the focus is a fixed objective. We then motivated the need for dynamic hierarchical goal managers to holistically coordinate the overlapping and/or contradicting goals of different applications as well as system-driven goals which may vary over time. Although we argue that explicit goal management (GM) is becoming a necessity for emerging many-core embedded systems, significant challenges remain open to realize systematic, efficient, interoperable, and robust goal management approaches. We also believe that goal management has to be incorporated into the system at the very early design stage to appropriately handle specification, design and verification. Although there are many open problems, we pose the following challenges to initiate research in this area:

- How to formally model and formulate the GM?
- How to verify it w.r.t. convergence, efficiency, robustness, QoS guarantees, etc.?
- How to design efficient cross-layer architectures for GM?
- How to handle the hierarchy of goals?
- How to make GM lightweight and interoperable?

## Acknowledgment

We thank M. Hashem Haghbayan, Anil Kanduri, Antonio Miele, and Pasi Liljeberg for the conception and development of the approaches presented in Section III. We also acknowledge financial support by the Marie Curie Actions of the European Union's H2020 Programme.

## References

[1] A. Rahmani *et al.*, *The Dark Side of Silicon*, 1st ed. Springer, Switzerland, 2016.
[2] M.-H. Haghbayan *et al.*, "MapPro: Proactive Runtime Mapping for Dynamic Workloads by Quantifying Ripple Effect of Applications on Networks-on-Chip," in *Proc. of NOCS*, 2015.
[3] M. Fattah *et al.*, "Mixed-Criticality Run-Time Task Mapping for NoC-Based Many-Core Systems," in *Proc. of PDP*, 2014.
[4] ——, "Smart hill climbing for agile dynamic mapping in many-core systems," in *Proc. of DAC*, 2013.
[5] A.-M. Rahmani *et al.*, "Dynamic Power Management for Many-Core Platforms in the Dark Silicon Era: A Multi-Objective Control Approach," in *Proc. of ISLPED*, 2015.
[6] M.-H. Haghbayan *et al.*, "Dark Silicon Aware Power Management for Manycore Systems under Dynamic Workloads," in *Proc. of ICCD*, 2014.
[7] C. Bolchini *et al.*, "Lifetime-aware load distribution policies in multi-core systems: An in-depth analysis," in *Proc. of DATE*, 2016.
[8] M. H. Haghbayan *et al.*, "Can Dark Silicon Be Exploited to Prolong System Lifetime?" *IEEE Design and Test*, 2017.
[9] ——, "A lifetime-aware runtime mapping approach for many-core systems in the dark silicon era," in *Proc. of DATE*, 2016.
[10] A. Kanduri *et al.*, "Dark Silicon Aware Runtime Mapping for Many-core Systems: A Patterning Approach," in *Proc. of ICCD*, 2015.
[11] NVIDIA, "Variable SMP - A Multi-Core CPU Architecture for Low Power and High Performance," *White paper*, 2011.
[12] H. Liu, "A Measurement Study of Server Utilization in Public Clouds," in *Proc. of DASC*, 2011.
[13] A. K. Singh *et al.*, "Mapping on multi/many-core systems: Survey of current and emerging trends," in *Proc. of DAC*, 2013.
[14] S. Pagani *et al.*, "TSP: thermal safe power: efficient power budgeting for many-core systems in dark silicon," in *Proc. of CODES+ISSS*, 2014.
[15] Y. Xiang *et al.*, "System-Level Reliability Modeling for MPSoCs," in *Proc. of CODES+ISSS*, 2010.