The Role of Self-Awareness and Hierarchical Agents in Resource Management for Many-Core Systems

Maximilian Götzinger[1], Amir M. Rahmani[1], Martin Pongratz[2], Pasi Liljeberg[1], Axel Jantsch[2], and Hannu Tenhunen[1]
[1]Department of Information Technology, University of Turku, Finland
[2]Institute of Computer Technology, TU Wien, Austria
{maxgot, amirah, pakrli, hatenhu}@utu.fi, {martin.pongratz, axel.jantsch}@tuwien.ac.at

***Abstract*—The future of Moore's Law is in jeopardy. The number of cores of many-core systems is steadily increasing for every technology node generation. Voltage scaling does not keep pace with the unabated decrease of transistor size. Higher leakage power and manufacturing variabilities are the consequences and lead to extremely critical power as well as thermal issues. These phenomena can downgrade the performance or endanger system's functionality as well as its reliability if they are not properly addressed. In near future, up to 90% of a many-core chip's area may have to remain inactive; this non-active area is termed Dark Silicon. These issues make the problem of resource management challenging. Future management systems need to be intelligent, anticipatory, and self-adaptive. They are supposed to integrate management of different aspects such as thermal, power, energy, performance, quality of service, process variability, occurrence of faults and aging effects, all in one. In this paper, we study the contributions in the literature focusing on techniques for dynamic resource management in multi- and many-core systems. We put emphasis on advanced approaches that exhibit learning, self-awareness, hierarchical monitoring and management. We categorize the existing approaches from a new perspective and argue that a self-aware hierarchical agent-based model is a proper methodology to monitor and management many-core systems, in particular when they need to deal with different competing goals. In addition, we evaluate the main objectives and trends in resource management of many-core systems in order to pave the way for designing future computer systems ranging from high-performance computers to embedded processors used in the era of Internet-of-Things.**

Keywords*—*Self-Awareness; Hierarchical Agent-Based System; Many-Core; Dark Silicon; Power-, Thermal-, and Reliability Management;

I. Introduction

As transistors shrink with each technology generation and voltage scaling has almost leveled off, the power- and thermal density increases [1]. On the other hand, the number of cores, which can be integrated in a single chip, is steadily increasing [2] and leads to the era of many-core systems. This consequently makes resource management issues more critical [3], and can affect the functionality and efficiency of many-core systems in short-term as well as in long-term conditions. An example for the former could be overheating that damages the device (in particular due to a positive feedback between temperature and leakage currents). In long-time, thermal issues worsen the susceptibility of the devices to aging and wear-out phenomena (such as time dependent dielectric breakdown, thermal cycling, or electromigration) [4], [5]. As a result, the reliability and the lifetime of the system can drastically decrease [6], [7]. In addition, smaller feature size implies relatively larger manufacturing process variability, thus leading to more vulnerability to environmental changes [8], [9]. These hardware variations are characterized through diversity in process-, voltage-, and thermal issues. Consequently, manufacturing deviations or uneven workload lead to resource under-utilization and/or hotspots which can cause performance degredation as well as significant unbalanced chip life-time or wear-outs [8], [10].

These effects and environmental variations can downgrade the performance or endanger the system functionality if they are not properly addressed. On top of the aforementioned issues, there exists a utilization wall [11] that is becoming more severe with the scaling of transistors, thus limiting the utilization of transistors on a chip. High power density and thermal issues are rapidly bringing computing systems to a crossroad where a chip can accommodate a large number of cores, however a significant fraction of them should be left unpowered at any time [12]. In other words, it will not be possible to run all cores and un-cores at the same time at full throttle; a substantial amount will have to be turned off. These large unpowered areas have to stay dark, and are called dark silicon [13], [14]. There are various studies concerned with dark silicon, estimating that 52% [15] or even up to 90% [11] of a high end-chip may have to remain dark within 5 years.

There have been extensive efforts to propose smart resource management techniques for many-core systems [9], [16]–[18], which having different approaches or ideas. However, there is still not a fully intelligent and self-adaptive solution [9] in the literature which is able to consider its own state, environmental situations, and a pool of competing goals. In this paper, we investigate and study the existing contributions aiming at tackling resource management issues for many-core systems in an adaptive and intelligent fashion. We then argue that further research on self-awareness with learning ability and hierarchical agent-based monitoring and management can considerably enhance the efficiency of the existing resource management techniques and can be considered as a promising avenue of research. The main contributions of this study are as follows:

- Investigating the existing literature from the perspective of self-adaptivity.
- Developing a comprehensive perspective on the area.
- Expediting progress on self-aware and hierarchical resource management of many-core systems.
- Supporting future research in the context of addressing issues in the dark-silicon era.

The rest of the paper is organized as follows. Resource management issues and possible solutions to circumvent them is described in Section II. Whereas, Section III shows the significance of self-awareness in complex resource manage-

ment scenarios. In Section IV, we discuss the advantages of a hierarchical agent-based model in meeting different system requirements. Section V investigates the resource management issues and presents existing solutions in the literature to cope with them. In section VI, we provide some guidelines and suggestions how to extend existing approaches and algorithms. Finally, Section VII concludes the paper.

II. Resource Management

There are many elements in a complex many-core system which can be considered as a resource. Some key examples are: a time slot from a core or a processing element (PE), the power budget of the chip, communication bandwidth, or even the estimated age of a component. This has resulted in emergence of different resource management fields for many-core system such as power and thermal management, dynamic mapping and core allocation, life-time balancing and reliability management, networks-on-chip (NoC) communication management, etc.

There exist several techniques and knobs to tune and adjust different system properties in order to enhance resource utilization, reliability, efficiency, and performance. Dynamic resource management can balance the workload to reduce power consumption, evenly distribute temperature, minimize thermal hotspots, enhance fault tolerance, or even balance the lifetime of the system. The actuation knobs widely used by these management techniques are: clock gating, power gating, dynamic voltage and frequency scaling (DVFS), dynamic application scheduling and mapping, task migration, network-on-chip adaptation, etc. In this section, we briefly present some of the popular knobs in this context.

Dynamic Voltage and Frequency Scaling: Adaptive up-scaling and downscaling of voltage and/or frequency (VF) of on-chip components is one of the most effective and widely used way to counteract thermal, power, and aging issues [19], [20]. It represents an approach that can dynamically modify a component's power consumption, thus avoid overheating [20].

Voltage scaling is a highly effective method to reduce static- as well as dynamic power. Equation 1 shows the proportional relation of the static power P_{static} and the supply voltage V_{DD}. Whereas, as shown in Equation 2, dynamic power $P_{dynamic}$ quadratically depends on the supply voltage for given load capacitance C_{load} and operation frequency f.

$$P_{static} = I_{leakage} * V_{DD} \tag{1}$$

$$P_{dynamic} = C_{load} * V_{DD}^2 * f \tag{2}$$

Frequency scaling concerns with the regulation of a component's operating frequency. As shown in Equation 2, dynamic power $P_{dynamic}$ is proportionally related with the operating frequency. The lower the frequency, the lower the dynamic power is. Even though according to Equation 1, static power P_{static} does not depend on the frequency f directly, in fact it indirectly does. A core needs a certain minimum supply voltage for a given frequency [21], [22]. For instance, Intel Single-Chip Cloud computer (SCC) [21], which offers 16 frequency levels to regulate the speed of a frequency Island, requires 0.8V to supply a core running at 533MHz, while for a core running at 800MHz a minimum of 1.1V is required. Therefore, choosing a lower frequency can lead to a lower requirement for supply voltage and, in further consequence, to a reduction of static power as well.

In [19], it has been shown that by lowering the VF level, energy savings of 4% to 24% can be achieved with the performance penalty of approximately 2.5%. Similarly, in [20], it is reported that 20% - 50% less power consumption and a significant reduction of temperature can be obtained by reducing the supply voltage by about 10%.

Clock Gating: Cores in a many-core system need to have the support to be halted (i.e., freezed) when they are idle for short period of time [23]. One technique is to clock-gate them while all the rest of the chip can still work at different variable speeds, controlled by clock frequency or supply voltage [13]. Clock gating is widely used to reduce dynamic power in particular at the Register Transfer Level [24], [25]. It is an efficient approach to minimize heat, radiation and power consumption while preserving the state of the core/component [26], [27]. According to [28], clock tree in computer systems consumes 15-45% of total systems power. As can be observed from Equation 2, clock gating blocks dynamic power consumption, however it does not mitigate the leakage power and its associated issues.

Power Gating: With the scaling of transistor size, each new technology generation results in a considerably higher leakage power. Accordingly to [29] and [30], leakage power can increase even up to 30 times from one generation to the next one. Therefore, it is of utmost importance to have the possibility to turn off idle cores to avoid unnecessary static power consumption [30]. Power gating is a widely accepted and effective solution to overcome this problem, however it has also disadvantages. Gating the power supply leads to a charged-up virtual ground rail which is close to V_{DD}. When the power supply is connected again, the virtual ground rail capacitance has to discharge to ground which leads to a wake-up time latency $T_{wake-up}$ [31]. Therefore, compared with the clock gating, power gating better reduces temperature as well as power due to saving both static and dynamic power, but has the drawback of a high boot up latency. Additionally, power gating can lead to noise on the power lines when circuit portions are switched on or off [32]. However, as presented in [32], there are different techniques to mitigate this phenomenon.

Dynamic Application Mapping: Application mapping is the process where a core is chosen for a task in order to maximize the system performance while minimizing latency and power consumption. In the case where workloads have unpredictable nature and applications enter and leave the systems at runtime, mapping has to be performed dynamically rather than at design time [33]. The allocation of tasks and the scheduling of applications on a many-core system can also impact power consumption and system's peak temperature [34], [35]. Distributing the workload stress across the system's cores can lead to a higher system's reliability and lifetime [36]. However, dynamic task mapping (DTM) may lead to missing deadlines when timing constraints are not carefully considered [35]. Therefore, it can be observed that DTM can have a direct or indirect impact on different system characteristics and resources such as power, temperature, performance, life-time, fault tolerance, etc.

Task Migration: Task migration has been recently introduced in the on-chip context as an effective way to balance different system characteristics at runtime [37]. Several task migration techniques exist in the literature [38], [39]. Even though task migration has shown to be effective, it has its own

associated drawbacks such as the delay to detach the task from the source core, move the task to the new location, and attach it to the destination core [37]. This overhead might be negligible in bus-based multi-core systems, but it is considerable in NoC-based many-core systems. In terms of performance, the distance between the core on that a given task is processed and the location of the task's resources should be as small as possible. Otherwise delays and the runtime will get longer. On the other hand, the locations of the tasks also influence the thermal profile of the chip. Many tasks packed in the same region can lead to a high thermal stressing or even hotspots [40], [41]. A well thermal balanced situation can be brought off with proper task migration [42], [43].

Reconfigurable On-Chip Communication: Network-on-Chip (NoC) is the widely accepted communication medium for many-core systems; mostly due to its scalability and flexibility [44]. There are many NoC-based multi- and many-core systems in the market such as Intel 80-core TeraFLOPS processor [45], Tilera 64-core TILE64 processor [46], etc. Reconfiguration of a NoC can be also considered as a way of actuation which can impact different system-level characteristics such as improving performance by avoiding congestion [47], mitigating hotspots [48], reducing power consumption [49], improving fault tolerance and reliability [50], etc.

Resource Management in Complex Many-core Systems: There is a high demand for efficient and intelligent thermal management techniques to circumvent different issues in complex many-core systems and maximize the resource utilization. Although, all these techniques and their combinations have been developed to great sophistication and are used in industrial practice today, the resource management of many-core systems still lacks efficiency and intelligence. The management in many-core systems has to have the ability to decide on its own about which parts to be made dark, dim, or work on full throttle. A self-aware system is needed that can learn about the variabilities of the chip and workload; a system that can monitor and set these things in every granularity.

III. Self-Awareness and Its Significance

To cope with the emerging issues in the dark silicon era, a smart and comprehensive resource management strategy for many-core systems is needed. A computer system, especially a many-core system has to provide an intelligent load balancing as well as a sophisticated voltage-frequency controlling. The goal is to control the system in charge of managing phenomena such as aging and overheating [8]. However, such a system will face several questions such as when will computation power be needed? How much will be needed? Where it will be needed? What is the current goal of the system? In the case of partially or fully conflicting goals (e.g., Lifetime balancing vs. Performance), what will be the strategy? Does the evaluation of the parameter in question need to consider history? In what kind of situation the system is currently being used? How should the system adapt to changes in situation?

Therefore, the system itself needs to be smart; it has to know how it can meet its requirements under given constraints. There are many properties that are linked to the concept of "smartness of a computer system", for example: self-adaptivity, self-awareness, and autonomy. Smart resource management systems have to be adaptive and able to change their behavior to achieve their goals in an efficient way, although environmental conditions or demands are subject to change.

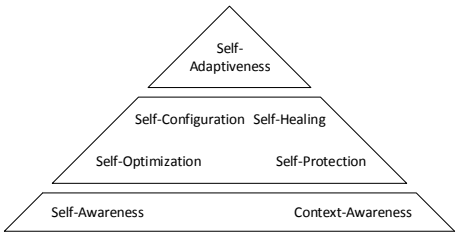

Fig. 1. The pyramid of self-adaptiveness. A system can only be self-adaptive if it is self-aware [53].

Self-awareness can enable autonomy of making decisions and taking actions [51]. As Figure 1 shows, a system has to be self-aware to be self-adaptive [52]. The computer system has to "feel" itself as well as the environment to react and efficiently adapt its behavior. For going not beyond the scope of this paper, we refer to [53] for a detailed description of the other self*-properties.

Self-awareness can be seen as an ability of an object to monitor its own state and behavior to adapt its activities to changes in its internal state as well as the external environment [52]. Originally, this behavior stems from human's psychology where he/she needs to cope with different situations. While it originates from biology and psychology, it recently has become widely studied and utilized in computer science. For example in [9], it is shown that a self-aware system can enable an intelligent load balancing as well as power reduction and hotspot mitigation.

In analogy to living subjects, the system-on-chip has to process the data of its sensors to figure out what is going on and how it should behave in the given situation [54]. Systems that only observe their own states have a restricted view on their own behaviour and performance, and therefore, they cannot be labeled as true self-aware systems. The term self-awareness comprises three main competencies: self-monitoring, attention, and situation awareness. A self-aware system has to track the environmental conditions, its own state, and how it behaves in a dynamically changing environment. In other words, such a system has to know about itself and its environment [54].

To realize self-awareness, the underlying many-core system needs to have access to a rich set of sensors and actuators, so that necessary sensory data is available for it to enable a continuous tracking and interpreting of its own as well as the environmental conditions in a closed-loop [8]. Additionally, an adaptive and reflective middleware is needed which is capable of managing the system resources on the basis of abundant sensory data (e.g., power sensors, thermal meters, performance counters, network traffic meters, aging sensors, etc.). In other words, a flexible hardware-software stack is required which constitutes the interface between the application layer and the operating system (OS) layer. This middleware traces parameters from the application, the own resources, and the environment to enable the necessary actions accordingly [9]. More precisely, while application requirements are generally the main goal that should be reached, parameters associated with the underlying hardware and the environment constitute the complementary constraints. All these parameters and requirements serve as input for a cost function based on which it is decided whether an action is needed to be taken or not [9].

A. Classes of Self-Awareness

Self-awareness can be divided into different categories of awareness. They are briefly presented in the following. More details on this concept can be found in [16].

Stimulus-aware: This is the foundation stone of the other forms of awareness. In principle, it means a self-aware system should have the property of recognizing an event as an *event* to react to it.
Interaction-aware: A system is called interaction-aware if it is able to understand reactions to its own acting in a way that it can provoke other participants to react as well.
Time-aware: A system is called to be time-aware when it can take into account not only actual ongoing parameters, but also the ones from the past. This property can enable predictions for the future and appropriate learning algorithms to be realized.
Goal-aware: Goal-aware constitutes the most abstracted knowledge. The system has knowledge about its goals as well as its objectives and is supposed to have a practicable strategy to reach them.

All these awareness categories are essential in order to actualize a system which is capable of efficiently and intelligently managing its own resources and meeting the given constraints and goals.

B. Properties of Self-Awareness and Self-Aware Resource Management

The aim of a self-aware and thermal-aware resource management approach is to seek for a fair compromise of performance and energy consumption with explicit consideration of thermal stress. The system performance has to be set to a level that the given goals are fulfilled, while the hardware is used in a energy-efficient way and its functionality is not in hazard [55]. However, there can be more goals where some might be partially conflicting [55]. In addition, different goals can compete with each other in demanding resources that might not be always possible or beneficial for the general functionality of the system to provide. Therefore, the system has to carefully navigate the goal space and explore: what it has to do, how it has to be done, and until when it has to be continued.

In [52], [56], [57], a set of properties of such a self-aware system are defined which are briefly introduced here:

Semantic interpretation: It describes the abstraction of information from a given set of data. It is related to stimulus-awareness discussed before.
Desirability scale: The desirability scale serves to determine whether an event or situation is desirable or not. There can be more than two states on a scale; for example: *excellent*, *very good*, *good*, *fair*, etc.
Semantic attribution: The algorithm that has the property of semantic attribution, maps events and situations on the desirability scale.
History of a property: A history value of a property corresponds to its evolution. To be aware of a property means to be aware of its history.
Goals: A goal constitutes the target of calculations with its requirements and can consist of a set of different sub-goals. These goals can also compete with each other.
The purpose: The purpose of a system is about achieving all given goals as best it can.
Expectation on environment: The system expects from its environment to be bounded to some constraints so that the system can properly operate.
Expectation on subject: The system expects from itself to follow certain constraints and to provide certain requirements so that it can properly operate.
Inspection engine: The inspection engine integrates all important observations into one consistent inertial system.

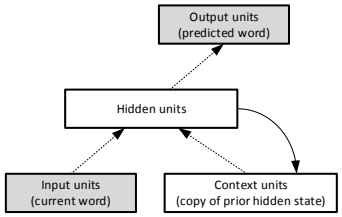

Fig. 2. A simple computer model built to recognize the contexts of words in given stream of sentences [58].

C. Learning Ability

Elman [58] presents a computer model (see Figure 2) built to explore the context of words in a given stream of sentences. The model consist of an input unit, an output unit, and a processing layer (called hidden unit). First, the system is supposed to recognize where a word starts and ends in a given stream received from the input unit. The output is separated into words. Secondly, the computer system should understand the context of these words. Therefore, Elman added a learning unit (called context unit) which is fed with information from the processing layer. It understands more and more contexts with each information it gets. In other words, the context unit supports the processing layer with its decisions.

This simple concept can be also exploited in different domains such as on-chip resource management. Due to manufacturing process variations, not every many-core chip from one series is identical. Additionally, not every core of such a chip will be exactly the same. Therefore, the chip itself needs to learn about the circumstances and environmental conditions through its smart resource management framework. It has to learn things such as What temperatures are fine for different chip areas? How much load is manageable for different components without damaging or over-stressing them in long-term?

In the same way Elman did in his work, a self-aware many-core system can learn about its own abilities, weaknesses, requirements as well as goals given by the application. The learning ability is an essential for self-awareness and can result in significant improvements in system efficiency [59].

IV. Hierarchical Agent-Based Architecture for Monitoring and Management

SoC architectures are often rigid and lack self-adaptation or exhibit self-adaptivity in simplified ways. The self-adaptive capabilities are often limited in terms of efficiency and comprehensiveness where complex trade-offs are typically not considered [16]. As components need be abstracted to acquire knowledge about their semantics and desirability, a model-based design is required in this context. With such an architecture, components can be represented by a function, a timing, and communication specifications [9].

A fine- and coarse-grained knowledge can be obtained when a hierarchical online framework is utilized in an architecture [60], [61]. With such a detailed representation of knowledge, a self-adaptive system can more efficiently operate and meet its goals as well as expectations [16]. Separating different monitoring layers is essential in this context as many-core systems are becoming extremely complex where all monitoring services have their own requirements and constraints [9]. In

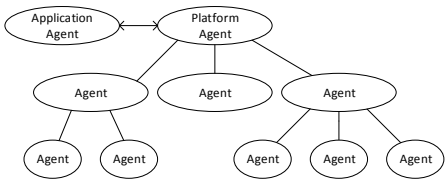

Fig. 3. The figure shows a possible structure of an agent-based model [9].

other words, if different observations are separated to different layers, the information with all its semantic content can be abstracted properly [9]. Guang *et al.* [62] present a hierarchical agent-based model that can be considered as an appropriate architecture to develop and realize the discussed properties of a self-aware and self-adaptable system. A hierarchical agent-based architecture consists of agents which are modules that are hierarchically structured and process data on different levels of abstraction [9]. Therefore, this architecture can provide services needed for a reliable monitoring as well as reconfiguration in both coarse- and fine-grained granularity. Through the different scopes and priorities of several system's agents, the system can be fully or partially reconfigured [17], making this model a promising choice to implement self-adaptation [9]. A disadvantage of this system could be the transmission time between two low-level agents, which are placed on totally different branches and have to communicate over multiple hierarchical levels.

As shown in Figure 3, there are two agents on the top level of this model: an application agent and a platform agent. While the application agent sends runtime application requirements to the platform agent, the latter is responsible for: task mapping, process scheduling, run-time resource management, and fault tolerance of the entire system. The application agent is not aware of the hardware conditions. The whole platform appears like a black box to it, where it can input requirements as well as goals and get results back. Whereas, the platform agent is aware of the entire hardware in a high level of abstraction [9]. The platform agent sends commands to agents below it and these agents report about their knowledge as well as their insights to the platform agent and execute commands that they get. There can be more than two layers in the hierarchy where each layer is connected to different monitoring and reconfiguration functions on different granularities. Such a system (see Figure 3) is freely expandable and does not need to be necessarily symmetrically structured [9].

On the bottom of the model, cell agents are placed, which monitor and reconfigure smaller units associated with them. All agents that reside between the platform agent and the cell agents are called cluster agents. Every cluster agent is connected to at least one cell agent or cluster agent [9]. All cluster- and cell agents are distributed over the whole chip and have their own tasks. Each gets its instructions from the agent above and reports their results to the same direction. While the platform agent has a global influence on the entire chip, with a wide range of monitoring services, a cell agent is only involved with the regional workload, voltage, frequency, etc [9]. A cluster agent's sphere of influence depends on the location of the cluster agent as well as on its connections.

V. Self-Aware and Hierarchical Resource Management and The Overlooked Challenges

In previous sections we discussed about problems and avenues that come up with new technologies. In this section we present connections between them to show that it is possible to counteract thermal, power, and resource management issues with a learning self-aware hierarchical agent-based model.

A. Power and Thermal Management

Adaptive DVFS has been proven to be an extremely effective way to meet power- as well as thermal management challenges [19], [20], [63], [64]. A component's power consumption can be modulated to avoid overheating or for just saving power [20]. Many-core systems are mostly planned to be homogeneous but in reality they are heterogeneous. This heterogeneity comes from manufacturing variabilities and intrinsic as well as extrinsic defects [65]. Runtime schedulers as well as global power managers that ignore the heterogeneous design will most probably barter for performance losses together with a high power wastage [66], [67]. The various cores will vary in frequency, leakage, hardware functionality, and aging effects [64].

MaxBIPS, for example, is an algorithm that exhaustively tries to find the best VF level combinations to drive the hardware at a performance as high as possible under given constraints [63]. Their dynamic management policies achieve better results than static power managements or chip-wide DVFS [63]. However, MaxBIPS is not scalable and, therefore, it is not suitable for many-core systems [59].

Winter *et al.* show in their work [64] that global power management based on linear optimizations are not scalable and, therefore, inadequate for many-core systems. They analyze a global power management for complex many-core architectures and propose a scheduling and power management algorithm (Steepest Drop) for a many-core system with up to 256 cores. They show that their approach performs similar as prior algorithms like LinOpt [68], but it has 75$\times$ less runtime overhead on a 256-core system.

In [59], Chen and Marculescu state that the above mentioned algorithm [64] from Winter is not suitable for many-core systems as it does not take budged overshoot into account. Therefore, they introduce an on-line distributed reinforcement learning algorithm (OD-RL) which is also based on DVFS. The new approach shows that it suppresses the budged overshoot better than the prior ones. The OD-LR algorithm saves 98% budged overshoot due to its ability for adaptation and learning.

Wang *et al.* show in their work [69] that there is a need for a globally and hierarchically coordinated power management control. They state that other works [70]–[72], which do not consider the global application behaviour and do not follow a hierarchical structure, are mostly inefficient in terms of performance and power consumption. Their centralized controller constitutes a bottleneck and slows down the system when the number of cores is increasing. Also, works like [73] can be regarded as sub-optimal solutions because of their lack of global information [69]. Therefore, Wang et al. [69] propose a Hierarchical MultiAgent based Power (HiMAP) allocation scheme which uses clock gating and frequency scaling in a hierarchical structured way and reduces application's execution time up to 38% to 45% compared with PGCapping [71], PEPON [72], and DPPC [70] approaches.

Based on the above discussed works, it can be concluded that a power and thermal management system performs better when: i) it is self-aware, ii) it is able to learn, iii) its architecture is hierarchical, and iv) its structure is agent-based.

B. Application Mapping and Core Allocation Management

From the hardware viewpoint, the hierarchy of the agent-based model is only virtual. Each agent is mapped on one of the cores where it can carry out its tasks in parallel and communicate with each other over the given communication infrastructure. In other words, the various cores are allocated for the different tasks and respectively the different agents are allocated [23], [36].

Hanumaiah *et al.* [74] propose that the relation of temperature and performance is not as trivial as the connection between power and performance. While power and performance are connected over a simple algebraic equation, hundreds of equations will be needed to solve the other connection. This is aggravated by the fact that many-core systems are distinctly more complex than a single core processor. Optimizations made for single core are not necessarily convenient for many-cores [75].

There exists a lot of works that consider static thermal-aware mapping strategies such as [76], [77]. These works consider temperature and communication load, and use heuristics to find a static mapping but they do not take throughput requirements into account [35]. Whereas Sun *et. al.* consider throughput and task deadlines in their work [78]. However, in all these static mapping strategies, communication between different tasks is not considered [35]. Heuristic-based approaches are not able to achieve optimal results and the performance of such systems may degrade massively when the number of controllable resources is increasing [79].

A dynamic mapping strategy is needed to take all circumstances into account [35]. Ge and Qiu [34] propose a runtime multi-agent distributed task migration approach for choosing the best task allocation policy. Results show that their solution could save up to 18% of overall power consumption compared to random task mapping.

C. Reliability Management

Reliability management has different sub-classes such as fault tolerance and testing. The former is important to keep the system running when some faults occur; temporal as well as permanent. The latter is necessary to recognize faults. As we are focusing on dynamic resource management, we just consider online testing techniques and approaches.

Fault Tolerance: A many-core system is self-aware if all the parallelled running entities are self-aware [9]. This characteristic lead consecutively to advantages in terms of safety and fault tolerance. Permanent faults are usually handled by using redundancy techniques [80]. Through these techniques the system is reconfigured to keep on working. If one cell agent or cluster agent becomes inoperative, another one can take over its task [80], [81]; as long as the spare agent can control the same hardware and has the same possibilities of acting.

In case of the platform agent, the situation unfortunately is not that simple. There is no agent on top of the platform agent (see Figure 3) which can take over its responsibilities when it is broken. If the platform agent stops working, it would lead to a fatal system failure [9]. A possible solution to circumvent such an error is to use a backup platform agent with the same functionality. In case of a failure of the main agent, the backup agent could turn the main agent off and work in its stead [82].

There are also software-based techniques for fault tolerance. In case of a failure in a hardware component, the application code will be adapted in a way that the faulty component(s) will not be used [83], [84]. The hardware needs to be reconfigured when using such a software-based technique [80].

Online Testing: Manufacturing variations as well as the continuously scaling of transistors lead to an increasing likelihood for a hardware defect [85], [86]. The continuously increasing complexity of system-on-chips and an ever-growing need for reducing the time-to-market period lead to a demand for reliable software based self-tests (SBST) [85], [86]. Testing methods can be deterministic, random, and hybrid (both deterministic and random) [87], [88].

Most of the state-of-the-art testing systems are not power-aware [23]. However, this is an important feature when concerning the fact that testing a core under test usually consumes more power than a core in normal mode [23]. Therefore, testing should only be done when there are enough resources left and the temperature is safe [86]. Otherwise, it can harm the chip in short-term as well as in long-term. Haghbayan *et al.* [23] propose an approach to opportunistically test dark core when there are enough resources left [23]. With this method, the system's reliability could be significantly improved while the chip is running normally [86], [89].

D. Communication Management

Because NoCs are extremely scalable and flexible in their functionality [18], NoC is the most promising on-chip communication technique for many-core systems [90]. NoCs, respectively their components, are also shrinking with each generation, and therefore, they face the same obstacles in terms of power consumption and reliability like the rest of a many-core chip [18]. Today, the power consumption of a many-core system is largely driven by on-chip networks, with their interconnects [45], [91]. Different NoC components (e.g., routers, links, network interfaces, etc.) can be reconfigured by adaptively changing routing algorithm [47], link bandwidth [92], number and formation of virtual channels [49], or even the topology (e.g., from a full mesh to an irregular mesh in case of faulty links or routers) [93]. Clock gating, power gating and even DVFS can be applied to the NoC components to adjust power consumption and to reduce heating and aging [18]. This is also considered as reconfiguration as for example scaling VF of a router not only has an impact on its power consumption but also affects the bandwidth provided by the NoC. That is why NoCs need to be self-adaptive as well as reconfigurable to improve efficiency, provide reliability, and still guarantee the performance [18]. Since 2006 these issues have forced laying the foundation of self-adaptivity as well as self-awareness of NoCs: being able to monitor the network and react accordingly [18], [94].

VI. Inferences and Discussions

The previous section described existing solutions including their results, advantages as well as drawbacks. In this section, we provide some guidelines and suggestions how to extend the mentioned approaches and algorithms.

Communication: In a hierarchical agent-based model, agents are only connected with the agents above and below them. Since the cores are connected with a network and the hierarchical agent-based model is mapped on the parallel hardware; as a consequence, all agents are connected to each other. Therefore, it is possible to establish side-connections between any two agents. While, instructions and reports will be transferred over the hierarchical instructional channels,

additional data can be send over the side channels to provide it for other agents. This technique may reduce overhead.

Safety: It would be also possible to expand the number of spare platform agents to 3 or more. It is a matter of resource-issue and can be done as long as cores are available. Furthermore, cell agents could be available in duplicate given the systems constraints are not violated.

In addition, critical agents could store their learned knowledge in a global shared space when they are in an idle mode. In the case of a malfunctioning agent, another one will be able to not only take over its task, but also it can download the knowledge of its predecessor. Furthermore, the most important knowledge could be even stored in a non-volatile storage. If the entire chip is broken and has to be replaced by a new one, the knowledge could also be transferred to the replacement chip. However, not every knowledge can be adopted; knowledge about manufacturing process variations would be mostly useless.

Design process: Designing and implementing self-aware systems need to be carried out through elaborated and tested methods. To develop such a self-aware as well as self-adaptive system from scratch would not be sensible and economic. Therefore, it would be desirable to have a toolbox that enables creating hierarchically agent-based systems which can be configured as per need. With such a toolbox it would be possible to explore different possible learning as well as abstracting algorithms and methods for different goals. As a further consequence, this toolbox can be used to implement and explore hierarchical self-aware resource management systems.

VII. Conclusion

The hierarchical agent-based approach offers flexible and scalable way to organize on-chip management of resources and various non-functional requirements. Through its hierarchical appearance, fine-grained as well as coarse-grained knowledge can be obtained. A self-adaptive system can more efficiently operate and meet the goals and expectations when it is scalable. The learning capability is necessary to cope with the changing demands and hardware and its variabilites. In addition, modularity, reuse and portability should be taken into account. These principles and techniques apply not only to on-chip systems but will also prove to be useful in many embedded- and cyber-physical systems that are distributed in nature and consist of potentially many sensor, actuator and processing nodes.

References

[1] S. Drivers, "The international technology roadmap for semiconductors (itrs)," Tech. Rep., 2009.

[2] J. Kong *et al.*, "Recent thermal management techniques for microprocessors," *In ACM Computing Surveys*, 2012.

[3] M. H. Haghbayan *et al.*, "Dark silicon aware power management for manycore systems under dynamic workloads," in *ICCD*, 2014.

[4] J. S. S. T. Association, "Failure mechanisms and models for semiconductor devices," *JEDEC Publication JEP122G*, 2010.

[5] G. Sonnenfeld *et al.*, "An agile accelerated aging, characterization and scenario simulation system for gate controlled power transistors," in *AUTOTESTCON*, 2008.

[6] P. Gupta *et al.*, "Underdesigned and opportunistic computing in presence of hardware variability," *Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 1, 2013.

[7] S. Borkar *et al.*, "Parameter variations and impact on circuits and microarchitecture," in *DAC*, 2003.

[8] S. Sarma *et al.*, "On-chip self-awareness using cyberphysical-systems-on-chip (cpsoc)," in *CODES*, 2014.

[9] L. Guang *et al.*, "Hierarchical agent monitoring design approach towards self-aware parallel systems-on-chip," *ACM Trans. Embed. Comput. Syst.*, vol. 9, no. 3, 2010.

[10] M. H. Haghbayan *et al.*, "A lifetime-aware runtime mapping approach for many-core systems in the dark silicon era," in *DATE*, 2016.

[11] M. B. Taylor, "Is dark silicon useful? harnessing the four horsemen of the coming dark silicon apocalypse," in *DAC*, 2012.

[12] H. Esmaeilzadeh *et al.*, "Dark silicon and the end of multicore scaling," in *ISCA*, 2011.

[13] H. Khdr *et al.*, "Thermal constrained resource management for mixed ilp-tlp workloads in dark silicon chips," in *DAC*, 2015.

[14] A. Rahmani *et al.*, *The Dark Side of Silicon*, 1st ed. Springer, 2016.

[15] J. Henkel *et al.*, "New trends in dark silicon," in *DAC*, 2015.

[16] F. Faniyi *et al.*, "Architecting self-aware software systems," in *WICSA*, 2014.

[17] S. M. A. H. Jafri *et al.*, "Self-adaptive noc power management with dual-level agents - architecture and implementation." in *PECCS*, 2012.

[18] J. Isoaho *et al.*, "Survey of self-adaptive nocs with energy-efficiency and dependability," *Int. J. Embed. Real-Time Commun. Syst.*, vol. 3, no. 2, 2012.

[19] C.-H. Hsu *et al.*, "Compiler-directed dynamic voltage/frequency scheduling for energy reduction in microprocessors," in *ISLPED*, 2001.

[20] Y.-W. Yang and K. S.-M. Li, "Temperature-aware dynamic frequency and voltage scaling for reliability and yield enhancement," in *ASPDAC*, 2009.

[21] J. Howard *et al.*, "A 48-Core IA-32 message-passing processor with DVFS in 45nm CMOS," in *ISSCC*, 2010.

[22] S. Teräväinen *et al.*, "Software-based on-chip thermal sensor calibration for dvfs-enabled many-core systems," in *DFTS*, 2015.

[23] M. H. Haghbayan *et al.*, "A power-aware approach for online test scheduling in many-core architectures," *IEEE Transactions on Computers*, vol. 65, no. 3, 2016.

[24] S. Huda *et al.*, "Clock gating architectures for fpga power reduction," in *FPL*, 2009.

[25] M. Shaker and M. Bayoumi, "Novel clock gating techniques for low power flip-flops and its applications," in *MWSCAS*, 2013.

[26] V. Tiwari *et al.*, "Reducing power in high-performance microprocessors," in *Design Automation Conference, 1998. Proceedings*, 1998.

[27] Q. Wu *et al.*, "Clock-gating and its application to low power design of sequential circuits," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 47, no. 3, 2000.

[28] M. Pedram, "Power minimization in ic design: Principles and applications," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 1, no. 1, 1996.

[29] K. Bernstein *et al.*, "Design and cad challenges in sub-90nm cmos technologies," in *ICCAD*, 2003.

[30] H. Singh *et al.*, "Enhanced leakage reduction techniques using intermediate strength power gating," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 15, no. 11, 2007.

[31] A. Abdollahi *et al.*, "An effective power mode transition technique in mtcmos circuits," in *Proceedings. 42nd Design Automation Conference, 2005.*, 2005.

[32] A. Mukheijee and M. Marek-Sadowska, "Clock and power gating with timing closure," *IEEE Design Test of Computers*, vol. 20, no. 3, 2003.

[33] M.-H. Haghbayan *et al.*, "MapPro: Proactive Runtime Mapping for Dynamic Workloads by Quantifying Ripple Effect of Applications on Networks-on-Chip," in *NOCS*, 2015.

[34] Y. Ge and Q. Qiu, "Task allocation for minimum system power in a homogenous multi-core processor," in *IGSC*, 2010.

[35] V. Chaturvedi *et al.*, "Thermal-aware task scheduling for peak temperature minimization under periodic constraint for 3d-mpsocs," in *RSP*, 2014.

[36] M. H. Haghbayan *et al.*, "A lifetime-aware runtime mapping approach for many-core systems in the dark silicon era," in *DATE*, 2016.

[37] S. Holmbacka *et al.*, "A task migration mechanism for distributed many-core operating systems," *The Journal of Supercomputing*, vol. 68, no. 3, 2014.

[38] P. K. Saraswat *et al.*, "Task migration for fault-tolerance in mixed-criticality embedded systems," *SIGBED Rev.*, vol. 6, no. 3, 2009.

[39] S. Bertozzi *et al.*, "Supporting task migration in multi-processor systems-on-chip: A feasibility study," in *Proceedings of the Design Automation Test in Europe Conference*, 2006.

[40] K. R. Vaddina *et al.*, "Thermal analysis of job allocation and scheduling schemes for 3d stacked noc's," in *DSD*, 2011.

[41] A. Kanduri *et al.*, "Dark silicon aware runtime mapping for many-core systems: A patterning approach," in *ICCD*, 2015.

[42] D. Cuesta *et al.*, "Adaptive task migration policies for thermal control in mpsocs," in *ISVLSI*, 2010.

[43] F. Mulas *et al.*, "Thermal balancing policy for multiprocessor stream computing platforms," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 12, 2009.

[44] A. Jantsch and H. Tenhunen, Eds., *Networks on Chip*. Kluwer Academic Publishers, 2003.

[45] S. R. Vangal *et al.*, "An 80-tile sub-100-w teraflops processor in 65-nm cmos," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 1, 2008.

[46] S. Bell *et al.*, "Tile64 - processor: A 64-core soc with mesh interconnect," in *ISSCC*, 2008.

[47] P. Lotfi-Kamran *et al.*, "Edxy-a low cost congestion-aware routing algorithm for network-on-chips," *Journal of Systems Architecture*, vol. 56, 2010.

[48] A. M. Rahmani *et al.*, "Congestion aware, fault tolerant, and thermally efficient inter-layer communication scheme for hybrid noc-bus 3d architectures," in *NOCS*, 2011.

[49] ——, "Forecasting-based dynamic virtual channel management for power reduction in network-on-chips," *Journal of Low Power Electronics*, vol. 5, no. 3, 2009.

[50] ——, "High-Performance and Fault-Tolerant 3D NoC-Bus Hybrid Architecture Using ARB-NET-Based Adaptive Monitoring Platform," *IEEE Transactions on Computers*, vol. 63, no. 3, 2014.

[51] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, 2003.

[52] N. Dutt *et al.*, "Self-aware cyber-physical systems-on-chip," in *ICCAD*, 2015.

[53] M. Salehie *et al.*, "Self-adaptive software: Landscape and research challenges," *ACM Trans. Auton. Adapt. Syst.*, vol. 4, no. 2, 2009.

[54] J. Preden *et al.*, "The benefits of self-awareness and attention in fog and mist computing," *Computer*, vol. 48, no. 7, 2015.

[55] H. J. Siegel *et al.*, "Energy-aware resource management for computing systems," in *C3*, 2014.

[56] N. Dutt *et al.*, "Toward smart embedded systems: A self-aware system-on-chip (soc) perspective," *ACM Trans. Embed. Comput. Syst.*, vol. 15, no. 2, 2016.

[57] A. Jantsch and K. Tammemae, "A framework of awareness for artificial subjects," in *CODES+ISSS*, 2014.

[58] J. L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, no. 2, 1990.

[59] Z. Chen and D. Marculescu, "Distributed reinforcement learning for power limited many-core system performance optimization," in *DATE*, 2015.

[60] P. R. Lewis *et al.*, "A survey of self-awareness and its application in computing systems," in *SASOW*, 2011.

[61] T. Becker *et al.*, "Epics: Engineering proprioception in computing systems," in *CSE*, 2012.

[62] L. Guang *et al.*, "Interconnection alternatives for hierarchical monitoring communication in parallel socs," *Microprocessors and Microsystems*, vol. 34, no. 5, 2010.

[63] C. Isci *et al.*, "An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget," in *MICRO*, 2006.

[64] J. A. Winter *et al.*, "Scalable thread scheduling and global power management for heterogeneous many-core architectures," in *PACT*, 2010.

[65] S. Borkar, "Designing reliable systems from unreliable components: the challenges of transistor variability and degradation," *IEEE Micro*, vol. 25, no. 6, 2005.

[66] J. A. Winter and D. H. Albonesi, "Scheduling algorithms for unpredictably heterogeneous cmp architectures," in *DSN*, 2008.

[67] S. Herbert and D. Marculescu, "Variation-aware dynamic voltage/frequency scaling," in *HPCA*, 2009.

[68] R. Teodorescu and J. Torrellas, "Variation-aware application scheduling and power management for chip multiprocessors," in *ISCA*, 2008.

[69] X. Wang *et al.*, "Adaptive power allocation for many-core systems inspired from multiagent auction model," in *DATE*, 2014.

[70] K. Ma *et al.*, "Dppc: Dynamic power partitioning and control for improved chip multiprocessor performance," *IEEE Transactions on Computers*, vol. 63, no. 7, 2014.

[71] K. Ma and X. Wang, "Pgcapping: exploiting power gating for power capping and core lifetime balancing in cmps," in *PACT*, 2012.

[72] A. Sharifi *et al.*, "Pepon: performance-aware hierarchical power budgeting for noc based multicores," in *PACT*, 2012.

[73] Y. Ge *et al.*, "A multi-agent framework for thermal aware task migration in many-core systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 10, 2012.

[74] V. Hanumaiah *et al.*, "Throughput optimal task allocation under thermal constraints for multi-core processors," in *DAC*, 2009.

[75] R. Rao and S. Vrudhula, "Efficient online computation of core speeds to maximize the throughput of thermally constrained multi-core processors," in *ICCAD*, 2008.

[76] C. Addo-Quaye, "Thermal-aware mapping and placement for 3-d noc designs," in *SOC Conference*, 2005.

[77] Y. Cheng *et al.*, "Thermal-constrained task allocation for interconnect energy reduction in 3-d homogeneous mpsocs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 2, 2013.

[78] C. Sun *et al.*, "Three-dimensional multiprocessor system-on-chip thermal optimization," in *CODES+ISSS*, 2007.

[79] X. Wang *et al.*, "A low cost, high performance dynamic-programming-based adaptive power allocation scheme for many-core architectures in the dark silicon era," in *ESTIMedia*, 2013.

[80] S. Müller *et al.*, "A multi-layer software-based fault-tolerance approach for heterogenous multi-core systems," in *LATS*, 2015.

[81] S. Shamshiri *et al.*, "A cost analysis framework for multi-core systems with spares," in *ITC*, 2008.

[82] P. Rantala *et al.*, "Agent-based reconfigurability for fault-tolerance in network-on-chip," in *ERSA*, 2007.

[83] A. Meixner and D. J. Sorin, "Detouring: Translating software to circumvent hard faults in simple cores," in *DSN*, 2008.

[84] M. Schölzel, "Software-based self-repair of statically scheduled superscalar data paths," in *DDECS*, 2010.

[85] M. Kaliorakis *et al.*, "Accelerated online error detection in many-core microprocessor architectures," in *VTS*, 2014.

[86] M. H. Haghbayan *et al.*, "Online testing of many-core systems in the dark silicon era," in *DDECS*, 2014.

[87] M. Psarakis *et al.*, "Systematic software-based self-test for pipelined processors," in *DAC*, 2006.

[88] T. H. Lu *et al.*, "Effective hybrid test program development for software-based self-testing of pipeline processor cores," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 3, 2011.

[89] M. Shafique *et al.*, "The eda challenges in the dark silicon era," in *DAC*, 2014.

[90] J. Howard *et al.*, "A 48-core ia-32 processor in 45 nm cmos using on-die message-passing and dvfs for performance and power scaling," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 1, 2011.

[91] L. Shang *et al.*, "Dynamic voltage scaling with links for power optimization of interconnection networks," in *HPCA*, 2003.

[92] Y. C. Lan *et al.*, "A bidirectional noc (binoc) architecture with dynamic self-reconfigurable channel," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 3, 2011.

[93] F. J. *et al.*, "An efficient implementation of distributed routing algorithms for nocs," in *NOCS*, 2008.

[94] C. Ciordas *et al.*, "A monitoring-aware network-on-chip design flow," in *DSD)*, 2006.