On-Chip Dynamic Resource Management

Antonio Miele[1], Anil Kanduri[2], Kasra Moazzemi[3], Dávid Juhász[4,5], Amir M. Rahmani[3,4], Nikil Dutt[3], Pasi Liljeberg[2] and Axel Jantsch[4]

[1] *Politecnico di Milano, Italy;*
[2] *University of Turku, Finland;*
[3] *University of California, Irvine, USA;*
[4] *TU Wien, Vienna, Austria;*
[5] *Imsys AB, Stockholm, Sweden;*

ABSTRACT

The need for dynamic resource management has shadowed the exponential growth of on-chip transistor capacity, and the challenge is accentuated by the heterogeneity of resources, and the bewildering variety of constraints and requirements of applications, platforms and users. The field has started with a few research papers in the early 1990s but has grown today to over hundred yearly publications, leading to an accumulated body of literature presumable far above 1000 papers.

We focus on the *dynamic (run-time)* management of *on-chip* resources and mostly ignore design-time techniques and off-chip resources of larger electronic systems. Moreover, we do not attempt a complete review of all published work on the topic. Rather, this survey provides a structured review and discussion of the state of the art and is divided along the primary objectives of resource management techniques: performance, power, reliability and quality of service, each

Antonio Miele, Anil Kanduri, Kasra Moazzemi, Dávid Juhász, Amir M. Rahmani, Nikil Dutt, Pasi Liljeberg and Axel Jantsch (2019), "On-Chip Dynamic Resource Management", Foundations and Trends® in Electronic Design Automation: Vol. 13, No. 1-2, pp 1–144. DOI: 10.1561/1000000055.

The version of record is available at: http://dx.doi.org/10.1561/1000000055

of which has its dedicated chapter. We observe that many works describe dedicated techniques for point problems, like minimizing power or maximizing lifetime. In recent years more and more methods have started to appear that address two, three or more different goals of resource management, but work with a holistic scope that attempts to address and balance all relevant objectives is still rare. An exception is the area of reliability and life-time management. There we see that a large majority of approaches may be considered as holistic.

Observing current limitations and recent trends we assume that global and holistic approaches will make the most valuable contributions to this field in the years to come and therefore we expect that the focus of research will gradually shift to systematic methods combining point techniques in order to pursue and balance all relevant system goals of highly dynamic, adaptive systems.

The version of record is available at: http://dx.doi.org/10.1561/1000000055

Acronyms

ALU Arithmetic Logic Unit. 15

AVF Architectural Vulnerability Factor. 73, 74

BIST Built-In Self-Test. 79

BPIO Balanced Placement I/O. 37

CPU Central Processing Unit. 15, 28, 38, 57, 68, 69, 87, 98

DMA Direct Memory Access. 16

DPM Dynamic Power Management. 16, 19, 41, 43, 44, 45, 46, 47, 48, 49

DRAM Dynamic Random-Access Memory. 29, 93

DRM Dynamic Reliability Management. 59, 62, 63, 65, 68, 69, 70, 71

DSP Digital Signal Processor. 13, 15

DTM Dynamic Thermal Management. 41, 51, 52, 59

DVFS Dynamic Voltage and Frequency Scaling. 16, 40, 44, 46, 51, 53, 57, 62, 63, 64, 65, 67, 68, 71, 73, 83

DWC Duplication With Comparison. 76, 78, 81

EDF Earliest Deadline First. 48

EDP Energy Delay Product. 49

EM Electromigration. 57, 58, 60, 61, 65, 67, 68, 70

EPI Energy Per Instruction. 49

FeRAM Ferroelectric Random-Access Memory. 29

FinFET Fin Field-effect. 61, 64

FPGA Field Programmable Gate Array. 15, 52, 83, 93

FPU Floating-Point Unit. 63, 82

GPP General Purpose Processor. 13, 15

GPU Graphics Processing Unit. 15, 19, 28, 38, 59, 68, 69

HCI hot carrier injection. 58, 68, 69

HMP Heterogeneous Multi-core Processors. 47

HPC High Performance Computing. 49, 55, 78

I/O Input/Output. 13, 15, 16, 20, 21, 24, 36, 37, 38, 84, 85, 86, 88, 89

ILP Integer Linear Programming. 45

IPS Instruction Per Second. 26

ISA Instruction Set Architecture. 41, 93

MIMO Multiple-Input Mutiple-Output. 47

MRAM Magnetic Random-Access Memory. 29

MTTF Mean Time To Failure. 59, 60, 61, 62, 63, 64, 65

NBTI Negative Bias Temperature Instability. 57, 58, 60, 61, 66, 67, 68, 69, 70

NMR N-Modular Redundancy. 77

NoC Networks-on-Chip. 8, 16, 30, 31, 33, 42, 46, 53, 59, 60, 62, 65, 67, 77, 81

PCM Phase-Change Memory. 29

PE Processing Element. 13, 15, 16

PI Proportional-Integral. 64

PID Proportional-Integral-Derivative. 64

PLR Process-Level Redundancy. 77

QoR Quality of Result. 94

QoS Quality of Service. 8, 10, 11, 14, 17, 18, 81, 84, 85, 86, 87, 88, 89, 90, 91, 94, 95, 97, 99, 103

RRAM Resistive Random-Access Memory. 29

RTOS Real-Time Operating Systems. 50

SBST Software-Based Self-Test. 79, 80

SMT Simultaneous Multi-Threading. 51, 77

SoC Systems-on-Chip. 13, 42, 45

SRAM Static Random-Access Memory. 29, 50

SSD Solid State Disk. 36, 37, 38

STP System Through-Put. 26

STT-RAM Spin-Transfer Torque Random-Access Memory. 29

TDDB Time Dependent Dielectric Breakdown. 57, 58, 60, 70

TMR Triple Modular Redundancy. 77, 78, 79, 81

WCET Worst-Case Execution Time. 45

The version of record is available at: http://dx.doi.org/10.1561/1000000055

1

Introduction

Resource management has a long history in computing, from the early days of time-shared machines with pioneering fundamental work on run-time systems, distributed systems, real-time operating systems and middleware. The evolution in computing architectures – from single- to multi- and many-core platforms – has resulted in a Cambrian explosion in the diversity of computing architectures, applications and end-use cases, due to:

- Chips are getting more complex, with:
  - Increasing core count
  - Increasing core heterogeneity
  - Novel memory technologies and architectures
  - Disparate interconnects and I/O
- Workloads are getting more diverse and unpredictable (e.g., mobile to edge to data center)

- There is a renewed move towards programmable architectures tuned to certain domains (e.g., mobile, edge, cloud) and applications (neural networks, deep learning, security, cryptography, etc.)

- Computing systems must increasingly satisfy multi-dimensional constraints that straddle multiple metrics: performance, power/energy, temperature, reliability, lifetime, Quality of Service (QoS), etc. Furthermore, these constraints themselves may evolve over time (e.g., high-performance at times, throttling for impending thermal emergencies, and strategies for extending lifetime in the face of wear-out)

These trends have generated an incredibly large body of work for on-chip resource management in the past two decades, that have focused on many different combinations of architectures, workloads, constraints and use-cases. Indeed, due to large variations in the assumptions, use of different terminology, metrics, goals, and use-cases, anyone attempting to review the literature can easily get overwhelmed by the volume, diversity, and sometimes even seemingly contradictory approaches for on-chip resource management. Furthermore, advances in program analyses, system modeling, run-time monitoring, model building, and machine learning have generated new lines of research in dynamic and adaptive on-chip resource management that further complicate a global understanding of the current state-of-the-art and emerging directions for on-chip resource management.

A number of surveys offer systematic overviews of parts of the topics, challenges and techniques that we are concerned with. Some of them focus on specific subsystem of a many-core chip like Networks-on-Chip (NoC) (Bjerregaard and Mahadevan, 2006; Marculescu et al., 2009; Rahmani et al., 2010; Radetzki et al., 2013) or caches (Scolari et al., 2014), others deal systematically with particular metrics like power, energy (Maiterth et al., 2018), or aging (Khoshavi et al., 2017). Also, scheduling, mapping and task allocation have received a fair amount of attention in surveys (Singh et al., 2013a, 2017; Zhuravlev et al., 2012; Burns and Davis, 2017), with focus either on performance

Table 1.1: A selection of surveys.

Approximate computing	Mittal (2016); Xu et al. (2016a)
NoC	Bjerregaard and Mahadevan (2006)
NoC design	Marculescu et al. (2009)
3D NoC	Rahmani et al. (2010)
Fault tolerant NoC	Radetzki et al. (2013)
Cache management	Scolari et al. (2014)
Power management in high performance systems	Liu and Zhu (2010)
Energy and power aware job scheduling	Maiterth et al. (2018)
Thermal management	Kong et al. (2012)
Aging mitigation	Khoshavi et al. (2017)
Dark silicon	Shafique and Garg (2017)
Self-aware SoCs	Jantsch et al. (2017)
Energy and power aware job scheduling	Maiterth et al. (2018)
Mapping in many core systems	Singh et al. (2013a)
Resource allocation in hard- and soft- real-time systems	Singh et al. (2017)
Scheduling	Zhuravlev et al. (2012)
Mixed Criticality systems	Burns and Davis (2017)
Resource management in clouds	Jennings and Stadler (2015)

or real-time behavior. Current trends and opportunities like approximate computing (Mittal, 2016; Xu et al., 2016a), the dark silicon phenomenon (Shafique and Garg, 2017) and comprehensive self-monitoring on-chip (Jantsch et al., 2017) have recently inspired researchers to review, analyze and compare relevant work. But none of them, as Table 1.1 illustrates, covers all the aspects of on-chip resource management in a holistic way and many of them also include both design time and run time methods. It should be noted that any resource allocation

approach contributes to multiple metrics (e.g., energy, temperature, QoS, etc.) and cannot be adequately discussed in isolation. Therefore, a comprehensive review is needed to discuss the relation between different categories of on-chip resource management techniques and the metrics of interests in a holistic fashion.

This article attempts to cover all aspects of on-chip run-time resource management to facilitate understanding of recent trends in dynamic and adaptive strategies. We have organized the article into three major sections (also visualized in Figure 1.1):

1. Chapter 2 presents a taxonomy of on-chip resources, design metrics, objectives and constraints. This categorization helps the reader visualize the relationship between different resource categories (e.g., computing, storage, communication) and design metrics (e.g., performance, power/energy, temperature, QoS, lifetime, etc.). This chapter also relates the role of objectives and constraints that guide resource management policies.

2. Chapters 3 through 6 survey literature in dynamic on-chip resource management through the lens of the primary metrics that drive these bodies of work:

    - Chapter 3 reviews efforts focused on the traditional metric of optimizing system performance. Historically performance has been the major driving metric for computing platforms, and for many scenarios it continues to be an important design metric
    - Chapter 4 covers resource management techniques that address the metrics of power, energy and temperature. As the core count increased, the move to multi- and many-core architectures rapidly hit the power wall, resulting in a large body of work on power-aware resource management strategies via both hardware and software techniques. Concurrently, research on run-time energy efficiency gained traction via mapping and scheduling approaches. Thermal management techniques began to appear in the past decade, with the goal

of operating chips at a safe temperature via dynamic thermal management techniques.

- Chapter 5 surveys the large body of work on reliability, via the facets of lifetime management, soft-error resilience, and online fault management. Modern chips are increasingly susceptible to failures from a diverse set of causes, leading to premature lifetime failure (due to aging and wear-out) or intermittent/transient failures (due to soft errors). Numerous techniques have been developed to mitigate these failures. Additionally, there is a large body of work covering online fault management that dynamically detects, and proactively applies fault management strategies to prevent failures.
- Chapter 6 addresses QoS metrics that drive several important classes of applications. While QoS can be a nebulous "qualitative" term, we first give specific examples of QoS for different applications, and relate it to the specific metrics that drive the QoS. Here we survey resource management techniques from two angles: a) performance-bound QoS techniques that achieve desired QoS by controlling resources (e.g., compute, memory, I/O), and b) accuracy-bound QoS for applications that can tolerate some loss of accuracy in exchange for sacrificing resources and/or metrics via both static and dynamic strategies.

Note, that each chapter includes several tables listing all the approaches discussed in the various sections. The goal of these tables is to summarize the discussion and provide the taxonomy of the various approaches based on the specific aspects discussed in the text. Since the discussion in each section is specific to the aspects of the presented subtopic, each table has a custom organization of the rows and columns.

3. Chapter 7 addresses the limitations of existing work and outlines recent trends in enabling adaptive resource management in the face of dynamically varying workloads, system properties, and unforeseen environmental effects. Samples of emerging topics in

dynamic resource management are highlighted as examples of ongoing challenges facing researchers in this domain.

Chapters 3, 4, 5, and 6 can be read independently of each other and in any order. Hence, 2, 4, 7 would be reasonable flow of reading as would be 2, 6, 7, depending on the interest of the reader.

While we have attempted to do a comprehensive survey, it is by no means complete, but should provide the reader with a framework within which to navigate both existing, as well as evolving research efforts in on-chip dynamic resource management.

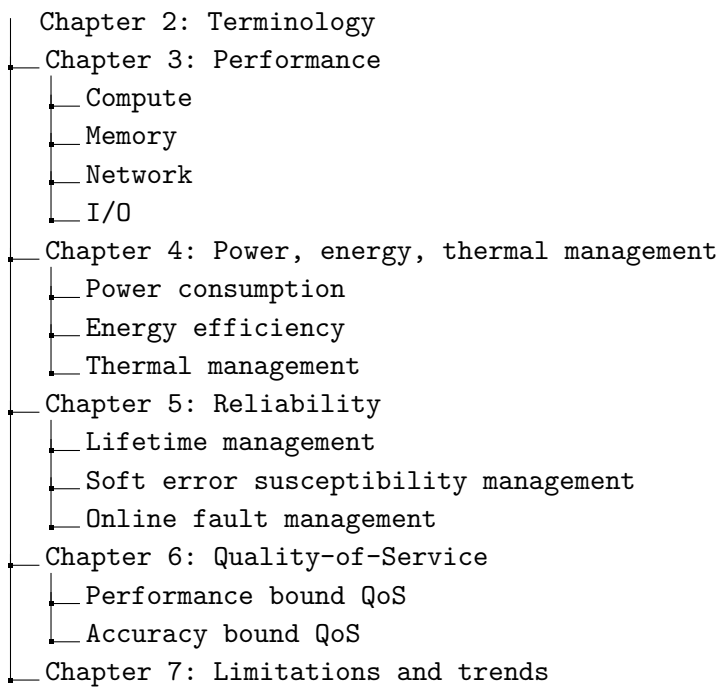

Figure 1.1: Article structure.

The version of record is available at: http://dx.doi.org/10.1561/1000000055

2

On-chip resources

As processing capacity and complexity of Systems-on-Chip (SoC) increases, the number, and, most importantly, the types of resources increase. State-of-the-art SoCs consist of tens or hundreds of Processing Elements (PEs) ranging from General Purpose Processor (GPP) cores to Digital Signal Processors (DSPs) to special purpose video encoders and encryption engines. Data flow is facilitated by a set of connected buses and packet switched networks that deliver the data to PEs at the rate and time needed. A variety of storage elements such as register files, caches and buffers smooths the flow of data through the system and makes sure the system does not have to grind to a halt when just one data item has not yet arrived. In addition, Input/Output (I/O) resources connect the on-chip processing to the vast off-chip memories and external sensors and actuators.

For understanding the scope of on-chip resource management, we need an inventory of the concepts that are related to resource management. We identify two main concepts in Figure 2.1:

1. *resources* are subject to allocation choices and control decisions;
2. *metrics* describe non-functional characteristics of the system.

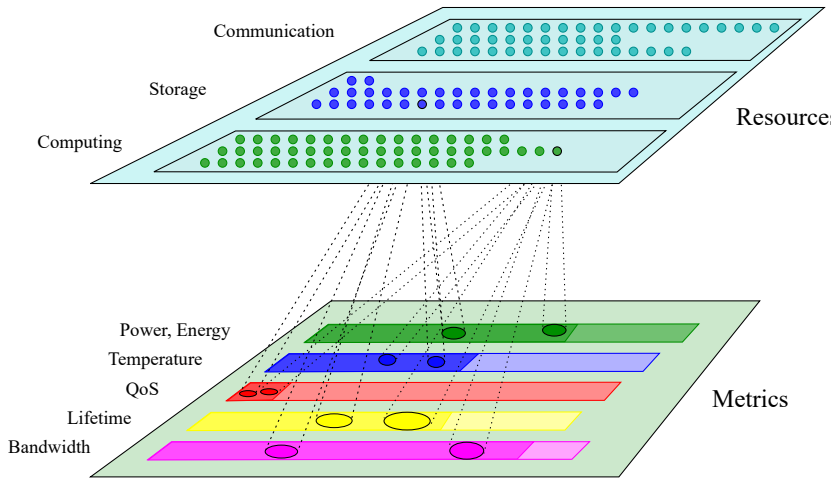

Figure 2.1: The upper plane represents resources, e.g. specific hardware, while the lower plane shows metrics. The allocation and usage of *each* resource throws shadows into the plane below, representing how it contributes to different metrics. The drawing illustrates a few shadows only but each and every resource contributes to the actual values of metrics. Whether any particular metric is positive or negative depends on the context i.e., the considered problem.

When a resource (Section 2.1), an actual hardware block, is activated and executes a task, its operation can be characterized by various metrics (Section 2.2), like consumed energy, generated heat, latency, bandwidth, or a Quality of Service (QoS) metric.

For understanding resource management techniques, we need an inventory of concepts that are used for defining and solving resource management problems. For a given resource management problem, a metric may serve as *objective* or *constraint* (Section 2.3). The same metric may be used in different roles for different problems. Resource management techniques utilize *observation* to collect information of the current state of the system and *prediction* to estimate its future behavior (Section 2.4).

2.1 Resources

Resources are physical entities that can be allocated to tasks by resource management. Control decisions of resource management also tune the

operation of resources, which affects operation of the system and hence the metrics.

Allocation choices of resource management are binary: a resource is either allocated to a task at a time or not. However, a resource may have the capacity to serve more than one task if the hardware is provided. Each resource has a maximum capacity of tasks that it can serve at a given time, which depends on its hardware structure. For instance, a Central Processing Unit (CPU) with two Arithmetic Logic Units (ALUs) could process two instructions at the same time.

Examples of resources are cores, buses, I/O pins, links, and memory locations. A link between two routers can be allocated to only one packet at any given time. As another example, a router in a communication network can only process four packets simultaneously coming from its four input ports. An on-chip network may be used by several tasks simultaneously at an abstract level but each physical component such as a register, a link, or a switch-box can serve only one or a limited number of packets at any moment. The network as a whole can transmit many packets and it can be considered as one communication resource with a maximum capacity of C bits/sec. Fractions of C can be allocated to individual users of the network by resource management which initiates communication over the network but the actual route and scheduling of the transmission is managed by the routers.

Table 2.1: Examples of resources

Category	Examples
Computation resources	CPU, GPU, DSP, FPGA
Communication resources	buses, networks, I/O pins, interrupt controllers
Memory resources	main memory, cache, register file

Resources can be divided into 3 sub-categories (Table 2.1) according to the functionality they provide as follows.

Computation resources are PEs which perform tasks. A PE can be a GPP, a dedicated accelerator e.g., Graphics Processing Unit (GPU) and DSP, or reconfigurable e.g., Field Programmable Gate Array (FPGA),

that are able to serve different purposes. Different PEs support various control features which are to be decided by the resource management. PEs may support different Dynamic Power Management (DPM) techniques, some provide hardware knobs for tuning the precision of arithmetic computations, reconfiguration is a point of control for reconfigurable PEs. Computation resources may be managed on the core level, but are often treated on a more abstract level as a pool of cores for particular resource management techniques.

Communication resources are utilized by tasks to exchange information with other tasks or the environment. Communication resources include buses, networks, I/O pins, and interrupt controllers. On-chip networks are complex resources, which are built up from several physical blocks implementing the functionality. Those internal resources, like buffers and routers, must be revealed for efficient resource management. Nevertheless, Networks-on-Chips (NoCs) are also managed as a whole without considering actual hardware blocks. Communication resources may support various control features e.g., Dynamic Voltage and Frequency Scaling (DVFS) for NoCs.

Memory resources are used by tasks to store and retrieve data. Memory resources are typically organized in a hierarchical structure. some storage resources, e.g. on-chip main memory and off-chip memories, are under software control, while others, e.g. the cache hierarchy, operate in a hardware-defined manner. Off-chip resources are out of the scope of this survey. It is also noteworthy that while off-chip memory resources are under software control, interfaces between them might involve logic beyond that control e.g., hardware-controlled buffers and Direct Memory Access (DMA) facilities. DPM techniques, typically power gating, may be supported by memories.

2.2 Metrics

Non-functional characteristics of the system are represented in various metrics: scales that provide means to evaluate particular aspects of the operation of the system. The metrics considered in the survey are

- performance (Chapter 3),
- power, energy, temperature (Chapter 4),
- reliability (Chapter 5),
- QoS (Chapter 6).

While the following chapters are named after metrics, the paper is organized according to objectives rather than metrics. The following chapters are dedicated to techniques that optimize a metric as objective (Section 2.3) — rather than to all techniques that relate to a metric in general. Certain techniques and topics (e.g., mapping and scheduling) may be relevant for optimizing various metrics and hence may appear in multiple chapters — with independent description in each chapter.

About the connection between resources and metrics, note that metrics characterizing one resource are typically interdependent. For example, scaling up frequency of a computation resource results in higher power dissipation and temperate but also increases performance.

2.3 Objectives and constraints

The subjects of resource management are the resources which are directly controlled by allocating tasks and tuning their operation. However, all actions of resource management are based on metrics that characterize the operation of the system.

Resource management makes its allocation choices and control decisions with respect to objectives to be accomplished. The objectives are defined as maximizing or minimizing some metrics. By allocating and controlling resources properly, resource management steers the system to meet requirements and target goals on *objective metrics*.

Allocation and control decisions of resource management might also have negative consequences. Particularly, over-utilizing the system, leading to excessive energy consumption and higher temperature, degrades system health and affects desired objectives negatively. While aiming at meeting requirements with respect to objective metrics, resource management must also take constraints on other metrics into account.

Proper resource management is based on heuristics aiming at containing some metrics below or above given limits and optimizing others.

Note that metrics cannot be generally sorted into groups of objectives and constraints as their role is dependent on the considered resource management technique. The surveyed techniques are structured according to their objective metrics.

2.4 Observing and predicting

Resource management is to optimize and contain some problem-specific characteristics of the system, that is metrics. Decisions need to be made so that metrics exhibit a desirable behavior in the future. Also, decisions can be made based on the current and possibly past states of the system.

The current state of the system is observed by sensors and future behavior is predicted by models. Note that the terms sensing and monitoring are used in literature with similar meaning to observation.

Sensors provide means for resource management to observe the current state of the system as in current values of metrics. Beyond taking notice of the current state, historical data can be collected over time.

We can identify physical sensors (e.g., power sensor, temperature sensor, hardware performance counter) and cyber sensors (e.g., abstract QoS, instrumented software, logs). Example of physical sensors can be the sensors used in SoCs that require several die temperature sensors (Vogt et al., 2007) to be integrated in a chip to manage the performance because die temperature directly affects leakage current level and performance of clock-based digital circuits (Hwang et al., 2010). Cyber sensors can provide a standard method for an application to directly communicate its performance and goals. As an example, Hoffmann et al. (2010) provide a framework that allows applications to express their performance in terms of a desired heart rate and/or a desired latency between specially tagged heartbeats.

Models are used to predict the future behavior of the system based on collected state information and planned control actions. Resource management uses models to evaluate possible control actions with

respect to past behavior, current state, and desired future behavior of the system.

Models may be used explicitly or implicitly in resource managers. An explicit model is present as a component of the resource manager and does compute its predictions whenever required. Different models may provide different accuracy because of predicting with different levels of details being taken into account. The more the details and accuracy, the more the computational complexity. The trade-off between accuracy of prediction and computational complexity is to be tuned with respect to the problem at hand.

Predictive models help to tune performance for varying workload. Gupta et al. (2016) predicts GPU performance online as DPM algorithm tunes GPU frequency. Proactive techniques are enabled by predictive models and achieve better quality of control than reactive methods. Temperature is predicted by Coskun et al. (2009) and the proactive thermal control that is based on the prediction realizes improved thermal profiles. Prediction may be used in relation to multiple metrics simultaneously, for example for predictive management of temperature and power (Singla et al., 2015).

In some cases, accurate-enough and yet relatively simple rules can be derived from abstract – perhaps simplified or estimate – models. Resource management may use such rules directly as part of the decision mechanism. Instead of implementing the underlying model explicitly, the model – as well as prediction – is implicit behind the incorporated rules. Such rules are called heuristics.

The version of record is available at: http://dx.doi.org/10.1561/1000000055

3

Performance

Execution time of a *workload* - application, task, thread, basic block or instruction - represents the *performance* metric on a given system. Accelerating a computational block within a workload improves latency and throughput. Achieving higher performance with efficient resource utilization has multi-fold impact on other system metrics such as power, energy and temperature - making performance optimization significant. Overall execution time of a workload can be broken down into time spent in computation, accessing memory, acquiring shared resources of network and Input/Output (I/O) peripherals. Run-time performance optimization techniques target to minimize at least one or more of these latencies, by provisioning compute, memory, network and I/O resources. In this section, we present such run-time techniques with improving overall performance as their primary objective, classifying them as per the resource they optimally allocate. Figure 3.1 shows the abstract classification of performance optimization techniques, detailed in the following sub-sections.

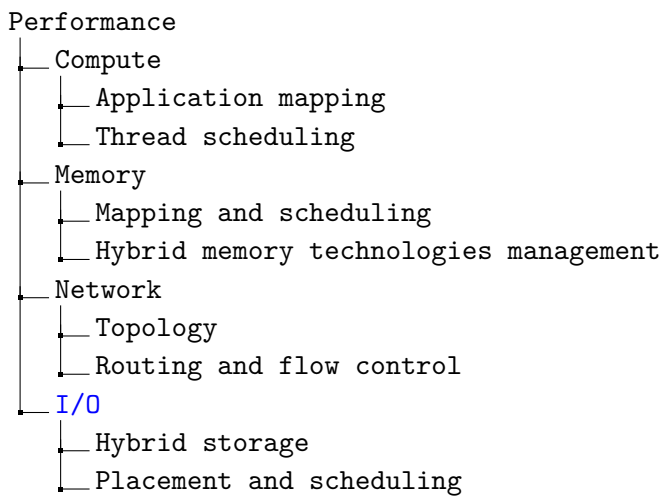

Figure 3.1: Classification performance through provisioning tangible resources

3.1 Compute

We focus on techniques that bring performance gains through allocation and provisioning of computational resources. These include *application mapping* - techniques to identify and allocate enough cores to the given task, optimal application placement to minimize network and memory latency, smart co-scheduling of concurrent applications to minimize contention, and *thread scheduling* - techniques to bind thread-to-core based on application-core match (in case of heterogeneity or asymmetry) and adjusting parameters such as voltage, frequency, CPU utilization time.

3.1.1 Application mapping

Given a tiled many-core architecture, placement of applications (or tasks within an application) on each core significantly effects the overall performance and energy consumption. Hu and Marculescu (2003) have identified and formalized the application mapping problem as the appropriate choice of task-to-core selection in order to maximize performance, minimize latency and energy consumption. The combinations of cores, private caches, shared last cache levels, memory controllers and on-chip

interconnects lead to complex shared resource contention in many-core systems (Chen et al., 2008). Considering these aspects, the choice of cores, memory and, importantly, their spatial alignment impacts performance significantly (de Souza Carvalho et al., 2010). While early solutions to mapping problem focused on minimizing energy consumption Hu and Marculescu (2003), automation of topology and mapping phases Murali et al. (2004), they are largely static in nature. As the number of cores and concurrent applications contending for resources keep increasing, dynamic policies for application-to-core mapping have become more relevant (Singh et al., 2013a).

Mapping policies' pre-requisite is core selection - to find a subset of cores among all the on-chip cores, which can be used to schedule an application (Bender et al., 2008). Existing mapping techniques largely rely on monitoring a binary core status - occupied or un-occupied, to track a list of available cores on the chip (Fattah et al., 2012; Haghbayan et al., 2015; Fattah et al., 2013) and application characteristics to develop heuristics for mappings (Xue et al., 2006; Shojaei et al., 2009). Once suitable number of cores are found, mapping strategies try to optimize for performance, minimal latency and energy consumption. Among the available cores, the choice of binding to specific cores depends on application's task level parallelism in order to allocate each task/thread to a core in an efficient manner (de Souza Carvalho et al., 2010). Some techniques use static profiling Xue et al. (2006) while others use fully dynamic monitoring for run-time mapping decisions. The common aspect among majority of run-time mapping frameworks is using a centralized control for core selection. On the other hand, Faruque et al. (2008) have proposed a distributed agent-based mapping strategy that emphasizes on scalability and adaptability.

Core selection is further influenced by other performance objectives such as minimizing total communication volume (Bender et al., 2008), network congestion among tasks of an application (Carvalho et al., 2007), interference between concurrent applications (Fattah et al., 2012) and maintaining regular geometric structure of resources by avoiding dispersion and fragmentation (Fattah et al., 2014). Bender et al. (2008) have proposed weighted Manhattan distance as a metric to quantify communication penalty among tasks of an application and

used this for an iterative task-by-task mapping. Minimizing intra-task communication distance leads to proximity metrics and reduces internal congestion (Carvalho et al., 2007). Further, mapping applications in regular geometric structures, preferably squared, facilitates optimal core selection for subsequent applications to be mapped, as suggested by Fattah et al. (2014). This problem was first addressed through an incremental mapping approach Chou et al. (2008), which considers other concurrently running applications while deciding on mapping of an incoming application. Such strategies avoid dispersion among available set of cores and minimizes the likeliness of fewer isolated cores which potentially could be under utilized. In a similar vein, mapping concurrent applications contiguously can also prevent dispersion or fragmentation of remaining available cores (Fattah et al., 2012). Since most of the mapping policies attempt to solve an NP-hard problem, a heuristics based core selection was proposed by Fattah et al. (2013). This hill climbing approach splits mapping into two steps viz., first node selection - to identify a potential favorable spatial location on the chip, followed by task allocation - to bind tasks to cores within the selected region.

While most of the aforementioned techniques are reactive, a proactive mapping strategy was presented by Haghbayan et al. (2015) by keeping track of a list of available first nodes that can suit any kind of incoming applications. It is to be noted that application mapping policies are largely targeted at applications that are task graph and message passing based, exhibiting higher degree of task level parallelism and concurrency. Further, adaptive mapping policies have considered factors such as process variation (Hong et al., 2009). A summary of application mapping as per their core selection criteria is presented in Table 3.1.

3.1.2 Thread scheduling

Application mapping techniques focus on identification of a subset of cores to map, considering application level properties. Thread scheduling lowers the abstraction to identify task-level performance metrics

Table 3.1: Application mapping techniques

Core selection criteria	Techniques
Availability	Fattah et al. (2012), Chen et al. (2008), Bender et al. (2008), Bender et al. (2008), Haghbayan et al. (2015)
Min. latency	Haghbayan et al. (2015), Fattah et al. (2012), Chen et al. (2008), Castrillon et al. (2012), Kanduri et al. (2015), Shojaei et al. (2009)
Max. throughput	Hong et al. (2009), Singh et al. (2013b), Singh et al. (2011), Zipf et al. (2009), Huang and Xu (2010), Kanduri et al. (2015)
Min. energy consumption	Faruque et al. (2008), Hu and Marculescu (2003), Murali et al. (2004), Chou et al. (2008), Shafique et al. (2014), Hu and Marculescu (2005)
Preserve regular structure	Fattah et al. (2013), Fattah et al. (2014), Fattah et al. (2012), Haghbayan et al. (2015)

and allocate resources at a per-thread granularity. This becomes relevant particularly with asymmetric, heterogeneous and non-monotonic cores with a wide range of power-performance characteristics to choose from (Navada et al., 2013). Scheduling techniques rely on low level metrics such as memory, compute and I/O access patterns and bandwidth requirements, and performance characteristics of underlying hardware to bind threads to cores for higher performance (Kumar et al., 2005). The key idea among all scheduling techniques is that thread level properties of an application guides the choice of suitable compute resources, which can provide relatively higher performance.

Chen and Guo (2014) have used a history based prediction of applications' performance on a core type to determine its suitability for mapping on a specific type of core. Van Craeynest et al. (2012) collect performance metrics of cycles-per-instruction, instruction-level-parallelism and memory-level-parallelism at run-time to estimate resource requirements of an application and then determine suitable thread binding rules. A similar approach was also used by Gaspar et al. (2014) where execution time is collected at run-time for potential thread allocation predictions.

Apart from thread binding based on application's affinity, a class of techniques identify bottleneck regions within an application to suitably allocate resources for acceleration. Kumar et al. (2005) identify serial regions of an application to allocate high performance larger cores for

such blocks while smaller cores are allocated for data and task parallel blocks. Identifying critical sections of code resulting in frequent stalls with synchronization issues, isolating and accelerating such regions leveraging asymmetric cores is proposed by Suleman et al. (2009). A similar approach, used by Joao et al. (2012), targets *bottlenecks* as the critical sections within in a thread which also effect other concurrent threads. Further, applications can exhibit a variation in workload with interleaved phases of compute and memory intensity, and critical and non-criticality. Adapting thread allocation to such phase change behavior by dynamically altering core choice appropriately was presented by Annamalai et al. (2013).

In addition to application's properties and thread level performance metrics, some techniques use a compound *utility* metric that combines application properties, threads' performance metrics, hardware suitability and resource efficiency. Saez et al. (2010) proposed comprehensive thread allocation approach by combining core and un-core compute capacity required and the degree of parallelism to decide on optimal thread placement. Koufaty et al. (2010) use profiling to determine a threads' bias to a type of core based on performance and the extent of resource utilization. Joao et al. (2013) used similar resource utilization measures to determine thread allocation decisions.

While most of the above techniques rely on provisioning, Torng et al. (2016) proposed work stealing to accelerate long latency threads and simultaneously utilize available resources efficiently. Scheduling techniques as per identification thread-to-core affinity are summarized in Table 3.2.

Table 3.2: Thread scheduling techniques

Core selection strategy	Techniques
Static	Koufaty et al. (2010), Suleman et al. (2009), Navada et al. (2013), Kumar et al. (2005)
Adaptive	Torng et al. (2016), Joao et al. (2013), Saez et al. (2010), Joao et al. (2012), Gaspar et al. (2014), Annamalai et al. (2013)

3.2 Memory

With increase in frequency and number of cores in emerging systems, rate and latency of access to data became a deciding factor of system performance. This data can be accessed from any point in the memory hierarchy i.e., registers, cache, main memory or disk. There have been many advances regarding the memory hierarchy in the recent years. We can categorize these improvements in two main classes. First, the management algorithms to take advantage of the existing memory structures. Second, the advancements in technology used in memory components. Some of these efforts increase the performance and response time of these components. Srinivasan and Lebeck (1998) discuss the effect of memory latency on performance of dynamically scheduled processors. On the other hand, some methods like those proposed by Ha et al. (2017) and Ahn et al. (2014) benefit from emerging technologies to make memory devices more energy efficient delivering higher performance. In this context, we focus on the techniques engineered to improve overall performance of the system leveraging the inherent characteristics of the memory hierarchy.

Assessing the performance of multi-program workloads running on a multi-threaded hardware is difficult because it involves balance between single program performance and overall system performance. This issue is common in current multi-core systems. Eyerman and Eeckhout (2008) argue for developing multi-program performance metrics in a top-down fashion starting from system-level objectives. This way the impact of memory accesses on overall performance of the system is considered. The authors proposed two performance metrics: Average Normalized Turnaround Time (ANTT) is a user-oriented performance metric and System Through-Put (STP) is a system oriented performance metric. STP corresponds to the weighted speedup metric and ANTT corresponds to the reciprocal of the mean metric. These two proposed metrics can replace Instruction Per Second (IPS) to better represent overall system performance. In addition, extracting performance metrics of the systems in a way that doesn't effect the execution of the programs might not be trivial. In order to avoid this issue, Nazari et al. (2017) and Sehatbakhsh et al. (2016) proposed new methods for profiling program

execution without instrumenting or otherwise affecting the profiled system.

Table 3.3: Memory management techniques

Memory management	Techniques
Mapping	Kim et al. (2010), Ghose et al. (2013), Agarwal et al. (2015)
Scheduling	Muralidhara et al. (2011), Kim (2010), Subramanian et al. (2016)
Technology	Meena et al. (2014), Li et al. (2017), Ebrahimi et al. (2010)

3.2.1 Mapping and scheduling

The memory scheduling algorithm should resolve memory contention by arbitrating memory access in such a way that competing threads progress at a relatively fast and even pace, resulting in high system throughput, fairness and desired performance.

Kim et al. (2010) proposed to divide threads into two separate clusters and employ various memory request scheduling policies in each cluster. The proposed thread cluster memory scheduling dynamically classifies threads with similar memory access pattern into either the latency-sensitive (memory-non-intensive) or the bandwidth-sensitive (memory-intensive) cluster. This work prioritized the latency-sensitive cluster over the bandwidth-sensitive cluster to improve system throughput. In addition, *niceness* is introduced and used as a metric that captures a thread's propensity to interfere with other threads. Ghose et al. (2013) argued that performing processor analysis of load instructions, and providing this analyzed information to memory schedulers, can increase the sophistication of memory decisions while maintaining a light-weight memory controller that can have a rapid response to scheduling actions. This is important as memory operating frequencies are rising and diversity of workloads running on heterogeneous computing systems is increasing. The HAMEX framework, proposed by

Moazzemi et al. (2016), can be used for design space exploration of memory access patterns in such systems.

Agarwal et al. (2015) proposed an approach to maximize cost and energy efficiency in heterogeneous systems integrating Central Processing Unit (CPU) and Graphics Processing Unit (GPU) which will increasingly use globally-addressable heterogeneous memory systems, making choices about memory page placement critical to performance. The proposed bandwidth-aware (BW-AWARE) placement, maximizes CPU throughput by balancing page placement across the memories based on the aggregate memory bandwidth available in the overall system.

Muralidhara et al. (2011) presented an alternative approach to reducing inter-application interference in the memory system: application-aware memory channel partitioning. The idea is to map the data of applications that are likely to severely interfere with each other to different memory channels. The data of light (memory non-intensive) and heavy (memory-intensive) applications are mapped to separate sections. A second improvement is to allocate the data of applications with low and high row-buffer locality. In addition, interference can be further reduced with a combination of memory channel partitioning and scheduling, which in this work is called integrated memory partitioning and scheduling where light applications are prioritized since these applications cause negligible interference with other applications and do not reduce the performance of competing applications unduly.

Kim (2010) proposes adaptive per-Thread Least Attainment Service memory scheduling, a novel memory scheduling technique that improves system throughput without the need for high level coordination between memory controllers. The key idea is to periodically order threads based on the service they have attained from the memory controllers so far, and prioritize in each period previously discriminated threads. The idea of favoring threads with least-attained-service is borrowed from the queuing theory literature.

Similarly, Subramanian et al. (2016) proposed a Blacklisting Memory Scheduler (BLISS), which achieves high system performance and fairness with low hardware cost and complexity. BLISS design is based on two observations. First, to mitigate memory interference, it is sufficient to separate applications into only two groups, one containing applications

that are vulnerable to interference and another containing applications that cause interference, instead of ranking individual applications with a total order. The vulnerable-to-interference group is prioritized over the interference-causing group. Second, this grouping can be efficiently performed by simply counting the number of consecutive requests served from each application. This low overhead ranking can reduce the overall interference in memory requests leading to higher performance in the system.

3.2.2 Hybrid memory technologies

There have been many break-throughs in memory technologies. Emerging nonvolatile memory technologies such as Magnetic Random-Access Memory (MRAM), Spin-Transfer Torque Random-Access Memory (STT-RAM), Ferroelectric Random-Access Memory (FeRAM), Phase-Change Memory (PCM), and Resistive Random-Access Memory (RRAM) can combine the speed of Static Random-Access Memory (SRAM), the density of Dynamic Random-Access Memory (DRAM), and the non-volatility of Flash memory (Meena et al., 2014). This opens up the opportunity to combine memory technologies in a system to gain benefit of a hybrid memory system. Although these technologies have disruptive potential, all of them have still to overcome some weaknesses in reliability, cost or manufacturability before their broad industrial deployment. But despite the ambiguity of the state of these technologies, researchers have explored their usage in system architectures and their impact on memory management policies.

For various hybrid memories Li et al. (2017) proposed a utility-based hybrid management scheme, which estimates the utility of migrating a page between different memory types, and uses the gathered information to enhance data placement. Ebrahimi et al. (2010) proposed an approach that provides fairness in the entire shared memory system, called Fairness via Source Throttling (FST). It estimates the unfairness in the entire shared memory system and if it is above a certain threshold set by system software or operating system, FST throttles cores causing unfairness by limiting the number of resource requests they can issue

and the frequency at which they do. Thus the resources are allocated more fairly.

3.3 Network

Networks-on-Chip (NoC) enable scalable design of on-chip interconnects for multi-core and many-core systems, to integrate cores and un-core components such as memory controller and cache blocks (Hemani et al., 2000; Dally and Towles, 2001). As the number of cores and on-chip components scale up, NoC efficiency became another crucial factor in determining overall system performance (Benini and De Micheli, 2002). A typical NoC infrastructure uses topologically arranged nodes such that each node has routers that are connected to cores and other routers. Communication among routers is enabled by switching mechanisms that forward *packets* - a unit of data - from a source to the specified destination router through communication channels. Several packets originating from different sources and traversing towards different destinations contend for shared network resources, which are to be resolved at every node. In view of these factors, network latency and thus performance are affected by (i) topology, (ii) routing and flow control.

3.3.1 Topology

Topology features the arrangement of nodes within a network representing the order of connection and the number of hops that packets traverse from source to destination (Fattah et al., 2012). Low diameter and high radix (number of communication ports of a router) topologies provide lower average hop count and thus higher performance (Balfour and Dally, 2014). Traditional topologies such as mesh and torus ease design complexity and are energy efficient, hence widely used in commercial processors (Wentzlaff et al., 2007; Howard et al., 2010). Advanced topologies include express cubes (Grot et al., 2009a), dragon-fly (Kim et al., 2008), slim-fly (Besta and Hoefler, 2014), flattened butterfly (Kim et al., 2007) and clos (Kao et al., 2011) that target high radix routers and/or low diameter networks. Optimized topologies are largely design

time approaches, while each such topologies feature smart routing and flow control mechanisms that are run-time adaptations. Most of the existing topologies follow the convention of finding the shortest possible path between a source and destination routers (Deo and Pang, 1984). Even advanced topologies use X-Y routing to retain simplicity and minimal deadlock free path finding (Grot et al., 2009a; Besta et al., 2018). However they are also adaptable for other routing algorithms, given the (high) radix of router architecture. Such run-time adaptations to improve NoC performance at run-time are discussed in the following sub-sections.

Table 3.4: Routing and flow control techniques

Level of adaptivity	Techniques
Oblivious	Feng and Shin (1997), Glass and Ni (1992), Seo et al. (2005), Deo and Pang (1984), Dally and Aoki (1993)
Local-awareness	Nilsson et al. (2003), Millberg et al. (2004), Dai et al. (2017), Balfour and Dally (2014), Rantala et al. (2008), Xu et al. (2016b), Chang et al. (2014), Al Faruque et al. (2012), Samman et al. (2013), Kim et al. (2007)
Global-awareness	Ma et al. (2011), Ascia et al. (2008), Zong et al. (2015), Ma et al. (2014), Mak et al. (2011), Dai et al. (2017), Besta and Hoefler (2014), Grot et al. (2009a), Tedesco et al. (2010), Catania et al. (2006), Carara and Moraes (2010)

3.3.2 Routing and flow control

Communication among different cores in an on-chip parallel processor is enabled through routing information in the form of packets Valiant and Brebner (1981). Routing algorithms provide the protocol for transporting packets across the network, considering underlying topology and router architecture (Singh et al., 2003). Routing strategies determine

end-to-end packet latency, which includes network congestion, shared-path contention and arbitration, effecting the performance (Wu et al., 2016). Routing strategies widely follow a two-step approach i.e., (i) path selection - to determine the next destination and (ii) flow control - to schedule the packets (Samman et al., 2013). Deterministic routing algorithms follow a fixed path decided at the source node itself, oblivious to traffic pattern and congestion, for the sake of simplicity (Li et al., 2006). Adaptive strategies on the other hand try to optimize performance, deciding the next immediate hop at every node, by monitoring network characteristics at run-time (Nilsson et al., 2003; Millberg et al., 2004; Al Faruque et al., 2012). Some adaptive routing techniques are based on heuristics and are congestion unaware for simplicity, while others rest on leveraging traffic patterns, application characteristics, congestion information and shared resource contention to decide on optimal route (Gratz et al., 2008; Feng et al., 2012). We present congestion oblivious, local aware and global aware adaptive strategies below, also summarized in Table 3.4.

Oblivious adaptivity

Monitoring different traffic and congestion patterns across the network and making an optimal choice of routing often turns into an NP-hard problem (Feng and Shin, 1997). Since adapting to such dynamic scenarios requires extensive router architecture design and includes routing complexity overhead (Feng and Shin, 1997), a class of algorithms use generalized adaptation (i) to avoid complexity and overhead, and (ii) to preserve general applicability and simplicity and (iii) provide worst-case throughput guarantees. Feng and Shin (1997) extensively tested different traffic patterns and routing strategies and concluded that no single routing strategy fits all traffic patterns and thus preferred to use a random free port direction towards the destination at every node. In a similar vein, Badr and Podar (1989) advocated to follow a zigzag path at every node, while Glass and Ni (1992) prefer to avoid turning as much as possible. Extending on the idea of minimizing number of turns, Sun et al. (2013) have proposed U2TURN routing, which identifies possible paths between a source and destination with a maximum of 2 turns

and distributes the traffic accordingly. Other oblivious yet adaptive routing algorithms include (Seo et al., 2005; Deo and Pang, 1984; Dally and Aoki, 1993). These routing strategies try to optimize a possible common case of traffic patterns with minimal router complexity, and are adaptive only to a fixed degree without congestion awareness.

Local-aware adaptivity

Early work in the NoC space has proposed deflection routing to avoid congested routers and spread the load more equally in the network based on local load signals (Nilsson et al., 2003; Millberg et al., 2004). Hu and Marculescu (2004) have proposed a smart router that switches between deterministic and adaptive routing, subject to on-chip traffic patterns. Other adaptive routing strategies used queue length based local congestion information such as buffer occupancy and/or bandwidth available, which reflects in channel, switch and port level congestion of a given node (Dai et al., 2017). This can in turn be used to avoid the most congested paths, or select the least congested path for high priority packets. Li et al. (2006) proposed dynamic X-Y routing (dyXY) by monitoring local congestion within the proximity of a node and re-direct packets to the least congested path. A similar approach is used by Rantala et al. (2008) to identify productive links - the least congested next nodes - and adapt traditional X-Y routing. Lysne et al. (2006) have proposed layered routing by virtually grouping channels into network layers. Each of the layers would be assigned a limited set of source and destination nodes adaptively to find shortest paths and for load balancing when needed. Balfour and Dally (2014) distinguished between packets as long and short and schedule the flows preemptively by reserving virtual channels ahead. In this case, they use adaptive X-Y routing to support the speculative flow control. Xu et al. (2016b) suggested to use also quantitative congestion information to represent the overall effect of congestion on performance with different paths, in order to choose the least congested route. Chang et al. (2014) predicted the least congested paths by monitoring the rate of change in buffer occupancy, considering both switch and channel level congestion. Al Faruque et al. (2012) used bandwidth available among all channels of

a node to choose the path with highest bandwidth and then dynamically re-assign buffers to the chosen route. Samman et al. (2013) followed a similar approach, considering bandwidth, congestion and congestion by monitoring buffer availability. Kim et al. (2007) used by-pass channels to shorten the hop distance to distant routers, yet retaining the principle of minimal path finding for the rest of the other routers. In contrast to choosing a suitable output channel, Wu et al. (2006) chose appropriate input packets to route through specific congested paths by monitoring upstream switch's contention.

All the above techniques rely on local information by monitoring the current node and the immediate neighboring nodes' congestion information alone, without global traffic pattern awareness.

Global-aware adaptivity

Using both local and global traffic patterns and congestion information allows for more efficient routing decisions to avoid potential congested paths and inter-application interference and provides workload consolidation (Ma et al., 2011). Ascia et al. (2008) identified neighbors-on-path for each possible direction to the destination node and chose the route with least possibility of congestion through to the destination node, having a regional congestion awareness. Zong et al. (2015) improved the same approach by considering congestion information traced over equal time intervals for fair estimates. Ma et al. (2011) and Ma et al. (2014) used both local and global congestion information to estimate total packet latency for possible paths and then chose the route with least delay. Both these approaches also provide workload isolation and avoid inter-application congestion. Similar to these region aware techniques, Chang et al. (2015b) divided the network into congested and un-congested clusters, and re-directed traffic towards un-congested clusters by monitoring channel history and current buffer vacancy. Mak et al. (2011) integrated a virtual dynamic programming network along with the original network, which holds the information on shortest possible paths from every node and thus to choose the route with minimum delay.

Adaptive routing combining path diversity and buffer vacancy is used by Chen et al. (2017) and Dai et al. (2017), also addressing fault tolerance. Feng et al. (2010) proposed a reinforcement learning strategy to monitor the network load situation for low latency routing decisions and to avoid faulty components. Besta and Hoefler (2014) presented globally adaptive routing relying on local information for minimalist path finding. They used different sets of virtual channels based on distance between source and destination routers. Besta et al. (2018) supported the possibilities of using both the above routing and flow control mechanisms, deciding between the optimal choice for a given instance. A similar look-ahead routing for mesh and torus topologies was used by Grot et al. (2009a), however restricting to only 1 virtual channel per port.

Along with the network information, leveraging application characteristics to make specific routing decisions based on communication patterns among any given nodes at IP level has been used by Tedesco et al. (2010); Catania et al. (2006); Carara and Moraes (2010). Most of the existing topologies follow the convention of finding the shortest possible path between a source and destination routers (Deo and Pang, 1984). Even advanced topologies use X-Y routing to retain simplicity and minimal deadlock free path finding (Grot et al., 2009a; Besta et al., 2018). However they are also adaptable for other routing algorithms, given the (high) radix of router architecture. While the aforementioned techniques try to adapt general routing techniques to specific architectures, Scott et al. (2006) proposed a two-step combination of non-deterministic up routing and deterministic down routing, based on the Clos architecture. Grot et al. (2011) followed a similar approach by designing specialized express channels for high priority packets, and used a re-route phase, if needed, for such packets. Bakhoda et al. (2010) proposed routers with limited connectivity, *half routers* to opportunistically cater for critical packets while reducing access to non-critical packets using half routers.

In addition to the techniques that dynamically adapt routing and flow control, source throttling i.e., controlling the rate of packet injection into the network at the source node itself has been proposed as another effective strategy. van den Brand et al. (2007) have presented congestion-controlled best effort (CCBE) architecture, to selectively control the

traffic load onto a router, subject to congestion monitored at run-time. Ogras and Marculescu (2008) have modeled traffic sources and routers to predict the amount of congestion potentially possible at each source node, and then control the packet injection rate at those sources accordingly in a pro-active manner. Chang et al. (2012) have further classified workloads into network and compute intensive and (network) bandwidth and latency sensitive to make adaptive source throttling decisions, to retain fairness and avoid any potential performance degradation.

3.4 Input/Output

With the advances in performance in multi/many-core systems, access to data becomes a prominent factor. With the surge in scientific kernels, deep neural networks and big data workloads that require massive amount of data, more applications are putting greater demands on end-to-end I/O performance (Pumma et al., 2017). In order to match speed of I/O with computational units many efforts both in hardware (i.e., Solid State Disk (SSD), hybrid storage) and software (i.e., placement, load balancing) have been proposed. Table 3.5 shows a classification of the techniques studied in the following sections.

Table 3.5: I/O management techniques

I/O management	Techniques
Hybrid storage	Gupta et al. (2009), Tai et al. (2017), Yang et al. (2016), Chen et al. (2011), Pritchett and Thottethodi (2010)
Placement	Neuwirth et al. (2016), Neuwirth et al. (2017), Wang et al. (2014), Elyasi et al. (2017)
Scheduling	Ausavarungnirun et al. (2012), Chang et al. (2015a), Tavakkol et al. (2018)

3.4.1 Hybrid storage

In many emerging heterogeneous systems, Nand-based flash memory is used as an intermediate level before disk to improve I/O performance

and reduce power. In many cases traditional cache algorithms such as LRU are used to manage these systems. Gupta et al. (2009) proposed a Demand based Flash Translation Layer (DFTL) which caches page-level address mappings in the flash. This would lead to reduced garbage collection overhead and improved performance. Tai et al. (2017) proposed a virtual flash resource manager which uses use bins with large spatial size for units of migration in order to update the data between flash and disk in a more relaxed fashion. This leads to a significant saving in memory space and reduction in I/O traffic. A similar global SSD resource management scheme is proposed by Yang et al. (2016). It splits the SSD into long-term and short-term zones and categorizes the content of the SSD as a second cache for disk into these zones. Pritchett and Thottethodi (2010) proposed SieveStore which uses of solid-state memory to filter access to storage ensembles. The filtering is used on highly-skewed popularity distribution of disk blocks and popular block-sets. This enables disk-caching to reduce I/O in an efficient manner. Similarly, Chen et al. (2011) proposed a hybrid storage scheme by monitoring I/O access patterns at run-time to identify blocks with long latency or semantically critical. These blocks will be stored in SSD for later access. In addition, this speeds up the I/O functioning as write-back buffer.

3.4.2 Load balancing and placement

Dynamic load balancing can be used in order to mitigate resource contention and improve the overall system performance. Neuwirth et al. (2016) tackled the imbalanced use of I/O resources to improve the performance using a topology-aware approach called Balanced Placement I/O (BPIO) to mitigate resource contention benefiting from a middle-ware. The same authors further improved their work proposing the TAPP-IO framework with a new placement strategy to support file-per-process I/O and single shared file I/O as well as intercepting file creation calls during run-time to balance overall workload on available storage Neuwirth et al. (2017). Wang et al. (2014) proposed a topology-aware strategy to enhance the performance of I/O on a per-application

basis using load balancing across the resources. Elyasi et al. (2017) leveraged the slack between sub-requests in order to improve response times of SSDs. Specifically, the paper presents the design and implementation of a slack-enabled re-ordering scheduler for sub-requests issued to each flash chip.

3.4.3 Scheduling

Interference in SSD-based shared storage systems has been a major issue. Chang et al. (2015a) addressed two types of interference, namely, queuing delay (QD) interference and garbage collection (GC) interference. They used a credit-based I/O scheduler designed to address QD interference and the flash translation layer designed to address GC interference. Staged memory scheduler was proposed by Ausavarungnirun et al. (2012) as a three-stage memory controller for heterogeneous systems with integrated CPUs and GPUs. This work groups requests to I/O based on row-buffer locality and focuses on inter-application request scheduling. Tavakkol et al. (2018) proposed a flash level interference-aware scheduler as an I/O request scheduling mechanism. Its goal is to provide fairness among requests using a three stage scheduling algorithm mitigating issues causing interference such as differences in request access patterns, the ratio of reads to writes, garbage collection.

3.5 Summary

We presented run-time techniques that improve system performance by allocating and provisioning compute, memory, network and I/O resources. Most of these techniques rely on monitoring application, task and thread level properties and leverage them at run-time to provide higher or sufficient amount of resources for higher performance. While some techniques require application level expression and translation, others entirely rely on run-time information. Combining monitoring techniques across different resources for co-optimization can provide flexible and modular control on performance enhancement.

The version of record is available at: http://dx.doi.org/10.1561/1000000055

4

Power, energy, and thermal management

Computer systems design is confronted with the need for high performance under limited power consumption. Diversity in type and increasing complexity of applications demand for higher computation power. To deliver this higher performance, designers have to consider the reasonable autonomy in battery-powered systems, operation cost of servers as well as reduction in the environmental impacts of *power consumption*. With the advances in multi/many-core system *energy efficiency* and controllable *temperature* became a vital metric in design of the emerging computer systems. Efficient resource utilization with consideration of power usage, energy efficiency and temperature can open the way for further optimization and achieving higher performance. Figure 4.1 represents a classification on resource management methods used to handle power consumption, energy efficiency and temperature control. *Power management* methods minimize or cap the momentary power consumption at any given time instant. *Energy Efficiency* focused approaches reduce the accumulated energy during the system live time. *Thermal management* methods resolve thermal issues and contain the system in a safe temperature while avoiding hot spots. In the rest of this section we survey some of the efforts in this area.

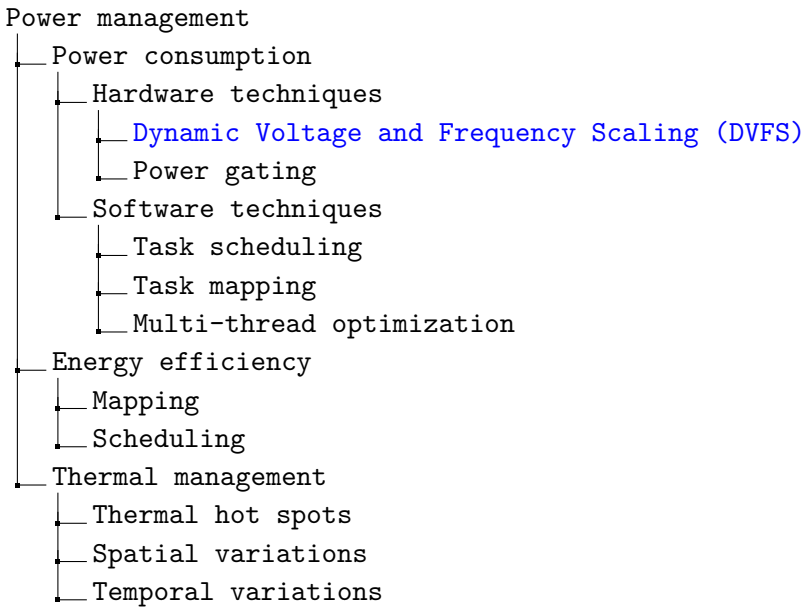

Figure 4.1: Techniques for power, energy and thermal management

4.1 Dynamic management objectives

In this section, we focus on three important in many ways correlated objectives in dynamic management of a computing system:

- Power
- Energy
- Thermal

While examining these three metrics used as objective of management techniques, we should bear in mind the cinch correlation between them. Power is consumed in every part of the circuit either as static power or dynamic power consumed during the active cycles of all resources such described in Section 2.1. This power consumption over time will lead to energy consumed or in other terms, power is energy per unit of time. The power consumed in each resource can generate heat which can increase the temperature in the whole or part of the

computing system. So why not only focus on reducing power which might lead to lower energy consumption and temperature? we should realize that power consumption is not the only deciding factor in energy consumption or inducing thermal emergencies. As we will see in the rest of this section, other factors such as operating voltage or frequency, performance, hotspots can also play a role in the dynamics of the system. Thus, there is a need for techniques centered toward optimizing each of these metrics (power, energy and thermal).

Prior to exploring techniques used in Dynamic Power Management (DPM) (Section 4.2), run-time energy efficient management (Section 4.3) and Dynamic Thermal Management (DTM) (Section 4.4), we take a look at measurement methods used in these techniques. This can show progress of sensors embedded in emerging devices and progress of models used in estimating power or temperature of a system. Subsequently, resources that can be managed using through run-time management techniques are identified and some of the techniques used to manage each resource is discussed. This can give us a better understanding of the scope of actuations in each resource that can later on be used in management of the whole system.

4.1.1 Managed components

As we have seen in the previous chapter on performance management techniques, the approaches proposed in the literature are fairly different for the computation, communication and memory subsystems. Hence, we start by reviewing efforts towards run-time power management of different computer system components. Management techniques in each individual component can later on be used to control the holistic system. Next, we survey the literature regarding the methods for dynamically managing systems for energy efficiency and reduced power consumption.

Computation: There is an extensive literature on how to improve the energy efficiency of computation. Power reduction can happen by means of transistor and gate design, efficient design of the Instruction Set Architecture (ISA) or control of voltage and frequency levels. Benini et al. (1999) primarily started devising policies to optimize power consumption

of computing devices. Lorch and Smith (2004) proposed a voltage scaling method for estimating the task work distribution and power estimation based on workloads dynamic behavior. You and Chung (2014) proposed a power management scheme to transition to standby and power-off states to reduce power consumption in embedded processors. Networks-on-Chip (NoC) platforms can be portioned to islands in order to avoid hot-spots and deliver smoother power management. This has been further discussed by David et al. (2011) using an Intel NoC chip. Run-time software power management methods have been utilized to benefit from run-time information during software execution. Teodorescu and Torrellas (2008) proposed an application scheduling technique for power management for chip multi-processors. Methods proposed by Lorch and Smith (2004) estimate the task work distribution and approximate the optimal continuous schedule by modifying the voltage scaling algorithm to minimize energy use without affecting perceived performance.

Communication: While many early works focused on designing low power and energy efficient computing units, over the years with the emergence of multi-core Systems-on-Chips (SoCs) and advances in on-chip interconnection architectures the share of power consumption in the communication increased. This motivated many system designers to develop energy efficient interconnects, e.g. Shang et al. (2003); Soteriou and Peh (2007). The work presented by Chou et al. (2008) addresses the energy-aware incremental mapping problem for NoCs with multiple voltage levels and proposes an efficient approach using the near convex region selection technique. Passos et al. (2006) took a look at the systematic design of communication intense devices while Chen et al. (2013) focuses on design of NoCs have to deliver low latency, high bandwidth, at low power. Authors in Chen and Joshi (2013) present a silicon-photonic multi-bus NoCs architecture between private L1 caches and distributed L2 cache banks which uses weighted time-division multiplexing. In order to provide an energy/performance aware mapping in NoCs architecture, Hu and Marculescu (2003) proposed a branch-and-bound algorithm which maps the IPs onto a generic regular NoC while guaranteeing to satisfy the specified constraints through bandwidth reservation. In addition, many efforts have been undertaken on NoC

DPM (Banerjee et al., 2009). A recent review of these works is provided by Guang et al. (2009).

Memory: There have been many advances in memory technology in the recent years. Some of these efforts increase the performance and response rate of these components. On the other hand, some methods like those proposed by Ha et al. (2017) and Ahn et al. (2014) benefit from emerging technologies to make memory devices more energy efficient. A range of studies such as (Nakai et al., 2005; Gurumurthi et al., 2003) aimed at the power consumption of memory devices by reducing the components voltage, speed or frequency based on the application demand at run-time.

4.2 Dynamic power management techniques

DPM has been proposed decades ago. Designers used DPM in the 90s (Chung et al., 1999) with the available run-time configurations such as scaling the supply voltage to lower the power consumption (Nielsen et al., 1994). Most opportunities to reduce power consumption comes from non-optimal configurations in hardware and software components.

Before diving into the analysis of efforts done in this domain, it is important to make a distinction between power-aware and low-power systems. The focus of low-power systems design is to minimize power. On the other hand, for a power-aware system meeting power and energy goals is a significant design consideration and in which the system modifies its behavior based on current power/energy availability. Some power-aware design goals may even increase power or energy consumption. Consider the case of a design for decreasing peak power in a processor: one method to attain this goal would be to use schemes that would intentionally delay the issue of some instructions to smooth the instruction issue distribution and, thus, decrease the peak consumed power. However, delaying some instructions could lead to the application being finished later than it otherwise would, therefore increasing the energy consumption. Thus, this scheme would be a power-aware, but not a low-power, design (Unsal and Koren, 2003). In the following we mainly focus on DPM for power-aware systems summarized in Table 4.1.

Table 4.1: DPM techniques

Power management	Techniques
Hardware	Kirovski and Potkonjak (1997), Ishihara and Yasuura (1998), Chen and Marculescu (2015a), Lee and Sakurai (2000), Krishna and Lee (2000), Luo and Jha (2001), Liu et al. (2001), Cheng et al. (1997), Qiu and Pedram (1999), Isci et al. (2006a), Azevedo et al. (2002), Pillai and Shin (2001), Shahosseini et al. (2017), Bogdan et al. (2013), Rahmani et al. (2015), Ogras et al. (2009), Ogras et al. (2008), Lackey et al. (2002), Kim et al. (2017), Cochran et al. (2011), Herbert and Marculescu (2007), Das et al. (2015)
Software	Li and Wolf (1997), Shin and Choi (1999), Huang et al. (2009), Kulkarni et al. (1998), Winter et al. (2010)

4.2.1 Hardware methods

One of the main techniques used in DPM is to use the inherent properties of hardware components to reduce power consumption. This can be achieved by reducing the voltage, frequency or even shutting down the processing units.

DVFS and power gating

DVFS is a common method in DPM. The computing unit (or communication media) must be augmented with hardware blocks that allow for changing the supply voltage dynamically. This is common in recent processors. Although, Kirovski and Potkonjak (1997) proved that task allocation and scheduling optimization problems using DVFS are NP-complete, heuristics to tune voltage and frequency are widespread because they save substantial power. However, reducing the frequency causes a slowdown in the execution of programs with potentially detrimental effects. Thus, DVFS heuristics usually trade off power savings against delay. One of the earliest works on DVFS has been presented by Yao et al. (1995). Their method to manage frequency and voltage is

called the *Average Rate* heuristic which sets the speed of the processor to the sum of average rate requirements of tasks in the frame. Theorems proposed for power-delay optimization in Ishihara and Yasuura (1998) utilize an Integer Linear Programming (ILP) problem to minimize energy consumption under an execution time constraint. Lee and Sakurai (2000) partition each task into time slots which enables DPM to change voltage at specific intervals. A hybrid method is used by Krishna and Lee (2000) using two algorithms consisting of an online phase, in which voltage settings are selected to reduce energy consumption assuming that tasks complete in their Worst-Case Execution Time (WCET). Considering that many tasks finish well before their WCET, this method uses an online phase which adjusts the voltage settings on-the-fly to reclaim any resources released by such tasks. A mechanism proposed by Luo and Jha (2001) performs variable-voltage scheduling via efficient slack time re-allocation, which helps reducing the average discharge power consumption as well as smooths the discharge power profile.

Liu et al. (2001) focused on power-aware scheduling in mission critical embedded systems. Their approach is incremental by solving one type of constraint at a time. First, a time-valid schedule is constructed from a constraint graph of the task. Next, this schedule is validated against the maximum power constraint to remove power spikes, and finally it is compared against the minimum power constraint and tasks are reordered to reduce power gaps and power utilization. Cheng et al. (1997) employed a fine-grained offline scheduling approach that saves power by combining multiple instructions into one complex instruction with lower power consumption, or by using low-power versions of instructions while considering task deadlines. Qiu and Pedram (1999) proposed a DPM method using Markov decision processes. The problem of DPM in such a system is formulated as a policy optimization problem and solved using an efficient "policy iteration" algorithm. Idea of managing power of voltage islands in SoC has been well studied Lackey et al. (2002). In this domain, Herbert and Marculescu (2007) shows the trade-offs involved in the choice of both DVFS control scheme and method by which the processor is partitioned into voltage/frequency islands and presents potential in using DVFS for dynamic power management in CMPs. Cochran et al. (2011) proposes a control technique to make DVFS and

thread packing control decisions in order to maximize performance within a power budget using a multinominal logistic regression classifier. Das et al. (2015) proposes an approach for DVFS in smartphones, which uses reinforcement learning to explore the trade-off between power saving opportunities using DVFS and dynamic core selection and application's performance at run-time. Architecture-independent imitation learning methodology is proposed in Kim et al. (2017) for DVFI control in many-core systems by using controllers that leverage the structural relationships between VFIs.

Isci et al. (2006a) monitored the run-time application in order to optimize power consumption by setting per core DVFS using a global system manager. Azevedo et al. (2002) employed an intra-task dynamic voltage scaling technique under compiler control using program checkpoints. Checkpoints are generated at compile time and indicate places in the code where the processor speed and voltage should be recalculated. These checkpoints are used at run-time to recalculate voltage and frequency settings. To reduce the overhead of dynamic scaling Pillai and Shin (2001) introduced DVFS in operating systems. Since then, the majority of operating systems have simple settings to benefit from DVFS even in embedded devices. A feedback control was used by Shahosseini et al. (2017) to manage the power consumption using DVFS knobs.

The trend towards multi/many-core platforms requires techniques that can formally guarantee power management of the system given a power budget. Bogdan et al. (2013) proposed a paradigm shift from power optimization based on linear models to control approaches based on fractal-state equations. Rahmani et al. (2015) developed a multi-objective DPM method that simultaneously considers limits on the total power consumption, dynamic behaviour of workloads, processing elements utilization, per-core power consumption, and the load on the NoC. This work uses fine-grained voltage and frequency scaling, including near-threshold operation, and per-core power gating to optimize the performance and power consumption. In addition, a disturbance rejecter is designed that proactively slows down running applications when a new application commences execution, to prevent sharp power budget

violations. Tilli et al. (2015) proposes a low overhead hierarchical model-predictive controller (MPC) for managing thermally safe sprinting with predictable resprinting rate, which ensures the correct execution of mixed-criticality tasks. A methodology for multi-clock/voltage domains is proposed by Ogras et al. (2009, 2008) by adaptively partitioning and voltage assignment using state-space feedback control strategy to dynamically scale the operating voltage and frequency around the static values and load balance the network traffic in the presence of workload and parameter variations. Chen and Marculescu (2015a) presents an On-line Distributed Reinforcement Learning (OD-RL) based DVFS control algorithm for many-core system performance improvement under power constraints that uses per-core reinforcement learning method for frequency management and a control theoretic method for global power budget allocation. This method takes advantage of fast response of the per-core reinforcement learning while making sure the global power consumption remains under the budget using Maximize-the-Max method. Muck et al. (2018) propose Multiple-Input Mutiple-Output (MIMO) for controlling various actuators in Heterogeneous Multi-core Processors (HMP) in order to control both power consumption and overall system performance.

4.2.2 Software methods

Software power management techniques can be categorized into two closely related areas of research. First, different studies explored the properties of workload variations and developed methods to identify and follow different execution behavior, commonly referred to as "phase analysis". Second, a large complementary set of research studied dynamic, on-the-fly system management techniques that can adaptively respond to these differences in application behavior (e.g. Isci et al. (2006b)).

Task scheduling and thread optimization

DPM mechanisms in real-time systems become more complex as the system has to meet certain deadlines while keeping the power below a certain budget. A synthesis algorithm by Li and Wolf (1997) uses a software-based cache partitioning and reservation technique to guarantee

cache hits for some tasks and therefore improve task schedulability. The scheduling algorithm used in this work is Earliest Deadline First (EDF). In EDF, the task with the earliest deadline has the highest priority. The method proposed by Shin and Choi (1999) and implemented in a kernel module, yields power reduction by exploiting slack times, both those inherent in the system schedule and those arising from variations of execution times. As an example, Huang et al. (2009) propose adaptive DPM for hard real-time systems. In their work, based on real-time calculus, event arrivals and resource services are modeled by arrival curves and service curves in the interval domain, respectively, and an online algorithm to adaptively control the power mode of the device is proposed, that postpones the processing of arrival events as long as possible. In a similar spirit, many cache aware methods have been proposed, such as (Kulkarni et al., 1998), to benefit from software optimizations to reduce power consumed by cache and memory subsystems. Further analysis of system-level power-aware design techniques is presented by Unsal and Koren (2003) who cover techniques ranging from the circuit and device level, to the architectural, compiler, operating system, and networking layers. In the case of many-core systems, scalability of scheduling algorithm becomes an important factor which Winter et al. (2010) analyzes some of these algorithms in terms The computational complexities of thread scheduling and global power management techniques.

4.3 Run-time energy efficient managers

Although for many computer systems, reducing power will lead to reduced energy consumption but it does not necessarily mean a resource management mechanism with the objective of power management will deploy same policies as a resource management method with energy efficiency goal. Energy directly relates to both *power* and *performance*. Final goal for energy efficient management methods is to find the optimal spot in power consumption while delivering the required performance over time. To this end, we take a closer look at energy efficient managers in this section. Computer systems are designed to deliver peak performance, but are often idle or used to perform tasks that do not require such performance.

Energy efficiency has become a major concern while dealing with high performance computing systems (Ge et al., 2010; Hsu and Feng, 2005). Architectural optimization to achieve a high performance with minimal power consumption has been a common practice for emerging applications. Qu et al. (1999) proposed a new pipelining mechanism with selectable voltage for each pipeline stage to minimize energy consumption. Evaluation of energy efficiency of a system can be related to both power consumption and performance of the running application. A common metric for the evaluation of energy efficiency is Energy Per Instruction (EPI), in Watt/MIPS or Joule/Instruction. Other metrics such as Energy Delay Product (EDP), which was initially proposed by Horowitz et al. (1994), and ED2P are used also in latency performance architectures as they assign a weight to the amount of time needed for an instruction to be processed (Attia et al., 2017).

Manzak and Chakrabarti (2000) proposed task scheduling algorithms that minimize energy or minimize power for the case when the tasks have various arrival times, deadline times, execution times and switching activities. The relation between the operating voltages for the minimum energy (power) assignment is determined theoretically and a polynomial time scheduling algorithm that uses this relation is developed to minimize energy consumption. The authors improved this method (Manzak and Chakrabarti, 2001) by first applying the existing task scheduling algorithms (Manzak and Chakrabarti, 2000) to obtain a feasible schedule and then distribute the available slack using an iterative algorithm that satisfies the theoretically obtained relation for minimum energy. Shafique et al. (2013) considered self-adaptive many-core systems to reduce the energy-delay^2 product. To avoid frequent allocation and de-allocation, this work enables applications to temporarily reserve their resources and to perform local power management decisions.

Diversity and complexity in High Performance Computing (HPC) systems require specific solutions for DPM, as presented by Rusu et al. (2006) for server clusters and by Beloglazov and Buyya (2010) for cloud systems. Hughes et al. (2001) targeted multimedia applications by using both architectural adaptation and dynamic voltage scaling. Similarly,

Unsal et al. (2001) proposed two complementary media-sensitive energy-saving techniques that leverage static information. First, a compiler-controlled data remapping scheme directs scalar accesses to a small scratchpad Static Random-Access Memory (SRAM) area. Second, a media-sensitive software-controlled caching framework eliminates cache tags. The same authors further improve their own results by showing that media applications are mapped more efficiently when scalar memory accesses are redirected to a mini-cache (Unsal et al., 2002). Using the Combined Static/Dynamic scheduler in the operating system as basis, Ma and Shin (2000) developed an Energy-Adaptive scheduler with an energy-aware scheduling algorithm that executes tasks to achieve effective use of limited energy by favoring low-energy and critical tasks.

Baynes et al. (2003) evaluated energy consumption in various Real-Time Operating Systems (RTOS) including preemptive systems and cooperative systems. Acquaviva et al. (2001) proposed real-time dynamic voltage scaling that modify the operating system's real-time scheduler and task management service to provide significant energy savings while maintaining real-time deadline guarantees. Mishra et al. (2015) proposes a probabilistic graphical model-based learning system to provide accurate online estimates of an application's power and performance in order to optimize energy efficiency of the system.

Table 4.2: Run-time energy efficient management techniques

Energy management	Techniques
Mapping	Baynes et al. (2003), Acquaviva et al. (2001), Unsal et al. (2001), Unsal et al. (2002), Manzak and Chakrabarti (2000), Manzak and Chakrabarti (2001), Ma and Shin (2000), Shafique et al. (2013), Attia et al. (2017), Mishra et al. (2015)
Scheduling	Hughes et al. (2001), Ge et al. (2010), Hsu and Feng (2005), Rusu et al. (2006), Beloglazov and Buyya (2010)

4.4 Dynamic thermal-aware management methods

Delivering high performance in computation does not only come with the cost of power consumption. Often circuits that perform at their peak suffer from thermal issues such as overheating or faults due to thermal emergencies. Many dynamic management methods try to avoid such conditions. On the other hand, minimizing power does not necessarily avoid thermal issues. In many cases, concentrated power usage in a small part of the electronic circuit can cause high a temperature on that spot leading to thermal failures although the power usage of the whole system might not be high.

Thermal induced problems can appear in various forms. *Thermal hot spots* accelerate failure mechanisms. Failure cases increase exponentially with temperature (Meterelliyoz et al., 2005). Hot spots also cause performance loss and lead to higher leakage of power (Vassighi and Sachdev, 2006). *Spatial variations* can cause clock skew resulting in transient or intermittent delay faults. Finally, *temporal variations* induce thermal cycling (Coskun et al., 2007) that can cause violation in completion of the cycle that is large enough to cool the component.

Thermal management techniques try to control the chip temperature. Many of the power management methods in this chapter are concerned with temperature while focusing primarily on overall power consumption. The goal of DTM is to address thermal hotspots or reduce spatial and temporal temperature variations. Frequency scaling, DVFS, Decode Throttling, Speculation control and cache toggling are some of the DTM techniques described by Brooks and Martonosi (2001). Donald and Martonosi (2005) use temperature aware scheduling for multi-threaded processors by taking advantage of Simultaneous Multi-Threading (SMT) unique flexibility of having multiple threads to adaptively counteract and prevent hot spots by selectively managing the execution of available threads. Liao et al. (2005) describe how smart performance and power modeling can reduce the power leakage and limit temperature increase which can eventually improve performance and power consumption. Jung and Pedram (2006) propose a framework in which thermal states are controlled by stochastic processes, i.e., partially observable semi-Markov decision processes. By using multi-objective design optimization

methods such as collaborative optimization operating temperature is reduced.

Thermal management is an important issue in embedded systems due to limited area and cooling methods. Utilizing on-line monitoring, Christoforakis et al. (2015) propose a response mechanism using a distributed power management algorithm for Field Programmable Gate Array (FPGA) to evenly reduce and normalize power transients and achieve a power-and thermal-aware coherent system. Ayoub et al. (2010) presented a DTM mechanism for memory subsystems which intelligently allocates workload pages to few memory units and powers down the rest of the memory.

Thermal management became a prominent challenge in fighting the expanding dark, nonactive, silicon areas on chip. Hajimiri et al. (2013) proposed thermal-aware computation using proactive memory-based computing to reduce the peak temperature of applications. This technique proactively transfers the instructions with frequent operand pairs to memory. In Kanduri et al. (2015) a dark silicon aware run-time mapping method was proposed that activates and deactivates cores as needed in order to evenly distribute power density across the chip. The same authors expand this idea Kanduri et al. (2018a) by providing enough thermal headroom for boosting the frequency of active cores upon performance surges. Lower operating temperatures from patterning also allows sustaining the boosting for longer periods, improving the performance further.

Lee et al. (2015) and Lo et al. (2016) considers 3D stacking architectures and the thermal limitations for such chips. The method proposed by Lee et al. (2015) combines dynamic cache management such as resource allocation, way-based power gating, and data migration with dynamic voltage and frequency scaling of processing cores in a temperature- and energy-aware manner. Lo et al. (2016) propose a thermal-aware dynamic operating system page allocation using future access pattern to find a best performance-oriented setting of the above factors. Also, An analytic model has been proposed to estimate the system performance considering the memory interference, the bandwidth variation, and the throttling impact.

Thermal aware communication systems can reduce the possibility of thermal emergencies in the system (Wolf et al., 2016). Chou et al. (2017) proposed a thermal aware method for dynamic buffer allocation for NoC systems. Zhang et al. (2014) introduces a job allocation technique that minimizes the temperature gradients among the ring filters to improve the application performance in silicon-photonic NoCs. In the context of local temperature hotspots in a load-imbalanced network, Murray et al. (2014) demonstrated power and thermal profiles improvement by utilizing congestion-avoidance routing with network-level DVFS in a mm-wave small-world wireless NoC.

Finally, we can observe a trend in industry towards utilizing machine learning Kaggle Inc. (2017); Parloff (2016) and neural network in power and energy management. This has been enabled by easy access to configuration knobs such as power gating, thread scheduling and frequency scaling. This ease of access to configuration knobs has been granted due to effectiveness of many of the approaches we discussed in this section. Therefore, the current trends will open the path for more complex management methods. Possible open challenges in power, energy or thermal management might be hierarchical management schemes, adaptive control or hybrid approaches comprised of machine learning, heuristics and control theoretic methods.

Table 4.3: Thermal management techniques

Thermal management	Techniques
Topology-aware mapping	Meterelliyoz et al. (2005), Ayoub et al. (2010), Vassighi and Sachdev (2006), Donald and Martonosi (2005), Coskun et al. (2007), Liao et al. (2005), Jung and Pedram (2006), Zhang et al. (2014), Kanduri et al. (2015), Chou et al. (2017), Kanduri et al. (2018a)
Resource allocation	Brooks and Martonosi (2001), Skadron et al. (2003), Hajimiri et al. (2013), Christoforakis et al. (2015), Lee et al. (2015), Lo et al. (2016), Wolf et al. (2016), Murray et al. (2014)

4.5 Summary

We presented run-time techniques that manage the system overall power consumption, helps energy efficiency and avoid thermal emergencies. Most of these techniques rely on monitoring application, partitioning and mapping to get the best power consumption out of shared resources. While some techniques heavily rely on hardware sensing, monitoring and control, others use run-time workload profiling and software approaches to manage the resources dynamically.

The version of record is available at: http://dx.doi.org/10.1561/1000000055

5

Reliability

In the last decade, as discussed in Semiconductor Industry Association et al. (2011) reliability has become a major issue in digital circuits. The aggressive scaling to nanoscale CMOS structures has caused a variety of reliability threats such as aging and wear-out acceleration due to the increased power densities and consequent thermal stress, higher susceptibility to soft errors not only in harsh environments but also at ground level, device variability leading to timing errors and other effects, etc.

This issue has been even more exacerbated by the pervasiveness of computing systems in nowadays life. In the past decade, reliability was a relevant driver only for the design of digital appliances for mission-critical applications, such as satellites, medical appliances, nuclear facilities or transportation systems. In most of these scenarios, failures would have caused damages to the humans or to the environment. Nowadays, computing systems are intensively employed to control and support any aspect of our life spanning from embedded appliances enabling smart buildings to High Performance Computing (HPC) systems used for supporting financial and decision processes. Thus, apart

the serious human and environmental risk, system failure may cause also considerable financial and monetary loss.

Given these motivations, reliability has become in general a main driver for digital systems design. In particular, when considering the focus of this paper on on-chip resource management, reliability needs to be considered at the same level of relevance of the other already discussed drivers (performance, power, energy, ...), thus leading to a reliability-aware dynamic co-optimization.

Reliability-aware run-time resource management approaches can be mainly classified according to the specific type of threat they aim at addressing and the corresponding reliability attribute introduced into the system. In particular, we may partition the approaches proposed in the literature as follows:

- **Lifetime management.** These approaches aim at preventing aging and wear-out threat by performing a more aging-aware distribution of the workload and tuning of the architectural parameters in order to prolong the lifetime of the system.
- **Soft error susceptibility management.** These approaches aim at distributing the workload and tuning the architectural parameters in order to trade-off the fault occurrence probability also in relationship with process variability effects and the offered performance level.
- **Online fault management.** These approaches aim at dynamically detecting and managing the occurrence of faults by tuning at run-time available resources, the running workload and the possibly available fault management mechanism.

The following sections will review the literature for each one of these aspects.

5.1 Lifetime management

Accelerated device aging causing timing errors or premature permanent failures is one of the most challenging threats for nowadays computing devices (Semiconductor Industry Association et al., 2011). Integrated

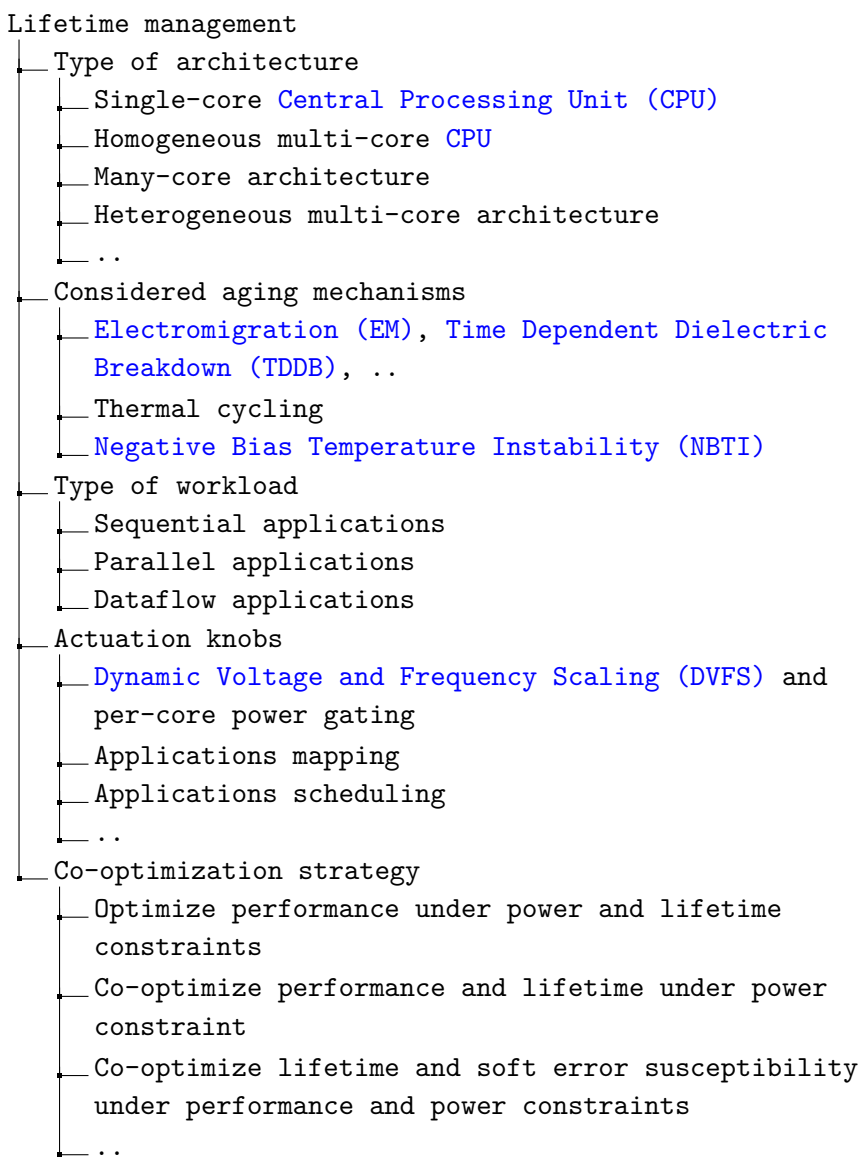

Figure 5.1: Criteria considered for classifying lifetime management approaches.

circuits suffer of several types of aging mechanisms, such as TDDB, NBTI, EM, hot carrier injection (HCI) or thermal cycling (JEDEC Solid State Tech. Ass., 2010). The main causes are jointly the sub-micro CMOS technologies leading to higher power densities and the intensive utilization of computing devices thus leading to a considerable thermal stress. Indeed, Karl et al. (2008) have shown that failure mechanisms have an exponential relationship with the temperature and an increase of 10-15°C may halve the expected lifetime.

The countermeasures adopted by the state-of-the-art approaches to prolong the system lifetime is to use and manage the available processing resources in a smarter aging-aware way for executing the workload. This strategy is particularly effective for multi-core or many-core platforms (both homogeneous and heterogeneous ones) thanks to the availability of a large set of "programmable" processing resources, representing a sort of redundancy, that can be dynamically tuned and selected for the execution of the various applications composing the workload. Furthermore, we should consider the fact that this kind of architecture often experiences a dynamic workload, where applications presenting different peculiarities and performance requirements enter and leave the system with unknown load characteristics. As a consequence, run-time management is even more critical because it can offer adaptivity to the system to be able to react to evolving running conditions w.r.t. reliability issues.

The commonly-used reliability-aware resource management approach is based on a feedback loop for monitoring the aging status of the various components within the system and subsequently taking decisions on the utilization of the resources. Ideally, these approaches require aging sensors (e.g. Blome et al. (2007); Ceratti et al. (2012)) to be integrated in the device. However, even if widely studied by the research community they are not commonly available in commercial platforms; moreover, they are mainly suitable only for the monitoring of timing errors. For this reason, per-core temperature sensors, that are generally provided in modern devices, are used as input of a reliability emulation monitors implementing standard stochastic reliability models described in JEDEC Solid State Tech. Ass. (2010). Such models can be used by these approaches to compute the *lifetime reliability* value $R_{lifetime}(t)$,

as the probability of the system not to permanently fail due to aging at time t, and eventually the Mean Time To Failure (MTTF), estimating the average lifetime of the class of devices.

It is worth mentioning the fact that the Dynamic Thermal Management (DTM) approaches discussed in Section 4.4 do not suffice in managing lifetime reliability. In fact, when considering the characteristics of several aging mechanisms, lifetime reliability is generally characterized by a stochastic exponential model of the temperature, considering both temperature levels and the time spent at those levels. Thus, the straightforward temperature control avoiding thermal peaks and hotspots or minimizing or smoothing the average thermal behavior among the cores is not able to capture and control the "cumulative" degradation behavior of the aging phenomena, thus leading to non-optimal solutions, as demonstrated in the literature (in particular by Srinivasan et al. (2004); Song et al. (2015)).

Starting from the first work by Srinivasan et al. (2004) proposing the Dynamic Reliability Management (DRM) in 2004, in the past fifteen years a plethora of approaches have been proposed focusing on many different aspects. Thus, this literature can be classified based on various criteria presented in Figure 5.1 and discussed in the following paragraphs. Then, past approaches will be reviewed.

Type of architecture

The first DRM approach by Srinivasan et al. (2004) focused in a single general purpose processor. After that, following also the architectural progresses in the subsequent years, different types of platforms have been considered spanning from the homogeneous multi-core architectures (e.g. Coskun et al. (2009); Das et al. (2014); Ma and Wang (2012)), where processing units are connected on a single bus and with a share memory, to the many-core architecture based on a Networks-on-Chip (NoC) (e.g. Ma et al. (2017a); Sun et al. (2014); Kim et al. (2016)). Recently, heterogeneous architectures (e.g. Baldassari et al. (2017); Chen et al. (2014b); Lee et al. (2018)), integrating asymmetric multi-cores, Graphics Processing Units (GPUs) or custom accelerators, have been also addressed in lifetime management.

It is worth mentioning that lifetime management is generally performed by considering the single processing units as the basic architectural element, thus without having the possibility to distinguish computation, communication and memory resources as previously done in the survey. This is due to the fact that temperature sensors on which the reliability emulation relies are integrated at core level; thus lifetime reliability is computed at the granularity of the single processing unit. Indeed, the processing core is the component in the system which is more susceptible to thermal hotspots and therefore requiring an aging monitoring. Nevertheless, it is also worth mentioning that in some types of distributed architectures, such as NoC-based many-core ones, each single processing unit integrates its own memory system and network interface in a single tile.

Considered aging mechanisms

To be effective the lifetime management strategy should integrate a certain level of knowledge about the considered aging mechanisms in order to limit the factors of stress. The effects on the lifetime reliability of most of these phenomena (e.g. EM, TDDB and NBTI) can be modeled for each single architectural core as an exponential function of the temperature level. Just as to mention, the MTTF of a single core working at a constant temperature T can be estimated by means of the Arrhenius equation as (JEDEC Solid State Tech. Ass., 2010; Xiang et al., 2010):

$$MTTF = A_0 e^{\frac{E_a}{kT}} \tag{5.1}$$

with A_0 being an empirically fitted constant, E_a the activation energy of the specific aging mechanism and k the Boltzmann's constant.

The only exception is thermal cycling; this phenomenon refers to wear-out effects caused by temperature variations which produce an inelastic deformation, eventually leading to failure. Thermal cycling depends on both maximum temperature, and amplitude and frequency of the temperature variations as shown in Coffin-Mason equation (Xiang et al., 2010). This equation measures the average number of thermal

cycles N_{TC} remaining before device failure:

$$N_{TC} = A_{TC} \left(\delta T - T_{th}\right)^{(-b)} e^{\frac{E_{a_{TC}}}{kT_{max}}} \tag{5.2}$$

where δT is the thermal cycle amplitude, T_{th} is the threshold temperature where the inelastic deformation begins, T_{Max} is the maximum temperature during the cycle, A_{TC} is an empirically determined fitting constant, b is the Coffin-Mason exponent constant, and $E_{a_{TC}}$ is the activation energy for thermal cycling. Then the MTTF due to thermal cycling is the product between the number of cycles N_{TC} and the average period of the cycles, i.e. the reciprocal of the cycle frequency.

As a consequence, we can partition the past approaches in works considering aging mechanisms subject to temperature levels (e.g. Kim et al. (2016); Yamamoto and Ababei (2014)) and works considering also thermal cycling (e.g. Chantem et al. (2013); Ma et al. (2017a); Das et al. (2014)). In particular, from the first group, most of the approaches focus on a single mechanism, generally EM since it is one of the predominant issues, claiming that the effectiveness of the approach can be proved also for all the other phenomena.

A more accurate model can be adopted for NBTI (Bhardwaj et al., 2006; Wang and Xu, 2014), capturing the threshold voltage shift and consequent variation on the propagation delays which eventually cause timing errors in the system. This model considers not only the effects of temperature variations but also the transition between stress and recovery phases of the critical logical nets. Thus, a last class of works (Sun et al., 2014; Karpuzcu et al., 2009) is specialized on NBTI. Finally, it is worth mention that the vast majority of works consider aging mechanisms in CMOS technology. Very few works acting at system level consider different technologies; for instance, the approach in (Cai et al., 2016) is specifically targeted for multi-cores implemented in Fin Field-effect (FinFET) transistor technology and consequently adopts specific models better describing the more complex relationship between temperature and aging.

As a final note, for a more accurate discussion on models for lifetime reliability please refer to JEDEC Solid State Tech. Ass. (2010); Xiang et al. (2010); Bhardwaj et al. (2006); Wang and Xu (2014).

Type of workload

Workload execution is obviously the main cause of thermal stress in the device, and workloads are highly dynamic. Therefore, workload distribution plays an important role in lifetime management and different strategies are adopted depending on the types of applications. For this reason, we may classify the past work depending also on the considered type of applications:

- **Sequential applications**, composed of a single thread occupying a single core (e.g. Chantem et al. (2013); Ma et al. (2017a)).
- **Parallel applications**, (one or many), each one spawning a set of threads (e.g. Kim et al. (2016)).
- **Dataflow applications**, that are generally considered in the NoC-based many-core platform and are modeled as a task graph (e.g. Sun et al. (2014); Haghbayan et al. (2017)).

Actuation knobs

All the tuning knobs have an effect on the various optimization drivers (performance, power consumption, temperature, etc.), and, therefore, also on the aging. DRM approaches can be classified on the basis of the set of knobs used in the work. The main knobs are:

- DVFS and per-core power gating (e.g. Srinivasan et al. (2004); Coskun et al. (2009); Das et al. (2014); Ma and Wang (2012)),
- Application mapping (e.g. Sun et al. (2014); Haghbayan et al. (2017)),
- Application scheduling (e.g. Coskun et al. (2009)).

Co-optimization strategy

The first DRM approach by Srinivasan et al. (2004) focused on the optimization of MTTF. However, lifetime is only one of the optimization drivers. As a consequence, most of the subsequent approaches

define a multi-objective optimizations involving mainly performance, power/energy consumption and lifetime. More precisely, various approaches consider a subset of the drivers to define a requirement to be satisfied and perform an optimization of the remaining ones. The two more interesting addressed co-optimization problems are:

- Optimize the performance while satisfying both a power budget and a lifetime target (e.g. Haghbayan et al. (2017)).
- Co-optimize the trade-off between performance and lifetime, possibly considering a power budget (e.g. Ma and Wang (2012)).

Finally, few works consider also soft error susceptibility in the co-optimization (e.g. Ma et al. (2017a,b)); they will be discussed later in Section 5.2.

Based on the discussed parameters, the literature review is discussed in the following. Then, Tables 5.1, 5.2, 5.3, 5.4, 5.5 summarize the discussion based on the aspects highlighted above.

First approaches for single-core CPUs. The idea of a Dynamic Reliability Management (DRM) has been proposed for the first time in 2004 by Srinivasan et al. (2004). The paper proposed an architectural model, called RAMP, to express the MTTF of a single core architecture running a single-threaded application per time, by considering various aging mechanisms. Indeed, the reliability model is quite simplistic since it is based on simplicity on a non-realistic constant failure rate allowing to compute the "instantaneous" MTTF based only on the current working configuration. Then, the paper discusses intuitively and without a concrete definition a very simple management strategy adapting the processor (in particular, by tuning DVFS and other architectural configurations, such as instruction window size or the number of Floating-Point Units (FPUs)) in response to workload change to meet the lifetime reliability target while limiting performance loss. Finally a last relevant contribution of this paper is a first demonstration of how dynamic temperature management is not optimal to control aging; in fact, this last one is not able to capture the long-term nature of the

aging process. In the subsequent years, few other approaches consider a similar scenario. In particular, Karl et al. (2008) defined an enhanced reliability model considering aging history. Then, the proposed approach controls DVFS by means of a Proportional-Integral-Derivative (PID) controller taking the currently estimated reliability as an input.

Approaches for multi-cores. Subsequent works manage lifetime reliability in multi-core architectures. In particular, Coskun et al. (2009) investigated the effects of task scheduling and DVFS on the lifetime reliability of a multi-core system executing a workload composed of single-threaded applications. Various common dynamic policies for scheduling (called stop_go, migration, balance, etc.), DVFS (DVFS-threshold, DVFS-performance, etc.), and cores' selection are systematically combined and evaluated in terms of MTTF, computed by means of RAMP, energy and performance. It is worth mentioning that this management strategy does not exploit a reliability-estimation feedback loop to drive decisions.

A similar scenario is considered in Ma and Wang (2012) where an approach called *PGCapping* receives in input from the architecture about current power consumption, per-core utilization, temperature and current reliability (computed by means of RAMP) and decide how to act on per-core power gating and DVFS to avoid exceeding a power budget, optimize performance, and balance lifetime; *PGCapping* integrates a Proportional-Integral (PI) controller and an iterative algorithm to separately control the two knobs. An interesting aspect of approaches focusing on multi-core architectures is the aim of balancing aging among cores in order to intuitively prolong the overall lifetime of the system. In general, they focus on some variation of the discussed scenario. Finally, Cai et al. (2016) proposes an approach to control DVFS aimed at reducing the aging effects in FinFET-based multi-cores.

Many other approaches have been proposed in the scenario of homogeneous multi-core systems running a multi-programmed and possibly parallel application workload. For instance, in Mercati et al. (2017), a more accurate aging model is considered and DVFS is controlled by means of a two-stage controller based on predictive models; in particular a long term controller is able to perform reliability budgeting to allow

to age faster when performance boost is required and recover in the idle period. The approach of Simevski et al. (2014) defines a simple round-robin scheduling strategy selecting the youngest core for the execution of the subsequent application. Another solution has been proposed by Das et al. (2014) where machine learning strategies are used to identify the optimal configuration in terms of workload mapping and DVFS control to handle both EM and thermal cycling aging. An interesting aspect of Mercati et al. (2017) and Das et al. (2014) is the fact that the proposed controller is actually implemented and tested on a real board. Finally, the approach by Song et al. (2015) systematically evaluated the use of advanced knobs such as phase-aware thread migration, DVFS and turbo boosting to trade off performance against lifetime.

Approaches for many-cores. In the recent past, research effort has been devoted to DRM approaches for novel architectures, such as NoC-based many-core architectures and heterogeneous multi-core platforms. Mapping decisions assume a crucial role in many-core platforms since it is necessary to identify the group of processing cores among the large availability of resources to be used to execute each application in an optimal way; more precisely, the topology of the selected cores has a relevant effect both on the performance, due to the intensive communications required by the several application tasks and the overall thermal stress on the various units.

The approaches by Chantem et al. (2013); Ma et al. (2017a) define run-time mapping policies for single-threaded applications aimed at optimizing lifetime; these approaches feature a reliability emulation module computing the current aging status of each core based on the temperature profile received from the architecture. These works are interesting since they consider various aging mechanism, including thermal cycling, and exploit a more accurate reliability model proposed by Xiang et al. (2010). A similar scenario is considered by Bolchini et al. (2016) where a systematic investigation of various basic mapping policies (workload balancing vs. usage of spare unit) has been carried out to analyze the overall MTTF of a many-core system capable of providing the minimum required performance level after a given number of subsequent permanent failures of cores. The strategy by Karpuzcu

et al. (2009) dynamically acts on V_{dd} scaling in small steps to control the aging of the units in a many-core architecture. The technique defines four different operating modes setting in a different way V_{dd} and the current frequency to obtain different compromises among aging, power consumption and performance. At run-time the controller selects the best operating mode. The work considers an accurate NBTI aging model showing the current degradation to the threshold voltage level. However, single-threaded applications are considered, and the application mapping and power budgeting are not fully addressed.

Many-core applications frequently execute dataflow applications, that are generally modeled with task-graphs. Several other approaches (e.g. Yamamoto and Ababei (2014); Bolchini et al. (2013b); Hartman and Thomas (2012)) consider this scenario. The approaches proposed in Yamamoto and Ababei (2014); Bolchini et al. (2013b) propose migration controllers periodically moving tasks of the application from elder cores to younger ones. Such an approach is sometimes too fine-grained since device aging process is slow (in the order of days). Therefore, a periodic migration of the workload would be necessary only for applications lasting for days or weeks. Another mapping approach in a similar scenario has been discussed by Hartman and Thomas (2012). The decision algorithm performs an almost-exhaustive exploration of the mapping to balance aging and minimize power consumption. Moreover, an interesting aspect of this work is the fact that, as by Bolchini et al. (2016) sequences of subsequent core failures until the overall system denial of service are considered in the evaluation of the approach. Some other technique by Sun et al. (2014) considers the mapping of application task-graphs onto many-core architectures focusing on the NBTI degradation effects. In particular the approach defines a zoning strategy to select the younger area of the resource grid to deploy the arrived applications. Indeed this approach performs a performance vs. lifetime optimization neglecting power consumption. Finally, an interesting aspect of the work presented by Huang et al. (2011) is the mixing between pre-computed design-time mapping solution for each mutually exclusive application to be run with an on-line adaptation strategy capable of refining the mapping according to the actual aging status of the architectural cores.

When considering many-core architectures, some works (Bhardwaj et al., 2012; Wang et al., 2016) have also focused on the NoC infrastructure in order to alleviate the switch aging. In fact, while processing resources are generally interchangeable since, especially in a homogeneous architecture, they are equal each other, failures in switches may cause the interruption in some communication paths. Therefore, such works define adaptive routing algorithms capable of balancing aging while preserving low communication latency.

Another relevant aspect in many-core architectures is the so-called *dark silicon* problem which dramatically limits the number of processing units that can be switched on at the same time (Rahmani et al., 2017a). As shown in many different works (Gnad et al., 2015; Kim et al., 2016), dark silicon becomes an opportunity for lifetime enhancement since the tight power budget offers a larger degree of freedom in the mapping decisions. In this direction, work by Gnad et al. (2015) takes the mapping policy into account as well as the power budget and the process variation status of the cores to mitigate NBTI effects. However, the mapping policy does not consider the topology of the architecture and related communication issues in threads distribution. A similar contribution is given by Kim et al. (2016) proposing a machine-learning strategy to perform aging-aware mapping with respect to the EM phenomenon. A simplified exponential model is used for lifetime reliability, similar to RAMP. Nevertheless, also in this case a shared-memory architecture is considered, and, as a consequence, the mapping strategy is quite simplistic. Nevertheless, none of these approaches take into account DVFS tuning for aging mitigation. The most comprehensive and efficient approach in the scenario of lifetime prolonging for many-cores in the dark silicon era as been proposed by Haghbayan et al. (2017), where the run-time policy is able to map several incoming application task-graphs and acting on DVFS and per-core power-gating to maximize performance while satisfying both the power budget and the given lifetime target.

Approaches for heterogeneous multi-cores. Finally, heterogeneous system architectures have been studied in terms of lifetime efficiency.

The challenge in this kind of architectures is the presence of different types of processing units, such as asymmetric multi-cores, such as the ARM big.LITTLE, GPUs and hardware accelerators; each one of them is capable of providing a different trade-off in terms of performance/power consumption and presents a different efficiency also w.r.t. the characteristics of the specific applications in execution. This has a relevant impact also on the aging of the various components, since most power-hungry units generally age faster than low power counterparts. Therefore, the investigated DRM approaches are aimed at dynamically selecting for each application to be executed on the processing unit offering the highest efficiency at the minimum cost, both in terms of power consumption and aging.

In this perspective, the works by Baldassari et al. (2017); Mück et al. (2017) define two similar reliability-aware run-time resource management controllers for the big.LITTLE multi-core architecture. As for many previous works on different architectures, the controller exploits a feedback loop to sense power consumption, temperature and workload performance; after that, reliability is computed based on temperature sensing by means of a stochastic model (EM is considered by Baldassari et al. (2017) while NBTI and HCI by Mück et al. (2017)). Regarding performance measures, hardware performance counters are used by Mück et al. (2017), while application-level throughput measures are performed by Baldassari et al. (2017). Then, both the two approaches feature a decision policy based on predictive models for performance, power consumption and aging, aimed at selecting for each application to be executed the most suitable unit capable of satisfying reliability constraint and performance requirements and minimizing the power consumption. As shown by Baldassari et al. (2017), this is achieved mainly by migrating on the LITTLE cores all low-demanding performance applications, because such cores present a considerably lower heating thanks to their low-power characteristics. Simultaneously, CPU quota and DVFS are also tuned to the minimum level providing the required application performance to even more reduce the power consumption and, consequently, temperatures. Finally, also workload balancing among the units of the same type helps in making the utilization uniform. The final interesting aspect of this work is the fact

that the defined controller has been implemented on a real big.LITTLE platform, i.e. the Hardkernel Odroid XU3 board featuring a Samsung 5542 chip.

In the heterogeneous architecture scenario, some other works (Chen et al., 2014b; Lee et al., 2018) specifically focus on the GPU resources exploiting the same feedback-loop control approach. Based on the observation that most of the kernels accelerated on GPU are still memory bound, the approach by Chen et al. (2014b) defines a run-time strategy to select the minimum number of streaming multi-cores in the GPU architecture able to provide the desired performance level. Then, the youngest streaming multi-cores are selected for the execution while the rest of the resources are power-gated. In such a way, power consumption and aging, focusing on NBTI, are dynamically minimized. Finally, the approach by Lee et al. (2018) considers a broader scenario w.r.t. the previous work by handling also process variability in the GPU cores. The proposed workload balancer, based also on some architectural improvements, aims at minimizing the aging due to NBTI and HCI mechanisms.

Table 5.1: Summary of lifetime-aware DRM approaches w.r.t. the type of architecture

Architecture	**Techniques**
Single-core CPU	Srinivasan et al. (2004), Karl et al. (2008)
Homogeneous multi-core CPU	Coskun et al. (2009), Ma and Wang (2012), Simevski et al. (2014), Das et al. (2014), Song et al. (2015), Cai et al. (2016), Mercati et al. (2017), Mercati et al. (2017)
Many-core architecture	Karpuzcu et al. (2009), Hartman and Thomas (2012), Bhardwaj et al. (2012), Bolchini et al. (2013b), Chantem et al. (2013), Sun et al. (2014), Gnad et al. (2015), Wang et al. (2016), Kim et al. (2016), Ma et al. (2017a)
Heterogeneous multi-core	Chen et al. (2014b), Baldassari et al. (2017), Mück et al. (2017), Lee et al. (2018)

Table 5.2: Summary of lifetime-aware DRM approaches w.r.t. the considered aging mechanism

Aging mechanism	Techniques
EM, TDDB, ...	Karl et al. (2008), Bolchini et al. (2013b), Simevski et al. (2014), Yamamoto and Ababei (2014), Song et al. (2015), Bolchini et al. (2016), Haghbayan et al. (2017), Mercati et al. (2017)
EM, TDDB, ... + Thermal cycling	Srinivasan et al. (2004), Coskun et al. (2009), Chantem et al. (2013), Das et al. (2014), Ma et al. (2017a)
NBTI	Karpuzcu et al. (2009), Sun et al. (2014), Gnad et al. (2015), Cai et al. (2016), Mück et al. (2017), Lee et al. (2018)

Table 5.3: Summary of lifetime-aware DRM approaches w.r.t. the type of workload

Workload	Techniques
Sequential applications	Srinivasan et al. (2004), Karl et al. (2008), Coskun et al. (2009), Karpuzcu et al. (2009), Ma and Wang (2012), Chantem et al. (2013), Bolchini et al. (2016), Ma et al. (2017a)
Parallel applications	Hartman and Thomas (2012), Bolchini et al. (2013b), Yamamoto and Ababei (2014), Sun et al. (2014), Bolchini et al. (2016), Cai et al. (2016), Kim et al. (2016), Haghbayan et al. (2017)
Dataflow applications	Sun et al. (2014), Das et al. (2014), Song et al. (2015), Gnad et al. (2015), Kim et al. (2016), Mercati et al. (2017), Haghbayan et al. (2017)

5.2 Soft error susceptibility management

Apart from the aging phenomenon, aggressive technology scaling has caused another reliability issue, i.e. a dramatic increase in the susceptibility for digital devices to *soft errors*. The main cause is the transistor miniaturization which has implied a considerable decrease in the critical

Table 5.4: Summary of lifetime-aware DRM approaches w.r.t. the actuation knob

Actuation knob	**Techniques**
DVFS and power gating	Srinivasan et al. (2004), Karl et al. (2008), Karpuzcu et al. (2009), Ma and Wang (2012), Cai et al. (2016), Mercati et al. (2017)
Mapping	Hartman and Thomas (2012), Chantem et al. (2013), Bolchini et al. (2013b), Simevski et al. (2014), Yamamoto and Ababei (2014), Bolchini et al. (2016) Sun et al. (2014)
DVFS, power gating and mapping	Das et al. (2014), Kim et al. (2016), Ma et al. (2017a), Haghbayan et al. (2017)
DVFS, power gating and scheduling	Coskun et al. (2009), Song et al. (2015), Baldassari et al. (2017), Mück et al. (2017)

Table 5.5: Summary of lifetime-aware DRM approaches w.r.t. the optimization strategy

Optimization strategy	**Techniques**
Optimize lifetime	Srinivasan et al. (2004), Karl et al. (2008), Coskun et al. (2009),Chantem et al. (2013), Simevski et al. (2014)
Optimize perf. under power/lifetime limit	Mercati et al. (2017), Haghbayan et al. (2017)
Optimize lifetime under power/perf. limit	Bolchini et al. (2013b), Gnad et al. (2015), Kim et al. (2016), Bolchini et al. (2016)
Co-optimize perf./power/lifetime	Karpuzcu et al. (2009), Hartman and Thomas (2012), Ma and Wang (2012), Song et al. (2015)
Co-optimize lifetime/soft errors under power/perf. limit	Xiang and Pasricha (2015), Ma et al. (2017a), Ma et al. (2017b)

charge necessary to make a memory or a logic element to bit flip, i.e. to commute the stored logic value. Thus, environmental phenomena, such as radiations (alpha particles or neutrons striking the circuit) or electro-magnetic interferences and other sources of noise, present a high probability to cause soft errors thus corrupting the data processing, even if not causing any disruptive permanent effect in the architecture.

Soft error susceptibility in a given working scenario is generally characterized with a constant failure rate. Thus, the *soft error reliability* $R_{soft\_errors}(t)$, being the probability of a single processing unit to complete successfully the execution of a given computation, is modeled with an exponential distribution. Then, $R_{soft\_errors}(t)$ for complex multi-core systems executing a multiprogrammed multi-task workload can be computed based on classical serial/parallel system models (Koren and Krishna, 2007). Advanced soft error reliability model later proposed (Zhao et al., 2008; Xiang and Pasricha, 2015) capture two additional aspects:

- the fact that the failure rate of the processing unit has an exponential relationship with the operating voltage/frequency level, and
- the high process variability affecting modern technologies which causes a heterogeneous characterization of the various processing cores within the same device as in terms of failure rate due to soft errors.

The literature presents some interesting approaches (a limited set when comparing against the impressive number of contributions devoted to lifetime optimization) aiming at performing run-time resource management; their goal is to tune architectural parameters and distribute the workload in order to maximize the soft error reliability, i.e. the probability of the system to correctly execute the running workload.

Zhao et al. (2008) considered the problem of distributing the workload, composed of various single-task applications and tuning voltage and frequency levels, to optimize the trade-off between performance, soft error susceptibility and energy. The proposed solution is statically generated and later adapted at run-time by means of simple heuristics

to refine voltage/frequency tuning. Same research teams have later proposed various other solutions to similar resource management problems to optimize soft error reliability.

Later, Kapadia and Pasricha (2015) have broadened the working scenario by considering also (1) process variation which cause different performance/reliability characteristics in the various processing elements, and (2) the dark silicon phenomenon, which sensibly constrain the power budget to avoid chip burnouts. In this scenario, authors proposed a run-time policy aimed at optimizing system performance and energy consumption while satisfying dark-silicon power constraints, and application-specific performance and soft error reliability constraints. The policy consists of a mapping strategy for applications modeled as task-graphs and DVFS tuning. This work has been later extended by Xiang and Pasricha (2015) to consider in the same scenario a co-optimization of also the lifetime reliability. Given the higher complexity of the extended scenario, a hybrid design-time/run-time solution has been defined to schedule a periodic workload composed of many task-graph applications with an energy budget being variable over the time; the goal is to maximize system performance, lifetime reliability and soft error reliability. Finally, the considered scenario is even more enhanced by Ma et al. (2017b) to consider also a heterogeneous platform, based on the big.LITTLE multi-core paradigm. In this case, the run-time policy performs mapping and voltage/frequency tuning to maximize soft error reliability within a given target lifetime requirement.

Another class of works (e.g. Naithani et al. (2017); Rehman et al. (2016); Rozo et al. (2018)) enhanced the soft error reliability model by considering the fact that soft error susceptibility varies also according to the actual application in execution. Naithani et al. (2017) considered a big.LITTLE architecture and aimed at scheduling applications on the cores to maximize soft error reliability. The work exploits the fact that big and LITTLE cores have different failure rates and moreover various applications present different susceptibility to soft errors. Therefore the state-of-of-the-art Architectural Vulnerability Factor (AVF) and some other derived metrics are used to measure the probability of each application to fail on a given architecture and the online scheduling policy selects the most suitable processing unit to maximize the reliability.

The main idea of this paper is to balance the failure rate by means of AVF index thus obtaining more accurate estimations.

In a similar way Rehman et al. (2016) considered further indexes (Instruction Vulnerability Index, IVI, and Functional Vulnerability Index, FVI) similar to the AVF to characterize and implement different compile-time versions of each application presenting a different trade-off between soft error reliability and performance. Then, a run-time mapping policy was used to select which version of the application to execute and on which resource of the multi-core architecture schedule it to satisfy execution deadlines and maximize reliability. Finally, Rozo et al. (2018) defined a hybrid approach combining a preliminary static scheduling of applications on the multi-core with a dynamic tracing of the fault occurrence in order to update the Fault Vulnerability Factor (FVF) and consequently remapping at run-time to improve the soft error reliability of the system.

Finally, in Naithani et al. (2018), another runtime scheduler is defined to monitor the reliability characteristics of all applications running on a heterogeneous multi-core, and dynamically distribute applications to the different core types to maximize system reliability, in terms of System Soft Error Rate.

A summary of the discussed techniques is presented in Table 5.6.

Table 5.6: Soft error susceptibility management

Reliability model	**Techniques**
Soft error rate	Zhao et al. (2008), Kapadia and Pasricha (2015), Xiang and Pasricha (2015), Ma et al. (2017b)
Soft error rate + AVF	Naithani et al. (2017), Rehman et al. (2016), Rozo et al. (2018), Naithani et al. (2018)

5.3 Online fault management

The last aspect to be discussed in this section regarding the reliability in on-chip dynamic management covers the mechanisms and the strategies to implement online fault management within the device. Indeed, when

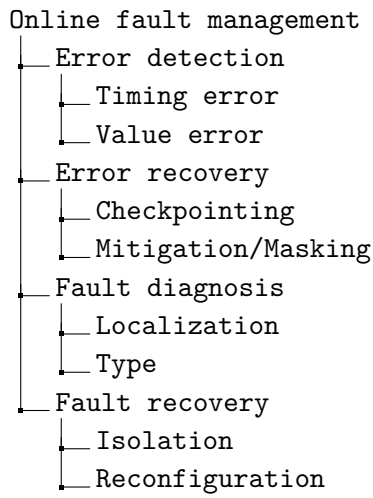

Figure 5.2: Criteria considered for classifying fault management approaches.

a fault occurs, the system has to be able to detect it and execute some kind of management action both to complete successfully the elaborations and to investigate the causes of the failure and, if possible, take some recovery actions. As in the previous cases focusing on lifetime and soft error susceptibility management, the availability of a large set of "programmable" processing cores in modern systems can be used opportunistically to implement some redundancies in the workload execution thus enabling fault management features.

There is a huge literature on fault tolerant mechanisms for modern multi/many-core systems. Our choice here is to discuss only those approaches that exploit the concept of dynamic resource management for online fault management purposes. In other words, we do not analyze architectural solutions provided with hardware based mechanisms; instead, we consider those approaches mainly relying on an accurate distribution of the workload and eventually some additional redundant applications on the available resources of a plain/non-hardened architecture, and tuning of the architectural parameters in order to enable fault management.

An interesting classification of fault tolerant mechanisms has been proposed by Mushtaq et al. (2011) considering multi-core systems. We

here reconsider such taxonomy in order to update and refine it to the current scenario considering on-chip resource management and we re-perform the literature review. The refined taxonomy of online fault management approaches is shown in Figure 5.2. We mainly identify four different classes discussed in the following sections (actually the first two classes are discussed together since the techniques are generally tightly coupled and employed in conjunction), while a summary of the most relevant analyzed approaches is reported later in Table 5.7.

5.3.1 Error detection and recovery

Error detection mechanisms are devoted to identifying the presence of an error whenever the fault affecting the system is activated. The effect of a fault may be of multiple types and consequently different techniques need to be applied:

- **Timing errors.** The application does not terminate at the expected execution time.
- **Crashes/Exceptions.** The application crashes and/or an exception causes its interruption.
- **Wrong result.** The application terminates generating a wrong result.

Timing errors are generally managed by means of watchdogs (e.g. Mahmood and McCluskey (1988)), i.e. hardware timers; the watchdog is set with a deadline tuned according to the application execution time, so that its expiration signifies the timing error. Moreover, crashes and exceptions can be directly managed by the operating system, possibly with the help of watchdogs. As a consequence such error detection mechanisms are quite simple and are not very relevant in a discussion on dynamic resource management.

On the other hand, wrong results are generally identified by means of redundancies in the execution. The standard approach adopted in the considered scenario is the Duplication With Comparison (DWC) applied at software level. Here the challenge from the run-time resource management point of view is the necessity to distribute and schedule the

software replicas on the available processing units mainly to limit the overhead due to the additional computations. For instance, Mukherjee et al. (2002) investigated various thread level duplication techniques based on a Simultaneous Multi-Threading (SMT) mechanism or a chip-level multi-core architecture. In the former the two redundant threads are executed on the same core featuring SMT, while in the latter they are executed on two different cores within the same device. It is worth noting that the two threads are executed in a tight fashion with a strict synchronization of input and output transmissions with the memory, and specific hardware structures are required to perform the online comparison of redundant (intermediate) outputs. Such strict synchronization has been relaxed in a subsequent proposal by Smolens et al. (2006) by decoupling thread execution. Finally, with the aim of balancing reliability and performance, Wells et al. (2009) proposed process duplication only for applications requiring fault detection while exploiting the two cores to perform parallel processing thus maximizing performance in the rest of the cases.

Software redundancies are also used to implement error recovery mechanisms. In particular, fault mitigation is generally implemented by using three replicas, dubbed as Triple Modular Redundancy (TMR), so that it is possible to perform a majority voting to identify the correct result. Further approaches exploit N-Modular Redundancy (NMR) to increase the tolerance to multiple failures and re-execution/checkpointing strategies thus exploiting time redundancy. To mention relevant works, Shye et al. (2009) proposed Process-Level Redundancy (PLR), a software layer running on the top of the operating system, automatically managing process triplication and their synchronized execution. Vargas et al. (2018) proposed an approach to improve the reliability of NoC-based multi/many-core architectures. In particular, NMR is used to achieve fault tolerance, and replicas are distributed on isolated partitions to guarantee that no faulty replica contaminate the execution of any other replica. Then, Chen et al. (2007) adapted at run-time the checkpointing schema according to the architecture's health and current working configuration aiming at optimizing performance. In another work, Subasi et al. (2017) applied at run-time selective replication in a

HPC system in order to improve reliability of the system while decreasing resource costs. Finally, Bolchini et al. (2012) defined a run-time fault management layer for many-cores architecture aimed at scheduling TMR replicas of multi-threaded applications in order to optimize the global system performance.

A more advanced run-time fault management approach is proposed by Bolchini et al. (2013a). The approach focuses on multi/many-core architectures and considers different hardening techniques, such as DWC, TMR and DWC+re-execution, that can be applied at different granularity levels on the running multi-threaded applications (i.e. voters and checkers can be placed for each task or only on the final application output). Thus, the approach is provided with a wide set of hardening techniques offering the possibility to tune the achieved fault management features vs. the performance overhead due to the redundancies. When a quite low error rate is experienced, DWC is used just to signal the occurrence of errors. Then, if errors are frequent, TMR is applied at task granularity level. In this way, the approach is able to mitigate the effect of the faults and, additionally, based on minority voting and error frequency count, diagnose possible failed components. In between such two extreme situations, other techniques offer intermediate fault management benefits and performance overheads. Finally, a scheduling strategy is devoted to balance the load caused by the various task replicas on the available cores, and at the same time satisfying mapping constraints related to the applied techniques.

Chen et al. (2016) considered performance heterogeneity of the architecture in the mapping of redundant applications on multi-cores due to the aging and variability phenomena; in particular cores present different maximum operating frequencies. The proposed mapping policy decides at run-time on which cores it should distribute the redundant threads and which redundant scheme to use (TMR or none) in order to maximize reliability and guarantee performance constraints in terms of deadlines. Indeed, it is worth noting that even though heterogeneity is predominant in modern architectures, this is the only work addressing this aspect.

5.3.2 Fault diagnosis

The goal of fault diagnosis is to locate the faulty component within the architecture in order to take subsequent recovery actions. Fault diagnosis can be performed concurrently with error detection and recovery or in subsequently.

Fault diagnosis, and in particular localization, can be performed concurrently to fault detection/mitigation by exploiting TMR minority voting; the replica with the minority result is faulty, as performed for instance by Bolchini et al. (2013a). This kind of approach is generally suitable for any possible type of fault (permanent or transient). Then, an exact classification of the type of fault can be performed by re-execution; if the error persists the fault is permanent, otherwise transient. Nevertheless, in order to increase the localization capabilities, TMR has to be applied at finer granularity levels as shown by Bolchini et al. (2013a), where TMR is applied at task level. A similar diagnosis approach is used by Hari et al. (2009), where the first execution after the failure, executed on the same core of the multi-core architecture, is used to identify transient faults. Then, if the error is detected another time, a second re-execution on a different core is used to distinguish permanent faults and design bugs. Again, in all these approaches, application scheduling and, therefore, run-time resource management is forced by the constraints of fault diagnosis activities.

Fault diagnosis can be performed also as an independent strategy. One of the main strategies is the online self-testing that is run concurrently to the execution of the nominal applications. In particular, this approach is devoted to identifying permanent faults, due to their persistent nature. It can be mainly performed by using hardware Built-In Self-Test (BIST) (Hetherington et al., 1999) mechanisms or by using Software-Based Self-Test (SBST) (Psarakis et al., 2010) programs. In this survey, we focus on the last approach since it is the only one "compliant" with the dynamic resource management. In particular, the idea of SBST is to run specific software routines concurrently to the nominal workload to perform an online testing of the resources within the architecture in order to identify permanently failed components. The integration of the SBST approach within the dynamic resource

management framework is obtained by enhancing the scheduling algorithm in order to issue self-test routines periodically based on a specific policy. As discussed by Skitsas et al. (2018), there are two possible scheduling approaches:

- periodically initiate the testing simultaneously on all the cores of the system (e.g. Apostolakis et al. (2009)), or
- run SBST routines on the various cores independently based on a given period (e.g. Skitsas et al. (2018)) or selectively based on a priority rule (e.g. Skitsas et al. (2016); Haghbayan et al. (2016b)).

As an example of the former class, Apostolakis et al. (2009) proposed an SBST approach for symmetric shared-memory multi-cores exploiting core level execution parallelism to run the test concurrently on all the cores thus reducing the execution time. Then, Foutris et al. (2010) extended the previous work by using multi-threading thus even more improving performance. However, the main limitation of this approach is the fact that the overall architecture is forced to go offline during the test.

To overcome this limitation, in the second class of approaches cores are selectively tested thus exploiting an ad-hoc scheduling policy to limit the impact on the system performance. Haghbayan et al. (2016b) proposed an SBST routine scheduling algorithm for many-core architectures aimed at testing cores on the basis of the suffered stress, in terms of an aging metric, during the execution of the nominal workload. The scheduling policy is aimed at avoiding any degradation in system performance while not violating the power budget of the many-core system. A similar approach was proposed b Skitsas et al. (2016) for multi-core architectures optimizing the selection of the candidate core based on the organization of the cache hierarchy. Being memory intensive programs, SBST routines executed by some core can be subsequently reused from some other units sharing some cache level so that cache hit percentage is maximized and consequently performance overhead is limited. Finally, other two approaches proposed by Skitsas et al. (2018); Kaliorakis et al. (2014) perform periodic test scheduling by considering the memory hierarchy in a similar way as Skitsas et al. (2016) considering multi-core

and many-core architectures, respectively, with the aim at minimizing the overall test time thus increasing the service time.

5.3.3 Fault recovery

Once a unit is diagnosed as damaged, the last step is to perform fault recovery actions mainly to isolate the damaged component. Again, this strategy is particularly suitable to multi/many-core architectures, where the cores or functional units can be disabled when faulty and the system activity can continue with the remaining healthy ones. In this way, this strategy offers a sort of graceful degradation to the system, allowing to subsequently disable cores, and, consequently, degrading maximum computational power achievable. Thus, the system will survive until there is a minimum number of resources capable at providing the minimum Quality of Service (QoS) required to the system.

For instance, both Bolchini et al. (2012) and Bolchini et al. (2013a) integrated such a recovery strategy in their framework for many-cores to disable faulty cores at run-time. In such a way they provide full-fledged solutions aimed at performing all fault management actions (from error detection to fault recovery) and thus adapting provided performance levels. Chou and Marculescu (2011) integrated the isolation of faulty units in their resource management approach for NoC-based many-core architectures running task-graph based applications; the approach aims at optimizing performance and communication energy. Another relevant approach has been presented by LaFrieda et al. (2007) where units of a multi-core architecture are coupled dynamically to implement DWC. This flexibility allows also for tolerating permanently failed units by means of isolation, thus reorganizing the core pairs for DWC purposes.

At a finer granularity, Aggarwal et al. (2007) proposed a modular multi-core architecture where it is possible to execute applications in DWC or TMR lock-step configuration. When a faulty component is found (e.g. integer unit, cache bank, or memory controller) the architecture can be reconfigured to isolate the failure thus remapping the computations on another spare unit. It is worth mentioning another work proposed by Gupta et al. (2008) where a resilient reconfigurable multi-core architecture is proposed; the architecture is designed as a

Table 5.7: Online fault management

Error detection	**Timing error**	**Error value**
	Mahmood and McCluskey (1988)	Mukherjee et al. (2002), Smolens et al. (2006), Wells et al. (2009)
Error recovery	**Checkpointing**	**Mitigation/masking**
	Chen et al. (2007), Bolchini et al. (2013a)	Shye et al. (2009), Vargas et al. (2018), Subasi et al. (2017), Bolchini et al. (2013a)
Fault diagnosis	**Localization**	**Localization+type**
	Bolchini et al. (2013a)	Hari et al. (2009), Apostolakis et al. (2009), Skitsas et al. (2018), Skitsas et al. (2016), Haghbayan et al. (2016b)
Fault recovery	**Isolation**	**Reconfiguration**
	Bolchini et al. (2013a), Chou and Marculescu (2011), Chou and Marculescu (2011)	Aggarwal et al. (2007), Gupta et al. (2008), Psarakis et al. (2014)

reconfigurable network of replicated processor pipeline stages. In case a pipeline stage is affected by a fault, the architecture is reconfigured in order to isolate the faulty stage and replace it with the equivalent element of any other pipeline. Again, permanent faults will cause a progressive degradation of performance. Moreover such finer grained reconfiguration requires a specific network for interconnecting pipeline stages of all the cores within the architecture. Finally, this component-based isolation strategy is also employed by Tzilis et al. (2016); they proposed a mapping approach exploiting a sort of health table to collect for each core in the many-core architecture which are the healthy components and the failed ones. Thus, an application, or a task, can be mapped on a given core only if this last one provides all components required for the execution. As an example, a task requiring floating point operations can be performed only on cores where the FPU is healthy. The advantage of this approach with respect to the previous two is that there is no need of specific hardware mechanisms to implement component isolation and core reconfiguration.

A last type of approach performing fault recovery specifically targeted to Field Programmable Gate Array (FPGA) devices is proposed by Psarakis et al. (2014). The authors consider a self-healing processor system implemented on FPGA devices. In this scenario soft errors affecting the configuration memory present a permanent effect. Thus, the processor is capable of identifying possible faults affecting the configuration memory by means of hardware-based detection mechanisms; such mechanisms trigger the execution of a specific software routine performing a reconfiguration of the faulty region in order to restore the correct hardware functionality.

5.4 Summary

We presented run-time resource management techniques aimed at handling reliability issues. In particular, we have discussed three different issues that are: (1) lifetime management, (2) soft error susceptibility management, and (3) online fault management. Most of the techniques are aimed at steering the workload distribution, resource assignment (mapping and/or scheduling) and architectural tuning (DVFS and similar knobs) to co-optimize reliability (aging, soft error susceptibility, and fault management capabilities) and some other classical metrics, i.e. performance and power consumption. The main difference among them is the different working scenario each one of them considers (e.g. type of architecture or workload, type of reliability issue, ...). Consequently, the result is a large variety of run-time strategies each one tailored to the specific peculiarities of the considered scenario.

The version of record is available at: http://dx.doi.org/10.1561/1000000055

6

Quality of Service

Quality of Service (QoS) is one of the primary metrics to qualitatively evaluate the system's efficiency in satisfying applications' requirements. With varying algorithmic models, user interaction, responsiveness, end results, compute, storage, network and Input/Output (I/O) requirements, qualitative significance of resources vary among applications. Thus different applications perceive different measurable system parameters as QoS. For instance, streaming applications consider per-input latency to be significant, whereas batch applications consider per-input throughput as a relevant QoS metric, rather than latency (Guo et al., 2007b). On the other hand, enterprise applications consider guaranteed resource allocation (infrastructure-as-a-service) itself as defining QoS measure (Zhou et al., 2016). Examples of QoS metrics from widely used latency, throughput, I/O and service centric application domains are presented in Table 6.1. QoS remains one of the first order design constraint in soft, hard and mixed criticality real-time systems, where the notion of QoS corresponds to lower worst case execution time and guarantees on meeting hard deadlines. Most of the techniques in the context of real-time systems focus on individual jobs, schedulability in presence on other concurrent jobs and arbitration among them to

guarantee meeting deadlines (Burns and Davis, 2013). In the context of this survey, we move the abstraction higher, focusing on user and application specific QoS measures. As such, QoS metrics are typically expressed at user/application level such that they can be translated into system level parameters to measure levels of QoS being provided and to scale resources to match specified QoS requirements (Herdrich et al., 2016).

Table 6.1: Examples of QoS metrics

Application	QoS metrics
Multimedia	Frame rate (Hamers and Eeckhout, 2010)
Streaming	Latency per input (Guo et al., 2007a)
Data processing, analysis	Throughput (Yang et al., 2013)
Enterprise, business analytics, SaaS, IaaS	Availability (Ranganathan and Jouppi, 2005)
Web search, financial analytics	Latency per query (Lo et al., 2014)
Social media	End user latency (Mars et al., 2011)
User, service centric	Responsiveness (Papazoglou and Georgakopoulos, 2003)

With diverse requirements of applications and a range of tunable knobs to guarantee those requirements, on-chip support becomes necessary to guarantee QoS (Grot et al., 2011). However, run-time QoS management becomes challenging with (i) variable workload characteristics and QoS requirements of applications, (ii) expression and translation of user/application level QoS metrics into system level parameters, and (iii) resource contention and arbitration among concurrent applications. On-chip resource management to satisfy QoS requirements while meeting system objectives is further complicated with frequent conflicting resource allocation decisions (Zhao et al., 2016). A widely used strategy for QoS management is to *translate* input QoS metrics into system parameters, *monitor* the system level metrics and *decide* on resource allocation, such that they are scaled to satisfy specified QoS (Zhao et al., 2016). QoS requirements among applications vary largely based on

- nature of computation - compute, memory and I/O intensity, streaming inputs and batch processing
- nature of end result - numerical, perceptive, soft and hard real-time, and user-interaction

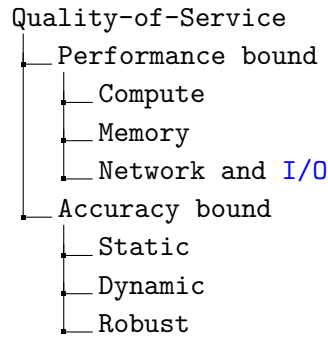

Figure 6.1: Abstract classification QoS management

This leads to an abstract classification where QoS requirements are often eventually *performance-bound* and/or *accuracy-bound*. Figure 6.1 shows a summary of techniques for on-chip QoS management. While performance bound QoS metrics are met through allocating compute, memory and network bandwidth resources, accuracy-bound QoS metrics are guaranteed through quality monitoring and control. In the following sub-sections, we present major underlying approaches and strategies for QoS guarantees in the context of multi-core and many-core systems running concurrent applications, enabled with user/application defined QoS targets and system level QoS measures.

6.1 Performance bound QoS

Existing performance-bound strategies allocating and/or dynamically scaling enough amount of platform resources viz., compute, memory, network and I/O bandwidth to satisfy QoS requirements. Allocation decisions, particularly running concurrent applications (and/or tasks within an application) with diverse QoS requirements needs further arbitration - to identify and alter applications' priorities, for efficient resource allocation and utilization. In this context, run-time resource allocation policies can be majorly classified as elite, utilitarian and fair - based on their dynamic prioritization criteria. *Elite policies* prioritize applications with higher QoS requirements while *utilitarian approaches* prioritize applications with efficient resource utilization. Both these approaches allocate more/enough amount of resources for prioritized

applications to guarantee QoS. *Fair policies* allocate resources equally among all applications such that concurrent applications do not unfairly suffer from prioritized applications. In the following sections, we present QoS oriented resource management techniques based on allocating compute, memory and network resources, which implicitly fall under one of the aforementioned policy classes. An overview of different QoS oriented resource allocation techniques are summarized in Table 6.2

6.1.1 Compute

We focus on techniques that allocate compute resources to meet QoS requirements, including application to core assignment, larger Central Processing Unit (CPU) time slices and higher voltage and frequency levels. QoS management techniques rely on requirements expressed at user/application level, allocate resources to match those requirements and dynamically scale the resources (if needed) by monitoring run-time performance metrics (Zhou et al., 2016; Tang et al., 2012). Typical requirements include latency, throughput and guarantees on compute infrastructure etc., while monitoring metrics include instructions-per-second and speed up. Combining a set of cores, memory and network bandwidth into a *compute infrastructure package* and providing such packages to applications as per their QoS requirements is a simple yet effective approach (Zhou et al., 2016). Specifically, enterprise applications which consider guaranteed allocation of compute infrastructure itself as a quality metric, referred to as infrastructure-as-a-service (IaaS), benefit from this approach (Zhou et al., 2016). While allocation decisions are deterministic and expected QoS is guaranteed, it might lead to resource over/under utilization under workloads with varied QoS requirements - largely due to the enforced isolation among different applications. This issue can be addressed by clustering applications with similar requirements into a batch and allocating resources accordingly, avoiding fragmentation i.e., under-utilized/interleaved compute resources (Lo et al., 2014). A further optimized strategy is *smart co-location* of concurrent applications with diverse requirements at application mapping and scheduling phases (Mars et al., 2011). This approach tries to find a suitable combination of applications with diverse requirements which

Table 6.2: QoS provisioning

Resource	Allocate	Techniques
Compute	More cores, higher parallelism, heterogeneity,customized cores	Application/thread mapping task migration, scheduling
Storage	Larger cache slices - capacity Adaptive memory allocation - bandwidth	Cache partitioning, page coloring proximity to memory mapping
Network and I/O	Customized interconnect architecture, higher network and I/O bandwidth	Location aware mapping and adaptive routing for I/O and network bandwidth

can fit into a batch, such that QoS requirements can be met while also retaining efficient resource utilization (Yang et al., 2013).

Apart from the aforementioned approaches, another class of run-time techniques leverage heterogeneous architectures for QoS guarantees. A combination of general purpose, asymmetric and specialized cores within heterogeneous systems expose more options for executing - (i) concurrent applications with diverse requirements (Petrucci et al., 2015) and analogously for (ii) concurrent tasks within an application (Joao et al., 2012). QoS requirements of each application/task are identified by translating user/application level specifications. Based on these metrics, run-time techniques choose execution units (among heterogeneous options available) that are more suitable for each application/task such that QoS requirements are met (Li and Nahrstedt, 1999; Yang et al., 2013). Dynamic adaptation and resource scaling decisions are made by monitoring system level performance metrics such as speed up, instructions-per-second etc. (Herdrich et al., 2016; Tang et al., 2012; Zhang et al., 2014). These techniques are particularly beneficial for concurrent workloads with diverse requirements, where run-time decisions can match each application/task's QoS requirements appropriately with available heterogeneous options - satisfying QoS demands with efficient resource utilization (Delimitrou and Kozyrakis, 2013; Lo et al., 2014).

6.1.2 Memory

Contention for lower levels of cache and With compute-memory performance gap, allocating larger cache slices and higher memory bandwidth significantly enhances performance bound QoS metrics (Sung et al., 2017). Concurrent applications with diverse memory access patterns, compute-memory phases and intensities and cache utilization is a major

reason for application slow down and QoS degradation (Tang et al., 2011; Subramanian et al., 2015). Cache partitioning to provide either larger or at least sufficient enough last level cache slices is a common approach to meet QoS requirements of latency critical applications (Kasture and Sanchez, 2014; Iyer et al., 2007). Identifying application/thread priority and scaling cache allocation accordingly, following utilitarian principles is another strategy to improve overall throughput metrics (Herdrich et al., 2016; Sharifi et al., 2011; Sung et al., 2017; Guo et al., 2007a). Optimizing for memory controller proximity reduces shared resource pressure further and provides predictable QoS guarantees (Beckmann et al., 2015; Subramanian et al., 2015). Allocating higher memory bandwidth to applications that gain from such provisioning through smart paging is proposed in Ye et al. (2014); Zhang et al. (2009); Cho and Jin (2006). Memory provisioning techniques typically monitor low level resource utilization metrics such as cache miss rate, memory access penalty, application progress and cache utilization factor for dynamic prioritization (Guo et al., 2007b). While software defined allocation decisions are embedded within operating system level scheduling policies, efficient dynamic cache and memory allocations decisions require micro-architectural extensions to identify and translate between user/application defined QoS performance metrics and system level QoS utilization metrics (Li et al., 2012, 2011; Herdrich et al., 2016).

6.1.3 Network and I/O

Allocating higher network and I/O bandwidth to prioritized applications can guarantee latency and throughput QoS requirements. Existing techniques have used customized router architecture, virtual channels, flow control and frame scheduling to provide higher network bandwidth for dynamically identified priority applications. Classification of network into shared resource and non-shared resource clusters to allocate non-QoS and QoS tasks respectively through novel router architecture was proposed in Grot et al. (2011). Assigning each flow into frames and intelligent scheduling globally synchronized frames to optimize for latency is proposed in Lee et al. (2008). The same idea is extended by Ouyang and Xie (2010) with a flexible local frame scheduling and

preemptive flit reservation for more bandwidth for high priority applications. Distinguishing between latency and throughput sensitivity of best effort (BE) and guaranteed throughput (GT) to optimize their respective flow control is proposed by Diemer and Ernst (2010); Diemer et al. (2010). While BE applications are prioritized by default, priority is inverted to GT when BE applications have used enough buffer space reflecting in a certain throughput guarantee. Assigning a fixed bandwidth to each flow and monitoring its bandwidth utilization to prioritize and allocate network resources to utility frames is proposed in (Grot et al., 2009b). A similar approach with hybrid fair and elite round robin bandwidth allocation using weighed priorities is proposed in (Heißwolf et al., 2012). Each of these techniques dynamically determine priority of packets (originating from priority applications) and route them first, while other low priority packets wait in the queue.

Table 6.3: Performance oriented QoS management techniques

Scheduling	**Design spec.**	**Robust**
	Delimitrou and Kozyrakis (2013), Lo et al. (2015), Petrucci et al. (2015), Shelepov et al. (2009), Van Craeynest et al. (2013)	Koufaty et al. (2010), Delimitrou and Kozyrakis (2014), Saez et al. (2012)
Provisioning	**Compute**	**Platform**
	Wang and Martínez (2016), Wang and Martínez (2015), Zahedi and Lee (2014), Saputra et al. (2002)	Kasture and Sanchez (2014), Beckmann et al. (2015), Sanchez and Kozyrakis (2011), Beckmann and Sanchez (2013), Kim et al. (2004), Cho and Jin (2006), Zhang et al. (2009), Ye et al. (2014)

Managing QoS through Co-scheduling: In addition to provisioning, spatial scheduling of concurrent applications and tasks within an application to suitable cores can improve both per-application latency and per-chip throughput. Mars et al. (2011); Yang et al. (2013); Zhao et al. (2016) have shown that contention for shared resources of memory and functional units among concurrent applications is the reason for application slow down due to interference. Mars et al. (2011);

Subramanian et al. (2015) quantified the extent of QoS degradation of certain workloads when run with other workloads to predict an estimate. This can be used to intelligently schedule applications by co-locating them such that QoS and utilization are met. Yang et al. (2013) extended this principle into a monitor (bubble) and act (flux) phase. QoS of latency sensitive applications are observed in bubble phase for estimates and during flux phase, batch applications are scaled down as per QoS requirements met in bubble phase.

6.2 Accuracy bound QoS

Thus far, we have described QoS in terms of performance with an implicit assumption that all computations are accurate. However, applications from certain domains present tolerance to inaccurate computations, due to their inherent error resilience. Applications such as multi-media processing, big data analytics, computer vision, machine learning, internet-of-things etc., often deal with redundant, noisy analog data and rely on iterative and probabilistic algorithms (Esmaeilzadeh, 2015; Nair, 2015). Such computation models and input data characteristics makes these workloads tolerant to inaccurate results. Approximate computing has emerged as another paradigm for performance and energy gains by relaxing accuracy (Esmaeilzadeh et al., 2012a). Performance demands of emerging workloads and energy efficiency of resource constrained computer systems can be addressed with approximate computing, leveraging the inherent error resilience. Despite the performance and energy gains, approximation requires disciplined tuning to guarantee an acceptable quality of end result. In this section, we discuss various techniques that exploit accuracy trade-offs in a disciplined and controlled fashion to maximize performance and energy gains while striving for minimizing quality loss. While the previous section provides a performance bound perspective, this section presents strategies for accuracy bound QoS which focus on quality of result.

Accuracy bound quality management requires strategies to enable approximation, on top of which run-time systems for dynamic quality management and control are deployed. Implementing approximation requires interaction among all layers of the computing stack ranging

from algorithms, programming languages, compilation, ISA extensions, architecture through logic and devices. In the context of this survey, we first present techniques to enable approximation, followed by different run-time quality control approaches.

6.2.1 Enabling accuracy trade-offs

We present different techniques which realize approximation at run-time across the computing stack, split into two steps (1) identification and expression of error resilient behavior and (2) exploiting the error resilience through architecture and hardware implementation.

Identification and Expression There are a range of works which identify approximable regions of code (Carbin and Rinard, 2010; Rinard, 2006; Carbin et al., 2011; Kling et al., 2012) and express them through programming language constructs (Misailovic et al., 2010; Sidiroglou et al., 2011; Carbin et al., 2012, 2013; Rinard, 2007; Bornholt et al., 2014) at a software level to realize approximate execution. Discarding a sub set of non-critical tasks within an application to reduce the workload was proposed in Rinard (2006), and provided an acceptable end result. Similarly, skipping some iterations over compute intensive and bottle neck loops, *loop perforation*, for reduced number of computations within acceptable quality loss was proposed by Misailovic et al. (2010) and Carbin et al. (2011). Further, optimizing the Pareto-space for the number of loops to be skipped to maximize performance and minimize accuracy loss with rigorous off-line training inputs has been explored by Sidiroglou et al. (2011), while *Relax* (Carbin et al., 2012) provides formal constructs to identify and express loop and code perforation and task skipping (Carbin et al., 2012). Although the identification and expression of these techniques happens at design time, approximation is realized at run-time, although without requiring on-chip management. *EnerJ* proposes programming language constructs and compiler extensions for expressing error tolerant behavior of an application through approximate data types (Sampson et al., 2011). The idea is to run the specified instructions and corresponding data objects on approximate hardware, realizing approximation at run-time. Samadi et al. (2013) and

Samadi et al. (2014) provide dynamic compilation support to generate approximate versions of compute intensive kernels to reduce workloads, without tweaking the hardware.

Architectural and hardware implementation An identification, expression and translation mechanism similar to *EnerJ* was used by Esmaeilzadeh et al. (2012a,b); St Amant et al. (2014); Moreau et al. (2015), with implementation of approximate hardware being the major distinction among them. Truffle (Esmaeilzadeh et al., 2012a) uses voltage over-scaling to realize distinct accurate and approximate versions of cores and provide Instruction Set Architecture (ISA) extensions to support expression of approximation. Esmaeilzadeh et al. (2012b); St Amant et al. (2014); Moreau et al. (2015) follow a similar approach to *Truffle* infrastructure, however using neural functional units and limited precision analog circuits to provide approximate functionality. When a compiler pass indicates an error resilient region, approximate hardware is invoked at run-time and the corresponding computational block is implemented on the approximate hardware units. Neural network based Field Programmable Gate Array (FPGA) accelerators for opportunistic approximation at run-time for diverse sets of machine learning and multi-media applications were widely explored by Grigorian et al. (2015); Chen et al. (2014a); Du et al. (2015); Liu et al. (2015); Chen et al. (2014c).

Apart from custom acceleration, traditional prediction and speculation techniques revised with relaxed prediction penalty were presented by Thwaites et al. (2014) and San Miguel and Badr (2014). In both cases, the idea is to exploit value locality of data to execute on predicted data, avoiding fetching new data into the pipeline, using ISA extensions to reflect approximate loads and stores. Full scale ISA extensions for representing approximate instructions is presented by Venkataramani et al. (2013), where each instruction has a *quality* bit that represent the extent of hardware approximation. Flikker (Moscibroda and Zorn, 2011) uses ISA extensions to represent critical and non-critical data and stores them accordingly in accurate and approximate memory units using variable refresh rates for Dynamic Random-Access Memory (DRAM). Similar variable precision storage using hints from the instruction level

were explored by Cui et al. (2014), Qiao et al. (2015) and Shoushtari et al. (2015).

In summary, most of the architectural techniques follow the baseline approach of expressing approximable regions or functional blocks, compilation and ISA extensions to support such expression and hardware units including custom accelerators and storage units which are invoked at run-time to execute on approximate hardware.

6.2.2 Accuracy bound QoS management

Reasoning for accuracy loss, guarantees on end results and error bounds, and control on Quality of Result (QoR) becomes crucial for the viability of approximation techniques. Existing works have largely focused on approximation techniques alone and using profiling, calibration and light weight checks for nominal quality control (Laurenzano et al., 2016). In the context of QoS management, we focus on techniques that strive for maintaining bounded error, guarantees and control on quality of result. QoR of most of the deterministic applications largely depends on input data. While hard guarantees can be provided only over pre-defined input data and deterministic approximation, empirical and statistical guarantees are sufficient for quality control, allowing efficient usage of approximation (Moreau et al., 2017). We divide quality assurance techniques into three categories viz., *static* - profiling, *dynamic* - calibration and *robust* - control and roll back. Table 6.4 shows a summarized classification of QoS management techniques. In addition to the quality control techniques, we also present techniques that exploit approximation to realize specific objectives such as power capping and meeting performance guarantees. These techniques are detailed in the following sections.

Static techniques Static techniques depend on profiling, where each approximation technique is validated against a fixed, perhaps exhaustive, set of inputs to derive empirical guarantees on error. While identification, analysis and modeling of quality loss is static in nature, these techniques employ the obtained heuristics dynamically. Approximation techniques

typically identify error resilient regions of code such as critical sections (Esmaeilzadeh et al., 2012b), forgiving computations (Samadi et al., 2013), resilient data types (Sampson et al., 2011) within an application as candidates for various types software approximations including loop perforation (Sidiroglou et al., 2011), tiling (Samadi et al., 2013) and hardware approximations such neural acceleration (Esmaeilzadeh et al., 2012b) and relaxed functional units (Venkataramani et al., 2013). Static techniques use profiling - executing candidate code blocks over (exhaustive) set of data inputs to extract average and worst case relative error bounds (Samadi et al., 2013; Baek and Chilimbi, 2010; Sampson et al., 2011). Static techniques are as effective as the input data coverage i.e., error bounds can be guaranteed for input sets that are pre-evaluated, which can in turn be used at run-time for quality control.

Table 6.4: Managing accuracy as QoS

Parameter	Static	Dynamic and robust
Compute	Wang et al. (2017b), Ho et al. (2017), Li et al. (2015), Mitra et al. (2016), Carbin et al. (2011), Carbin and Rinard (2010), Misailovic et al. (2010)	Zhang et al. (2017), Raha et al. (2015), Xu et al. (2017), Park et al. (2016), Tziantzioulis et al. (2016), Sui et al. (2016), Mahajan et al. (2016)
Logic	Gebregiorgis et al. (2017), Bruestel and Kumar (2017)	Huang et al. (2012), Li et al. (2015)

Dynamic techniques Dynamic quality control techniques use calibration - executing each candidate block of code over both accurate and approximate methods to determine nature and extent of errors induced at run-time (Esmaeilzadeh et al., 2012b; Sidiroglou et al., 2011). These approaches then either rely on user-defined or application level accuracy requirement targets to determine whether approximate execution is within an acceptable quality range. Some techniques use the target accuracy requirement as a feedback to explore accuracy-performance

Pareto space to configure the extent of approximation (Sidiroglou et al., 2011; Baek and Chilimbi, 2010). Dynamic techniques are efficient in providing empirical and/or statistical guarantees on quality, however they require additional hardware/software overhead for continuous monitoring and execution of both accurate and approximate versions. Reducing sampling rate of monitoring might ignore errors induced during the un-sampled interval.

Robust techniques Robust quality control techniques monitor accuracy loss at run-time and further roll back to accurate execution in case of errors induced beyond acceptable thresholds. Robust techniques address the limitations of static techniques which can provide guarantees only over tested inputs, and dynamic techniques which have overheads and lesser coverage within sampled invocation. Robust techniques use predictive, online learning, light weight checks and monitoring strategies to compute quality loss and predict the extent of quality loss for subsequent inputs (Laurenzano et al., 2016; Wang et al., 2017a). The quality loss is compared against user defined accuracy requirements to determine on either toning down aggressive approximation or selecting a different type of approximation technique (Xu et al., 2017; Grigorian et al., 2015). In case of unacceptable results, these approaches roll back i.e., re-execute the candidate code blocks in accurate mode to cover for the accuracy loss. Robust techniques include re-configuring the extent of approximation (Wang et al., 2017a; Grigorian et al., 2015), re-generation of the type of approximation used iteratively (Xu et al., 2017; Moreau et al., 2017) and pro-actively (Sui et al., 2016; Laurenzano et al., 2016), and roll back by re-executing the code block accurately (Khudia et al., 2015; Mahajan et al., 2016).

Exploiting accuracy trade-offs While opportunistic approximation is open-looped with the target of improving performance or energy efficiency in general, some techniques have used approximation to realize specific goals of maximizing performance within minimum energy and loss of accuracy. Kanduri et al. (2016) have proposed approximation as another knob for power management, to cover up for the performance loss incurred in power capping techniques. They switch the mode of

execution within an application from accurate to approximate and vice-versa subject to system demands. A similar approach to meet deadlines of applications with hard real-time constraints was proposed by Tan et al. (2015). This can be extended in the context of both multi-core and many-core systems running emerging workloads from media processing domains and can be exploited opportunistically subject to application requirements (El-Harouni et al., 2017; Palomino et al., 2016). Particularly with battery operated mobile processors and resource constrained systems, approximation can be used for efficient resource allocation (Kanduri et al., 2018b).

6.3 Summary

We presented both performance-bound and accuracy-bound QoS management techniques. Most of these techniques rely on a wide range of application and system level monitors to translate QoS requirements into measurable parameters for dynamic provisioning. While specific application domains and platforms have benefited from such monitoring and resource allocation, QoS management has been confined to those domains only. A more generalized approach to describe application/user level QoS requirements and formal translation of such requirements as system parameters can ensure wider coverage for QoS guaranteed systems.

The version of record is available at: http://dx.doi.org/10.1561/1000000055

7

Limitations of current approaches and recent trends

As this survey witnesses, the body of work on specific resource management techniques is overwhelming. They cover all levels from devices, circuits, to architecture, operating system and application. They address a variety of optimization objectives for energy, performance, wear out, lifetime, quality of service, and others. Many of them are certainly useful and today there is probably no SoC as part of a product that does not deploy dynamic resource management techniques. The energy efficiency of modern smart phones could not even closely be achieved without extensive resource management and server farms could not be cooled if unused resources were not de-activated even if cooling was free of charge.

Holistic approaches However, as this survey also illustrates, individual techniques for pursuing isolated optimization goals can do only so much. Managing the on-chip network, the Central Processing Unit (CPU) cores, and the off-chip memory independently of each other leads to sub-optimal solutions, and optimizing only for power without considering temperature, lifetime, and quality of service does not live up to the requirements in modern adaptive, resource constrained devices and

systems. Rahmani et al. (2017b) use dynamic task mapping in a multi-core platform as a case study to show that the pursuit of isolated objectives leads to sub-optimal solutions that fail to meet a broader range of goals. A performance driven task allocation algorithm, as proposed by Haghbayan et al. (2015), results in dense mappings with small areas of the chip heated up beyond safe temperature margins that significantly shortens the system's lifetime. On the other hand, a power constraint driven allocation approach, as presented by Kanduri et al. (2015), leads to lower than necessary performance and still compromises lifetime. Finally, a lifetime maximizing task allocation approach, as developed by Haghbayan et al. (2016a), sacrifices performance and still cannot contain temperature peaks that lead to major temperature dissipation challenges. Rahmani et al. (2017b) illustrate that all relevant system objectives have to be considered and optimized simultaneously, lest important performance targets are compromised.

They would be compromised unnecessarily because in most cases there is sufficient slack and margin to meet multiple objectives at the same time, if only the problem definition is broad enough and the set of techniques is large enough. For that reason the broad set of resource management techniques surveyed in the previous chapters are useful when combined, integrated and selectively used by a system level goal manager that can flexibly pursue the goal, or set of goals, that are most critical at a specific time.

However, we observe that in some areas holistic approaches are more popular than in others. Work focusing on Quality of Service (QoS), as covered in Chapter 6, is typically rather single-minded. There are many approaches that target specific objectives using specific resources, but targeting power and performance at the same time, using all, or many, resources is a rarity. Exceptions to this trend can be found in cross layer approaches in Section 6.2.1.

On the other hand, power management approaches always consider also performance objectives as illustrated by proposals for general purpose processors (Rotem et al., 2012), mobile devices (Kadjo et al., 2014), or game engines (Pathania et al., 2014). Most other works in Chapter 4 also take at least performance objectives into account in addition to power or energy.

At the other end of the spectrum a large majority of approaches for reliability, and in particular for lifetime management, may be considered as holistic. Reliability is usually considered as a relevant but also an additional metric. Thus, it is frequently co-optimized with performance and power as shown in Table 5.5 for approaches managing lifetime. In the same way, such approaches consider all the resources and try to act on the vast majority of knobs they can access.

Adaptivity Most of the resource management techniques surveyed here define the optimization objectives at design time and hard-code them in hardware or software. While this approach is efficient for static embedded systems with a predictable application load, it lacks flexibility when type and load of applications vary and are unknown at design time, when the system has to react to unforeseen changes in the environment, and when the system has to cope with unexpected deterioration and anomalies in its own behavior. In those cases preordained objectives are limitations, and the ability to reformulate and re-prioritize objectives at run time becomes a prerequisite for truly adaptive and autonomous systems. A case in point studied by Hoffmann et al. (2012) are applications that come with their own objectives with respect to power and performance requirements. Each application registers its objective with the platform, which in turn tries to meet these objectives. The platform is based on the application heartbeats framework (Hoffmann et al., 2010), where the platform interrupts the application at periodic events, called heartbeats, to measure its performance and power consumption and to compare them with the application's objectives. Maggio et al. (2011) studied and compared several decision strategies within the heartbeats framework to meet best applications' various objectives and requirements. The strategies under study included optimization heuristics, control theory based algorithms, and machine learning approaches. They found that "for systems with a broad or unknown range of target applications, adaptive control may provide the best general solution" (Maggio et al., 2011).

Scalability On the other hand, in the context of resource management for many-core systems, scalability is key as the resource manager needs

to manage a complex system with a large number of cores. To address scalability, there is growing interest towards modular resource management methods to offer scalable, autonomous, and coordinated resource management in many-core systems. For instance, Rahmani et al. (2018) recently presented SPECTR as a scalable resource management mechanism leveraging formal supervisory control theory (SCT) to combine the strengths of classical control theory with state-of-the-art heuristic approaches. SPECTR is a scalable and robust control architecture and a systematic design flow for hierarchical control of many-core systems. It leverages SCT techniques such as gain scheduling (Donyanavard et al., 2018) to allow autonomy for individual controllers while facilitating automatic synthesis of the high-level supervisory controller and its property verification.

Wealth of sensory data and control knobs From the actuation point of view, as more resource management techniques are integrated into SoCs, knobs to control these techniques become available to architecture, OS, or application levels. More control knobs and more sensory data opens the path to more complex and dynamic resource management strategies, a trend which is observable in the literature. A main challenge, due to variability and large sensory data sets, is to make sense of the data and predict future behavior. A number of recent works address these challenges with machine learning approaches (Martinez and Ipek, 2009; Bitirgen et al., 2008; Tesauro and et.al., 2007; Tesauro et al., 2006; Dutt et al., 2016; Wu and Louri, 2016; Singh and Rao, 2014; Blanton et al., 2015). For instance, Blanton et al. (2015) propose to build-in the capability for continuously learning key personality traits throughout the lifetime of the system to identify faults, wear-outs and performance deterioration, and Chen and Marculescu (2015b) use reinforcement learning to dynamically allocate processing resources to application threads.

Summary Based on current trends and the demand for adaptivity, we expect novel solutions to emerge that collect a wealth of sensory data, dynamically assess the meaning of these data, and make resource

allocation decisions by considering objectives with dynamically varying priorities. Specifically, machine learning techniques have been a promising trend for modeling the complexity of interaction among different on-chip resources and the corresponding effect on resource metrics (Gupta et al., 2018). Further, these techniques have targeted beyond the conventional fixed single and multi-objective allocation policies, towards dynamically varying goals (Shamsa et al., 2018, 2019). On this note, a general, hierarchical goal management scheme has recently been sketched by Rahmani et al. (2017b) for multi-core platforms and by Jantsch et al. (2018) for distributed IoT systems. However, many elements of such a strategy are still missing such as

- a formalism to define objectives in a flexible way and independent of the control and optimization technique that pursue the objectives;
- a reasonable hierarchy of goals that reflect the requirements of applications and the structure and interdependence of all relevant goals;
- strategies to dynamically rearranges the goal hierarchy and updates goal priorities.

The version of record is available at: http://dx.doi.org/10.1561/1000000055

8

Conclusions

From the early 1990s, when the first papers on dynamic on-chip resource management began to appear, the research on this topic has generated a steady flow of publications as illustrated in Figure 8.1. Although this review does not provide an exhaustive list of references, it shows the attention that the field has received during the last three decades.

However, as discussed in the previous chapter, the field has matured to the extent, that adding a new point technique that focuses on one metric for one subsystem is of limited value. Instead, we anticipate a new phase where progress is expected from holistic methods that take the entire SoC into account and cover all relevant metrics from performance to power to aging to Quality of Service (QoS). Already, we can observe nascent attempts for generic and comprehensive goal management that not only considers all metrics as part of the systems goals but also allows for dynamic adaptation of those goals depending on the evolving external situation and the internal state of the system. As the application demand on more adaptive systems keeps increasing we expect more holistic, dynamic management methods to be proposed and studied. This in turn will encourage the development of new point

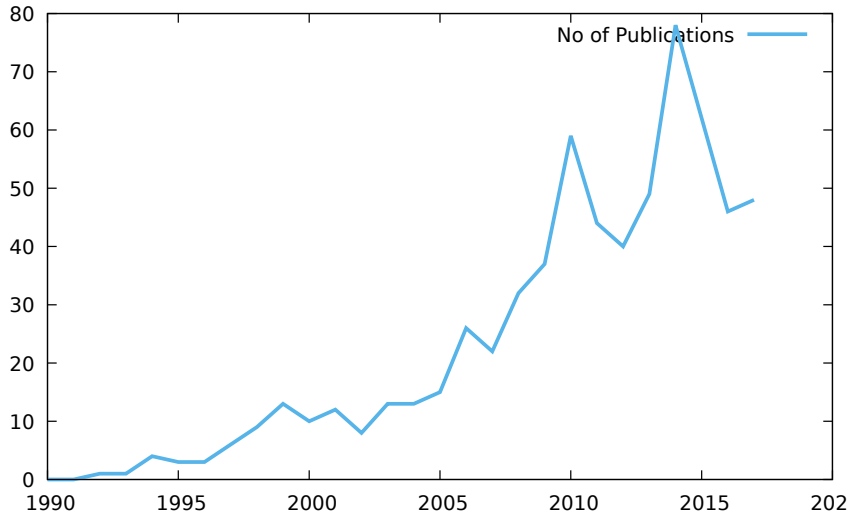

Figure 8.1: Number of publications per year cited in this review.

techniques and reconsideration of previously studied techniques in the light of such holistic methods.

Acknowledgment

We acknowledge financial support by the Marie Curie Actions of the European Union's H2020 Program, NSF Information Processing Factory grant (CCF-1704859), and Academy of Finland project ACTER (decision number 311304). This research was also partially funded by the European Union's Horizon 2020 Framework Program for Research and Innovation under grant agreement no 674875 (oCPS Marie Curie Network).

References

Acquaviva, A., L. Benini, and B. Riccó (2001), 'Energy Characterization of Embedded Real-time Operating Systems'. *SIGARCH Comput. Archit. News* pp. 13–18.

Agarwal, N., D. Nellans, M. Stephenson, M. O'Connor, and S. W. Keckler (2015), 'Page Placement Strategies for GPUs Within Heterogeneous Memory Systems'. *SIGPLAN Not.*

Aggarwal, N., P. Ranganathan, N. P. Jouppi, and J. E. Smith (2007), 'Configurable Isolation: Building High Availability Systems with Commodity Multi-core Processors'. In: *Proc. of Intl. Symp. on Computer Architecture (ISCA)*. pp. 470–481.

Ahn, J., S. Yoo, and K. Choi (2014), 'Dynamic power management of off-chip links for Hybrid Memory Cubes'. In: *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*. pp. 1–6.

Al Faruque, M. A., T. Ebi, and J. Henkel (2012), 'AdNoC: Runtime adaptive network-on-chip architecture'. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* pp. 257–269.

Annamalai, A., R. Rodrigues, I. Koren, and S. Kundu (2013), 'An Opportunistic Prediction-based Thread Scheduling to Maximize Throughput/Watt in AMPs'. In: *Proceedings of the 22Nd International Conference on Parallel Architectures and Compilation Techniques (PACT)*. pp. 63–72.

Apostolakis, A., D. Gizopoulos, M. Psarakis, and A. Paschalis (2009), 'Software-Based Self-Testing of Symmetric Shared-Memory Multiprocessors'. *IEEE Transactions on Computers* **58**(12), 1682–1694.

Ascia, G., V. Catania, M. Palesi, and D. Patti (2008), 'Implementation and analysis of a new selection strategy for adaptive routing in networks-on-chip'. *IEEE Transactions on Computers* pp. 809–820.

Attia, K. M., M. A. El-Hosseini, and H. A. Ali (2017), 'Dynamic power management techniques in multi-core architectures: A survey study'. *Ain Shams Engineering Journal* pp. 445–456.

Ausavarungnirun, R., K. K.-W. Chang, L. Subramanian, G. H. Loh, and O. Mutlu (2012), 'Staged Memory Scheduling: Achieving High Performance and Scalability in Heterogeneous Systems'. *SIGARCH Comput. Archit. News*.

Ayoub, R., K. R. Indukuri, and T. S. Rosing (2010), 'Energy efficient proactive thermal management in memory subsystem'. In: *2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*. pp. 195–200.

Azevedo, A., I. Issenin, R. Cornea, R. Gupta, N. Dutt, A. Veidenbaum, and A. Nicolau (2002), 'Profile-based dynamic voltage scheduling using program checkpoints'. In: *Proceedings 2002 Design, Automation and Test in Europe Conference and Exhibition*. pp. 168–175.

Badr, H. G. and S. Podar (1989), 'An optimal shortest-path routing policy for network computers with regular mesh-connected topologies'. *IEEE transactions on computers* pp. 1362–1371.

Baek, W. and T. M. Chilimbi (2010), 'Green : A Framework for Supporting Energy-Conscious Programming using Controlled Approximation'. In: *Proceedings of the ACM SIGPLAN conference on Programming language design and implementation -PLDI '10*. pp. 198–209.

Bakhoda, A., J. Kim, and T. M. Aamodt (2010), 'Throughput-Effective On-Chip Networks for Manycore Accelerators'. In: *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. pp. 421–432.

Baldassari, A., C. Bolchini, and A. Miele (2017), 'A dynamic reliability management framework for heterogeneous multicore systems'. In: *Proc. of Intl. Symp. on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. pp. 1–6.

Balfour, J. and W. J. Dally (2014), 'Design Tradeoffs for Tiled CMP On-chip Networks'. In: *ACM International Conference on Supercomputing 25th Anniversary Volume*. pp. 390–401.

Banerjee, A., P. T. Wolkotte, R. D. Mullins, S. W. Moore, and G. J. M. Smit (2009), 'An Energy and Performance Exploration of Network-on-Chip Architectures'. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* pp. 319–329.

Baynes, K., C. Collins, E. Fiterman, B. Ganesh, P. Kohout, C. Smit, T. Zhang, and B. Jacob (2003), 'The performance and energy consumption of embedded real-time operating systems'. *IEEE Transactions on Computers* pp. 1454–1469.

Beckmann, N. and D. Sanchez (2013), 'Jigsaw: Scalable Software-defined Caches'. In: *Proceedings of the 22Nd International Conference on Parallel Architectures and Compilation Techniques*. Piscataway, NJ, USA, pp. 213–224, IEEE Press.

Beckmann, N., P.-A. Tsai, and D. Sanchez (2015), 'Scaling distributed cache hierarchies through computation and data co-scheduling'. In: *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*. pp. 538–550.

Beloglazov, A. and R. Buyya (2010), 'Energy Efficient Resource Management in Virtualized Cloud Data Centers'. In: *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. pp. 826–831.

Bender, M. A., D. P. Bunde, E. D. Demaine, S. P. Fekete, V. J. Leung, H. Meijer, and C. A. Phillips (2008), 'Communication-aware processor allocation for supercomputers: Finding point sets of small average distance'. *Springer Algorithmica* pp. 279–298.

Benini, L., A. Bogliolo, G. A. Paleologo, and G. D. Micheli (1999), 'Policy optimization for dynamic power management'. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* pp. 813–833.

Benini, L. and G. De Micheli (2002), 'Networks on Chips: A New SoC Paradigm'. *Computer* pp. 70–78.

Besta, M., S. M. Hassan, S. Yalamanchili, R. Ausavarungnirun, O. Mutlu, and T. Hoefler (2018), 'Slim NoC: A Low-Diameter On-Chip Network Topology for High Energy Efficiency and Scalability'. In: *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. pp. 43–55.

Besta, M. and T. Hoefler (2014), 'Slim Fly: A Cost Effective Low-diameter Network Topology'. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. pp. 348–359.

Bhardwaj, K., K. Chakraborty, and S. Roy (2012), 'Towards graceful aging degradation in NoCs through an adaptive routing algorithm'. In: *Proc. of Design Automation Conf. (DAC)*. pp. 382–391.

Bhardwaj, S., W. Wang, R. Vattikonda, Y. Cao, and S. Vrudhula (2006), 'Predictive Modeling of the NBTI Effect for Reliable Design'. In: *Proc. of IEEE Custom Integrated Circuits Conf. (CICC)*. pp. 189–192.

Bitirgen, R., E. Ipek, and J. Martinez (2008), 'Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach'. In: *41st annual IEEE/ACM International Symposium on Microarchitecture*. pp. 318–329.

Bjerregaard, T. and S. Mahadevan (2006), 'A Survey of Research and Practices of Network-on-chip'. *ACM Computing Surveys* **38**(1).

Blanton, R. D., X. Li, K. Mai, D. Marculescu, R. Marculescu, J. Paramesh, J. Schneider, and D. E. Thomas (2015), 'Statistical Learning in Chip (SLIC)'. In: *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, ICCAD*. pp. 664–669.

Blome, J., S. Feng, S. Gupta, and S. Mahlke (2007), 'Self-calibrating Online Wearout Detection'. In: *Proc. of Intl. Symp. on Microarchitecture (MICRO)*. pp. 109–122.

Bogdan, P., R. Marculescu, and S. Jain (2013), 'Dynamic Power Management for Multidomain System-on-chip Platforms: An Optimal Control Approach'. *ACM Trans. Des. Autom. Electron. Syst.* pp. 46:1–46:20.

Bolchini, C., M. Carminati, and A. Miele (2013a), 'Self-Adaptive Fault Tolerance in Multi-/Many-Core Systems'. *Journal of Electronic Testing: Theory and Application* **29**(2), 159–175.

Bolchini, C., M. Carminati, A. Miele, A. Das, A. Kumar, and B. Veeravalli (2013b), 'Run-Time Mapping for Reliable Many-Cores Based on Energy/Performance Trade-offs'. In: *Proc. of Intl. Symp. on Defect and Fault Tolerance in VLSI and Nanotech. Systems (DFT)*. pp. 58–64.

Bolchini, C., L. Cassano, and A. Miele (2016), 'Lifetime-aware load distribution policies in multi-core systems: An in-depth analysis'. In: *Proc. of Conf. on Design, Automation & Test in Europe (DATE)*. pp. 804–809.

Bolchini, C., A. Miele, and D. Sciuto (2012), 'An adaptive approach for online fault management in many-core architectures'. In: *Proc. of Design, Automation Test in Europe Conf. Exhibition (DATE)*. pp. 1429–1432.

Bornholt, J., T. Mytkowicz, and K. S. McKinley (2014), 'Uncertain<T>: A First-Order Type for Uncertain Data'. In: *Proceedings of International conference on Architectural support for programming languages and operating systems - ASPLOS '14*. pp. 51–66.

Brooks, D. and M. Martonosi (2001), 'Dynamic thermal management for high-performance microprocessors'. In: *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*. pp. 171–182.

Bruestel, M. and A. Kumar (2017), 'Accounting for systematic errors in approximate computing'. In: *Proceedings of the Conference on Design, Automation & Test in Europe*. pp. 298–301.

Burns, A. and R. Davis (2013), 'Mixed criticality systems-a review'. *Department of Computer Science, University of York, Tech. Rep* pp. 1–69.

Burns, A. and R. I. Davis (2017), 'A Survey of Research into Mixed Criticality Systems'. *ACM Computing Surveys* **50**(6), 82:1–82:37.

Cai, E., D. Stamoulis, and D. Marculescu (2016), 'Exploring Aging Deceleration in FinFET-based Multi-core Systems'. In: *Proc of Intl. Conf. on Computer-Aided Design (ICCAD)*. pp. 111:1–111:8.

Carara, E. A. and F. G. Moraes (2010), 'Flow oriented routing for NOCS'. In: *SOC Conference (SOCC), 2010 IEEE International*. pp. 367–370.

Carbin, M., D. Kim, S. Misailovic, and M. C. Rinard (2012), 'Proving acceptability properties of relaxed nondeterministic approximate programs'. In: *Proceedings of {ACM} {SIGPLAN} Conference on Programming Language Design and Implementation - PLDI '12*, Vol. 47. pp. 169–180.

Carbin, M., S. Misailovic, M. Kling, and M. C. Rinard (2011), 'Detecting and escaping infinite loops with jolt'. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. 6813 LNCS. pp. 609–633.

Carbin, M., S. Misailovic, and M. C. Rinard (2013), 'Verifying quantitative reliability for programs that execute on unreliable hardware'. In: *Proceedings of Conference on Object-Oriented Programming Systems, Languages, and Applications - OOPSLA '13*. pp. 33–52.

Carbin, M. and M. C. M. Rinard (2010), 'Automatically identifying critical input regions and code in applications'. In: *Proceedings of international symposium on Software testing and analysis*. pp. 37–48.

Carvalho, E., N. Calazans, and F. Moraes (2007), 'Heuristics for dynamic task mapping in NoC-based heterogeneous MPSoCs'. In: *Proc. of IEEE/IFIP International Workshop on Rapid System Prototyping*. pp. 34–40.

Castrillon, J., A. Tretter, R. Leupers, and G. Ascheid (2012), 'Communication-aware mapping of KPN applications onto heterogeneous MPSoCs'. In: *Proceedings of the 49th Annual Design Automation Conference*. pp. 1266–1271.

Catania, V., R. Holsmark, S. Kumar, and M. Palesi (2006), 'A methodology for design of application specific deadlock-free routing algorithms for NoC systems'. In: *Hardware/Software Codesign and System Synthesis, 2006. CODES+ ISSS'06. Proceedings of the 4th International Conference*. pp. 142–147.

Ceratti, A., T. Copetti, L. Bolzani, and F. Vargas (2012), 'On-chip aging sensor to monitor NBTI effect in nano-scale SRAM'. In: *Proc. of Intl. Symp. on Design and Diagnostics of Electronic Circuits Systems (DDECS)*. pp. 354–359.

Chang, D.-W., H.-H. Chen, and W.-J. Su (2015a), 'VSSD: Performance Isolation in a Solid-State Drive'. *ACM Trans. Des. Autom. Electron. Syst.* pp. 51:1–51:33.

Chang, E.-J., H.-K. Hsin, C.-H. Chao, S.-Y. Lin, and A.-Y. A. Wu (2015b), 'Regional ACO-based cascaded adaptive routing for traffic balancing in mesh-based network-on-chip systems'. *IEEE Transactions on Computers* pp. 868–875.

Chang, E.-J., H.-K. Hsin, S.-Y. Lin, and A.-Y. Wu (2014), 'Path-congestion-aware adaptive routing with a contention prediction scheme for network-on-chip systems'. *IEEE Transactions on computer-aided design of Integrated circuits and systems* pp. 113–126.

Chang, K. K.-W., R. Ausavarungnirun, C. Fallin, and O. Mutlu (2012), 'HAT: Heterogeneous adaptive throttling for on-chip networks'. In: *2012 IEEE 24th International Symposium on Computer Architecture and High Performance Computing*. pp. 9–18.

Chantem, T., Y. Xiang, X. S. Hu, and R. P. Dick (2013), 'Enhancing Multicore Reliability through Wear Compensation in Online Assignment and Scheduling'. In: *Proc. of Conf. on Design, Automation & Test in Europe (DATE)*. pp. 1373–1378.

Chen, C. and A. Joshi (2013), 'Runtime Management of Laser Power in Silicon-Photonic Multibus NoC Architecture'. *IEEE Journal of Selected Topics in Quantum Electronics*.

Chen, C. O., S. Park, T. Krishna, S. Subramanian, A. P. Chandrakasan, and L. Peh (2013), 'SMART: A single-cycle reconfigurable NoC for SoC applications'. In: *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*.

Chen, F., D. A. Koufaty, and X. Zhang (2011), 'Hystor: Making the Best Use of Solid State Drives in High Performance Storage Systems'. In: *Proceedings of the International Conference on Supercomputing*. pp. 22–32.

Chen, G., F. Li, S. W. Son, and M. Kandemir (2008), 'Application mapping for chip multiprocessors'. In: *Proceedings of the 45th annual design automation conference*. pp. 620–625.

Chen, K. H., J. J. Chen, F. Kriebel, S. Rehman, M. Shafique, and J. Henkel (2016), 'Task Mapping for Redundant Multithreading in Multi-Cores with Reliability and Performance Heterogeneity'. *IEEE Transactions on Computers* **65**(11), 3441–3455.

Chen, Q. and M. Guo (2014), 'Adaptive Workload-aware Task Scheduling for single-ISA Asymmetric Multicore Architectures'. *ACM Trans. Archit. Code Optim.* pp. 8:1–8:25.

Chen, T., Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam (2014a), 'DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning'. In: *Proceedings of International conference on Architectural support for programming languages and operating systems*. pp. 269–284.

Chen, X., Y. Wang, Y. Liang, Y. Xie, and H. Yang (2014b), 'Run-time technique for simultaneous aging and power optimization in GPGPUs'. In: *Proc. of Design Automation Conf. (DAC)*. pp. 1–6.

Chen, Y., T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam (2014c), 'DaDianNao: A Machine-Learning Supercomputer'. In: *IEEE/ACM International Symposium on Microarchitecture*. pp. 609–622.

Chen, Y.-Y., E.-J. Chang, H.-K. Hsin, K.-C. J. Chen, and A.-Y. A. Wu (2017), 'Path-Diversity-Aware Fault-Tolerant Routing Algorithm for Network-on-Chip Systems'. *IEEE Transactions on Parallel and Distributed Systems* pp. 838–849.

Chen, Z. and D. Marculescu (2015a), 'Distributed Reinforcement Learning for Power Limited Many-core System Performance Optimization'. In: *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*.

Chen, Z. and D. Marculescu (2015b), 'Distributed Reinforcement Learning for Power Limited Many-core System Performance Optimization'. In: *Proceedings of the 2015 Design, Automation & Test in Europe Conference (DATE)*. pp. 1521–1526.

Chen, Z., M. Yang, G. Francia, and J. Dongarra (2007), 'Self Adaptive Application Level Fault Tolerance for Parallel and Distributed Computing'. In: *Proc. of Intl. Parallel and Distributed Processing Symp.- (IPDPS)*. pp. 1–8.

Cheng, S.-T., C.-M. Chen, and J.-W. Hwang (1997), 'Low-power design for real-time systems'. In: *Proceedings of ICICS, 1997 International Conference on Information, Communications and Signal Processing. Theme: Trends in Information Systems Engineering and Wireless Multimedia Communications (Cat.* pp. 1746–1750 vol.3.

Cho, S. and L. Jin (2006), 'Managing distributed, shared L2 caches through OS-level page allocation'. In: *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*. pp. 455–468.

Chou, C., U. Y. Ogras, and R. Marculescu (2008), 'Energy- and Performance-Aware Incremental Mapping for Networks on Chip With Multiple Voltage Levels'. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

Chou, C. L. and R. Marculescu (2011), 'FARM: Fault-aware resource management in NoC-based multiprocessor platforms'. In: *Proc. of Design, Automation Test in Europe Conf. (DATE)*. pp. 1–6.

Chou, C.-L., U. Y. Ogras, and R. Marculescu (2008), 'Energy-and performance-aware incremental mapping for networks on chip with multiple voltage levels'. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **27**(10), 1866–1879.

Chou, C. T., Y. P. Lin, K. Y. Chiang, and K. C. Chen (2017), 'Dynamic Buffer Allocation for thermal-aware 3D network-on-chip systems'. In: *2017 IEEE International Conference on Consumer Electronics - Taiwan (ICCE-TW)*. pp. 65–66.

Christoforakis, I., O. Tomoutzoglou, D. Bakoyiannis, and G. Kornaros (2015), 'Dithering-Based Power and Thermal Management on FPGA-Based Multi-core Embedded Systems'. In: *2015 IEEE 13th International Conference on Embedded and Ubiquitous Computing*. pp. 173–177.

Chung, E.-Y., L. Benini, A. Bogliolo, and G. D. Micheli (1999), 'Dynamic power management for nonstationary service requests'. In: *Design, Automation and Test in Europe Conference and Exhibition, 1999. Proceedings (Cat. No. PR00078)*. pp. 77–81.

Cochran, R., C. Hankendi, A. K. Coskun, and S. Reda (2011), 'Pack & Cap: Adaptive DVFS and Thread Packing Under Power Caps'. In: *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*.

Coskun, A. K., T. S. Rosing, and K. C. Gross (2009), 'Utilizing Predictors for Efficient Thermal Management in Multiprocessor SoCs'. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **28**(10), 1503–1516.

Coskun, A. K., T. S. Rosing, and K. Whisnant (2007), 'Temperature Aware Task Scheduling in MPSoCs'. In: *2007 Design, Automation Test in Europe Conference Exhibition*.

Coskun, A. K., R. Strong, D. M. Tullsen, and T. S. Rosing (2009), 'Evaluating the Impact of Job Scheduling and Power Management on Processor Lifetime for Chip Multiprocessors'. In: *Proc. of Intl. Conf. Measurement and Modeling of Computer Systems (SIGMETRICS)*. pp. 169–180.

Cui, Z., S. A. Mckee, Z. Zha, Y. Bao, and M. Chen (2014), 'DTail : A Flexible Approach to DRAM Refresh Management'. In: *Proceedings of International Conference on Supercomputing - SC '14*. pp. 43–52.

Dai, J., W. Ma, X. Jiang, and T. Watanabe (2017), 'Hybrid path-diversity-dominant output selection method for Network-on-Chip systems'. In: *SoC Design Conference (ISOCC), 2017 International*. pp. 125–126.

Dally, W. J. and H. Aoki (1993), 'Deadlock-free adaptive routing in multicomputer networks using virtual channels'. *IEEE transactions on Parallel and Distributed Systems* pp. 466–475.

Dally, W. J. and B. Towles (2001), 'Route Packets, Not Wires: On-chip Interconnection Networks'. In: *Proceedings of the 38th Annual Design Automation Conference (DAC)*. pp. 684–689.

Das, A., R. Shafik, G. V. Merrett, B. M. Al-Hashimi, A. Kumar, and B. Veeravalli (2014), 'Reinforcement Learning-Based Inter- and Intra-Application Thermal Optimization for Lifetime Improvement of Multicore Systems'. In: *Proc. of Design Automation Conf. (DAC)*. pp. 170:1–170:6.

Das, A., M. J. Walker, A. Hansson, B. M. Al-Hashimi, and G. V. Merrett (2015), 'Hardware-software interaction for run-time power optimization: A case study of embedded Linux on multicore smartphones'. In: *2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*.

David, R., P. Bogdan, R. Marculescu, and U. Ogras (2011), 'Dynamic power management of voltage-frequency island partitioned Networks-on-Chip using Intel's Single-chip Cloud Computer'. In: *Proceedings of the Fifth ACM/IEEE International Symposium*. pp. 257–258.

de Souza Carvalho, E. L., N. L. V. Calazans, and F. G. Moraes (2010), 'Dynamic task mapping for MPSoCs'. *IEEE Design & Test of Computers* pp. 26–35.

Delimitrou, C. and C. Kozyrakis (2013), 'Paragon: QoS-aware Scheduling for Heterogeneous Datacenters'. In: *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*. pp. 77–88.

Delimitrou, C. and C. Kozyrakis (2014), 'Quasar: Resource-efficient and QoS-aware Cluster Management'. In: *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. pp. 127–144.

Deo, N. and C.-Y. Pang (1984), 'Shortest-path algorithms: Taxonomy and annotation'. *Networks* pp. 275–323.

Diemer, J. and R. Ernst (2010), 'Back Suction: Service Guarantees for Latency-Sensitive On-chip Networks'. In: *Proceedings of the 2010 Fourth ACM/IEEE International Symposium on Networks-on-Chip (NOCS)*. pp. 155–162.

Diemer, J., R. Ernst, and M. Kauschke (2010), 'Efficient throughput-guarantees for latency-sensitive networks-on-chip'. In: *Design Automation Conference (ASP-DAC), 2010 15th Asia and South Pacific*. pp. 529–534.

Donald, J. and M. Martonosi (2005), 'Leveraging Simultaneous Multithreading for Adaptive Thermal Control'. In: *Proc. of the Second Workshop on Temperature-Aware Computer Systems*.

Donyanavard, B., A. M. Rahmani, T. Muck, K. Moazemmi, and N. Dutt (2018), 'Gain scheduled control for nonlinear power management in CMPs'. In: *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*. pp. 921–924.

Du, Z., R. Fasthuber, T. Chen, P. Ienne, L. Li, X. Feng, Y. Chen, and O. Temam (2015), 'ShiDianNao : Shifting Vision Processing Closer to the Sensor'. In: *Proceedings of IEEE/ACM International Symposium on Computer Architecture*.

Dutt, N., A. Jantsch, and S. Sarma (2016), 'Toward Smart Embedded Systems: A Self-aware System-on-Chip (SoC) Perspective'. *ACM Trans. Embed. Comput. Syst.* pp. 22:1–22:27.

Ebrahimi, E., C. J. Lee, O. Mutlu, and Y. N. Patt (2010), 'Fairness via Source Throttling: A Configurable and High-performance Fairness Substrate for Multi-core Memory Systems'. In: *Proceedings of the Fifteenth Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems*. pp. 335–346.

El-Harouni, W., S. Rehman, B. S. Prabakaran, A. Kumar, R. Hafiz, and M. Shafique (2017), 'Embracing approximate computing for energy-efficient motion estimation in high efficiency video coding'. In: *Proceedings of the Conference on Design, Automation & Test in Europe*. pp. 1388–1393.

Elyasi, N., M. Arjomand, A. Sivasubramaniam, M. T. Kandemir, C. R. Das, and M. Jung (2017), 'Exploiting Intra-Request Slack to Improve SSD Performance'. *SIGARCH Comput. Archit. News* pp. 375–388.

Esmaeilzadeh, H. (2015), 'Approximate Acceleration: A Path Through the Era of Dark Silicon and Big Data'. In: *Proceedings of the 2015 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*. Piscataway, NJ, USA, pp. 31–32, IEEE Press.

Esmaeilzadeh, H., A. Sampson, L. Ceze, and D. Burger (2012a), 'Architecture support for disciplined approximate programming'. In: *ACM SIGARCH Computer Architecture News*, Vol. 40(1). p. 301.

Esmaeilzadeh, H., A. Sampson, L. Ceze, and D. Burger (2012b), 'Neural Acceleration for General-Purpose Approximate Programs'. In: *Proceedings of IEEE/ACM International Symposium on Microarchitecture*. pp. 449–460, Ieee.

Eyerman, S. and L. Eeckhout (2008), 'System-Level Performance Metrics for Multiprogram Workloads'. *IEEE Micro* pp. 42–53.

Faruque, A., M. Abdullah, R. Krist, and J. Henkel (2008), 'ADAM: run-time agent-based distributed application mapping for on-chip communication'. In: *Proceedings of the 45th annual Design Automation Conference*. pp. 760–765.

Fattah, M., M. Daneshtalab, P. Liljeberg, and J. Plosila (2013), 'Smart hill climbing for agile dynamic mapping in many-core systems'. In: *Proc. of IEEE/ACM Design Automation Conference*.

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

Fattah, M., M. Ramirez, M. Daneshtalab, P. Liljeberg, and J. Plosila (2012), 'CoNA: Dynamic application mapping for congestion reduction in many-core systems'. In: *Proc. of IEEE International Conference on Computer Design*. pp. 364–370.

Feng, C., Z. Lu, A. Jantsch, J. Li, and M. Zhang (2010), 'A Reconfigurable Fault-tolerant Deflection Routing Algorithm Based on Reinforcement Learning for Networks-on-Chip'. In: *Proceedings of the International Workshop on Network on Chip Architectures (NoCArc)*.

Feng, C., Z. Lu, A. Jantsch, and M. Zhang (2012), 'A 1-cycle 1.25GHz Bufferless Router for 3D Network-on-Chip'. *IEICE Transactions on Information and Systems*.

Feng, W.-c. and K. G. Shin (1997), 'Impact of Selection Functions on Routing Algorithm Performance in Multicomputer Networks'. In: *Proceedings of the 11th International Conference on Supercomputing (ICS)*. pp. 132–139.

Foutris, N., M. Psarakis, D. Gizopoulos, A. Apostolakis, X. Vera, and A. Gonzalez (2010), 'MT-SBST: Self-test optimization in multithreaded multicore architectures'. In: *Proc. of IEEE Intl. Test Conf. (ITC)*. pp. 1–10.

Gaspar, F., A. Ilic, P. Tomás, and L. Sousa (2014), 'Performance-aware task management and frequency scaling in embedded systems'. In: *Computer Architecture and High Performance Computing (SBAC-PAD), 2014 IEEE 26th International Symposium on*. pp. 65–72.

Ge, R., X. Feng, S. Song, H. C. Chang, D. Li, and K. W. Cameron (2010), 'PowerPack: Energy Profiling and Analysis of High-Performance Systems and Applications'. *IEEE Transactions on Parallel and Distributed Systems* pp. 658–671.

Gebregiorgis, A., S. Kiamehr, and M. B. Tahoori (2017), 'Error Propagation Aware Timing Relaxation For Approximate Near Threshold Computing'. In: *Proceedings of the 54th Annual Design Automation Conference 2017*. New York, NY, USA, pp. 77:1–77:6, ACM.

Ghose, S., H. Lee, and J. F. Martínez (2013), 'Improving Memory Scheduling via Processor-side Load Criticality Information'. In: *Proceedings of the 40th Annual International Symposium on Computer Architecture*. pp. 84–95.

Glass, C. J. and L. M. Ni (1992), 'The Turn Model for Adaptive Routing'. In: *Proceedings of the 19th Annual International Symposium on Computer Architecture (ISCA)*. pp. 278–287.

Gnad, D., M. Shafique, F. Kriebel, S. Rehman, D. Sun, and J. Henkel (2015), 'Hayat: Harnessing Dark Silicon and Variability for Aging Deceleration and Balancing'. In: *Proc. of Design Automation Conf. (DAC)*. pp. 180:1–180:6.

Gratz, P., B. Grot, and S. W. Keckler (2008), 'Regional congestion awareness for load balance in networks-on-chip'. In: *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*. pp. 203–214.

Grigorian, B., N. Farahpour, and G. Reinman (2015), 'BRAINIAC: Bringing reliable accuracy into neurally-implemented approximate computing'. In: *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*. pp. 615–626.

Grot, B., J. Hestness, S. W. Keckler, and O. Mutlu (2009a), 'Express cube topologies for on-chip interconnects'. In: *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on*. pp. 163–174.

Grot, B., J. Hestness, S. W. Keckler, and O. Mutlu (2011), 'Kilo-NOC: A Heterogeneous Network-on-chip Architecture for Scalability and Service Guarantees'. In: *Proceedings of the 38th Annual International Symposium on Computer Architecture*. New York, NY, USA, pp. 401–412, ACM.

Grot, B., S. W. Keckler, and O. Mutlu (2009b), 'Preemptive Virtual Clock: A Flexible, Efficient, and Cost-effective QOS Scheme for Networks-on-chip'. In: *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture*. New York, NY, USA, pp. 268–279, ACM.

Guang, L., P. Liljeberg, E. Nigussie, and H. Tenhunen (2009), 'A review of dynamic power management methods in NoC under emerging design considerations'. In: *2009 NORCHIP*. pp. 1–6.

Guo, F., H. Kannan, L. Zhao, R. Illikkal, R. Iyer, D. Newell, Y. Solihin, and C. Kozyrakis (2007a), 'From chaos to QoS: case studies in CMP resource management'. *ACM SIGARCH Computer Architecture News* pp. 21–30.

Guo, F., Y. Solihin, L. Zhao, and R. Iyer (2007b), 'A framework for providing quality of service in chip multi-processors'. In: *Microarchitecture, 2007. MICRO 2007. 40th Annual IEEE/ACM International Symposium on*. pp. 343–355.

Gupta, A., Y. Kim, and B. Urgaonkar (2009), 'DFTL: A Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings'. *SIGPLAN Not.* pp. 229–240.

Gupta, S., S. Feng, A. Ansari, J. Blome, and S. Mahlke (2008), 'The StageNet fabric for constructing resilient multicore systems'. In: *Proc. of Intl. Symp. on Microarchitecture (MICRO)*. pp. 141–151.

Gupta, U., M. Babu, R. Ayoub, M. Kishinevsky, F. Paterna, and U. Y. Ogras (2018), 'STAFF: online learning with stabilized adaptive forgetting factor and feature selection algorithm'. In: *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. pp. 1–6.

Gupta, U., J. Campbell, U. Y. Ogras, R. Ayoub, M. Kishinevsky, F. Paterna, and S. Gumussoy (2016), 'Adaptive performance prediction for integrated GPUs'. In: *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. pp. 1–8.

Gurumurthi, S., A. Sivasubramaniam, M. Kandemir, and H. Franke (2003), 'DRPM: dynamic speed control for power management in server class disks'. In: *30th Annual International Symposium on Computer Architecture, 2003. Proceedings*. pp. 169–179.

Ha, C. Y., Y. X. Wang, and C. W. Chang (2017), 'Dynamic Power Management for wearable devices with Non-Volatile Memory'. In: *2017 International Conference on Applied System Innovation (ICASI)*. pp. 37–39.

Haghbayan, M. H., A. Kanduri, A. M. Rahmani, P. Liljeberg, A. Jantsch, and H. Tenhunen (2015), 'MapPro: Proactive Runtime Mapping for Dynamic Workloads by Quantifying Ripple Effect of Applications on Networks-on-Chip'. In: *Proc. of Intl. Symp. on Networks-on-Chip (NOCS)*. pp. 1–8.

Haghbayan, M. H., A. Miele, A. M. Rahmani, P. Liljeberg, and H. Tenhunen (2016a), 'A lifetime-aware runtime mapping approach for many-core systems in the dark silicon era'. In: *Proc. of Conf. on Design, Automation & Test in Europe (DATE)*. pp. 854–857.

Haghbayan, M. H., A. Miele, A. M. Rahmani, P. Liljeberg, and H. Tenhunen (2017), 'Performance/Reliability-Aware Resource Management for Many-Cores in Dark Silicon Era'. *IEEE Transactions on Computers* **66**(9), 1599–1612.

Haghbayan, M. H., A. M. Rahmani, A. Miele, M. Fattah, J. Plosila, P. Liljeberg, and H. Tenhunen (2016b), 'A Power-Aware Approach for Online Test Scheduling in Many-Core Architectures'. *IEEE Transactions on Computers* **65**(3), 730–743.

Hajimiri, H., M. A. Qathrady, and P. Mishra (2013), 'Proactive thermal management using memory based computing'. In: *2013 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*. pp. 110–115.

Hamers, J. and L. Eeckhout (2010), 'Scenario-Based Resource Prediction for QoS-Aware Media Processing'. *Computer* pp. 56–63.

Hari, S. K. S., M. L. Li, P. Ramachandran, B. Choi, and S. V. Adve (2009), 'mSWAT: Low-cost hardware fault detection and diagnosis for multicore systems'. In: *Proc. of Intl. Symp. on Microarchitecture (MICRO)*. pp. 122–132.

Hartman, A. S. and D. E. Thomas (2012), 'Lifetime improvement through runtime wear-based task mapping'. In: *Proc. of Intl. Conf. Hardware/software codesign and system synthesis (CODES)*. pp. 13–22.

Heißwolf, J., R. König, and J. Becker (2012), 'A scalable noc router design providing qos support using weighted round robin scheduling'. In: *Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on*. pp. 625–632.

Hemani, A., A. Jantsch, S. Kumar, A. Postula, J. Öberg, M. Millberg, and D. Lindqvist (2000), 'Network on Chip: An architecture for billion transistor era'. In: *Proceeding of the IEEE NorChip Conference*.

Herbert, S. and D. Marculescu (2007), 'Analysis of dynamic voltage/frequency scaling in chip-multiprocessors'. In: *Proceedings of the 2007 international symposium on Low power electronics and design (ISLPED '07)*.

Herdrich, A., E. Verplanke, P. Autee, R. Illikkal, C. Gianos, R. Singhal, and R. Iyer (2016), 'Cache QoS: From concept to reality in the Intel® Xeon® processor E5-2600 v3 product family'. In: *High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on*. pp. 657–668.

Hetherington, G., T. Fryars, N. Tamarapalli, M. Kassab, A. Hassan, and J. Rajski (1999), 'Logic BIST for large industrial designs: real issues and case studies'. In: *Proc. of Intl. Test Conf. (ITC)*. pp. 358–367.

Ho, N.-M., E. Manogaran, W.-F. Wong, and A. Anoosheh (2017), 'Efficient floating point precision tuning for approximate computing'. In: *Design Automation Conference (ASP-DAC), 2017 22nd Asia and South Pacific*. pp. 63–68.

Hoffmann, H., J. Eastep, M. D. Santambrogio, J. E. Miller, and A. Agarwal (2010), 'Application heartbeats for software performance and health'. *SIGPLAN Not.* pp. 347–348.

Hoffmann, H., S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. Rinard (2012), 'Dynamic knobs for responsive power-aware computing'. *ACM SIGPLAN Notices* **47**(4), 199.

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

Horowitz, M., T. Indermaur, and R. Gonzalez (1994), 'Low-power digital design'. In: *Proceedings of 1994 IEEE Symposium on Low Power Electronics*. pp. 8–11.

Howard, J., S. Dighe, Y. Hoskote, S. Vangal, D. Finan, G. Ruhl, D. Jenkins, H. Wilson, N. Borkar, G. Schrom, et al. (2010), 'A 48-core IA-32 message-passing processor with DVFS in 45nm CMOS'. In: *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*. pp. 108–109.

Hsu, C.-H. and W.-C. Feng (2005), 'A Power-Aware Run-Time System for High-Performance Computing'. In: *Proceedings of the ACM/IEEE Supercomputing Conference*.

Hu, J. and R. Marculescu (2003), 'Energy-aware mapping for tile-based NoC architectures under performance constraints'. In: *Proceedings of the 2003 Asia and South Pacific Design Automation Conference*. pp. 233–239.

Hu, J. and R. Marculescu (2003), 'Exploiting the routing flexibility for energy/performance aware mapping of regular NoC architectures'. In: *2003 Design, Automation and Test in Europe Conference and Exhibition*.

Hu, J. and R. Marculescu (2004), 'DyAD: smart routing for networks-on-chip'. In: *Proceedings of the 41st annual Design Automation Conference*. pp. 260–263.

Hu, J. and R. Marculescu (2005), 'Energy-and performance-aware mapping for regular NoC architectures'. *IEEE Transactions on computer-aided design of integrated circuits and systems* **24**(4), 551–562.

Huang, J., J. Lach, and G. Robins (2012), 'A Methodology for Energy-quality Tradeoff Using Imprecise Hardware'. In: *Proceedings of the 49th Annual Design Automation Conference*. New York, NY, USA, pp. 504–509, ACM.

Huang, K., L. Santinelli, J. J. Chen, L. Thiele, and G. C. Buttazzo (2009), 'Adaptive Dynamic Power Management for Hard Real-Time Systems'. In: *2009 30th IEEE Real-Time Systems Symposium*. pp. 23–32.

Huang, L. and Q. Xu (2010), 'Performance Yield-driven Task Allocation and Scheduling for MPSoCs Under Process Variation'. In: *Proceedings of the 47th Design Automation Conference*. pp. 326–331.

Huang, L., R. Ye, and Q. Xu (2011), 'Customer-aware task allocation and scheduling for multi-mode MPSoCs'. In: *Proc. of Design Automation Conf. (DAC)*. pp. 387–392.

Hughes, C. J., J. Srinivasan, and S. V. Adve (2001), 'Saving energy with architectural and frequency adaptations for multimedia applications'. In: *Proceedings. 34th ACM/IEEE International Symposium on Microarchitecture. MICRO-34*. pp. 250–261.

Hwang, W., S. Yoo, H. Ko, and B. Park (2010), 'An area efficient temperature sensor with software calibration for mobile application'. In: *2010 International SoC Design Conference*. pp. 349–352.

Isci, C., A. Buyuktosunoglu, C. y. Cher, P. Bose, and M. Martonosi (2006a), 'An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget'. In: *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*. pp. 347–358.

Isci, C., G. Contreras, and M. Martonosi (2006b), 'Live, Runtime Phase Monitoring and Prediction on Real Systems with Application to Dynamic Power Management'. In: *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*. pp. 359–370.

Ishihara, T. and H. Yasuura (1998), 'Voltage scheduling problem for dynamically variable voltage processors'. In: *Proceedings. 1998 International Symposium on Low Power Electronics and Design (IEEE Cat. No.98TH8379)*. pp. 197–202.

Iyer, R., L. Zhao, F. Guo, R. Illikkal, S. Makineni, D. Newell, Y. Solihin, L. Hsu, and S. Reinhardt (2007), 'QoS Policies and Architecture for Cache/Memory in CMP Platforms'. In: *Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*. pp. 25–36.

Jantsch, A., A. Anzanpour, H. Kolerdi, I. Azimi, L. C. Siafara, A. M. Rahmani, N. TaheriNejad, P. Liljeberg, and N. Dutt (2018), 'Hierarchical Dynamic Goal Management for IoT Systems'. In: *Proceedings of the IEEE International Symposium on Quality Electronic Design (ISQED 2018)*. USA.

Jantsch, A., N. Dutt, and A. M. Rahmani (2017), 'Self-Awareness in Systems on Chip – A Survey'. *IEEE Design Test* **34**(6), 1–19.

JEDEC Solid State Tech. Ass. (2010), 'Failure mechanisms and models for semiconductor devices'. *JEDEC Publication JEP122G*.

Jennings, B. and R. Stadler (2015), 'Resource Management in Clouds: Survey and Research Challenges'. *Journal of Network and Systems Management* **23**(3), 567–619.

Joao, J. A., M. A. Suleman, O. Mutlu, and Y. N. Patt (2012), 'Bottleneck Identification and Scheduling in Multithreaded Applications'. In: *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. pp. 223–234.

Joao, J. A., M. A. Suleman, O. Mutlu, and Y. N. Patt (2013), 'Utility-based Acceleration of Multithreaded Applications on Asymmetric CMPs'. In: *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA)*. pp. 154–165.

Jung, H. and M. Pedram (2006), 'Stochastic Dynamic Thermal Management: A Markovian Decision-based Approach'. In: *2006 International Conference on Computer Design*. pp. 452–457.

Kadjo, D., U. Ogras, R. Ayoub, M. Kishinevsky, and P. Gratz (2014), 'Towards platform level power management in mobile systems'. In: *2014 27th IEEE International System-on-Chip Conference (SOCC)*. pp. 146–151.

Kaggle Inc. (2017), 'The State of Data Science and Machine Learning'. https://www.kaggle.com/surveys/2017. Accessed: 2018-08-08.

Kaliorakis, M., M. Psarakis, N. Foutris, and D. Gizopoulos (2014), 'Accelerated online error detection in many-core microprocessor architectures'. In: *Proc. of IEEE VLSI Test Symp. (VTS)*. pp. 1–6.

Kanduri, A., M.-H. Haghbayan, A. M. Rahmani, P. Liljeberg, A. Jantsch, N. Dutt, and H. Tenhunen (2016), 'Approximation Knob: Power Capping Meets Energy Efficiency'. In: *Proceedings of the 35th International Conference on Computer-Aided Design*. New York, NY, USA, pp. 122:1–122:8, ACM.

Kanduri, A., M. H. Haghbayan, A. M. Rahmani, P. Liljeberg, A. Jantsch, and H. Tenhunen (2015), 'Dark silicon aware runtime mapping for many-core systems: A patterning approach'. In: *2015 33rd IEEE International Conference on Computer Design (ICCD)*.

Kanduri, A., M. H. Haghbayan, A. M. Rahmani, M. Shafique, A. Jantsch, and P. Liljeberg (2018a), 'adBoost: Thermal Aware Performance Boosting through Dark Silicon Patterning'. *IEEE Transactions on Computers*.

Kanduri, A., A. Miele, A. M. Rahmani, P. Liljeberg, C. Bolchini, and N. Dutt (2018b), 'Approximation-Aware Coordinated Power/Performance Management for Heterogeneous Multi-cores'. In: *Proceedings of the 55th Annual Design Automation Conference*. p. 39.

Kao, Y.-H., M. Yang, N. S. Artan, and H. J. Chao (2011), 'CNoC: high-radix clos network-on-chip'. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* pp. 1897–1910.

Kapadia, N. and S. Pasricha (2015), 'VARSHA: Variation and reliability-aware application scheduling with adaptive parallelism in the dark-silicon era'. In: *Proc. of Design, Automation Test in Europe Conf. Exhibition (DATE)*. pp. 1060–1065.

Karl, E., D. Blaauw, D. Sylvester, and T. Mudge (2008), 'Multi-Mechanism Reliability Modeling and Management in Dynamic Systems'. *IEEE Transactions on VLSI Systems* **16**(4), 476–487.

Karpuzcu, U. R., B. Greskamp, and J. Torrellas (2009), 'The BubbleWrap Many-core: Popping Cores for Sequential Acceleration'. In: *Proc. of Intl. Symp. on Microarchitecture (MICRO)*. pp. 447–458.

Kasture, H. and D. Sanchez (2014), 'Ubik: Efficient Cache Sharing with Strict Qos for Latency-critical Workloads'. In: *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. pp. 729–742.

Khoshavi, N., R. A. Ashraf, R. F. DeMara, S. Kiamehr, F. Oboril, and M. B. Tahoori (2017), 'Contemporary CMOS aging mitigation techniques: Survey, taxonomy, and methods'. *Integration, the VLSI Journal* **59**, 10 – 22.

Khudia, D. S., B. Zamirai, M. Samadi, and S. Mahlke (2015), 'Rumba: An online quality management system for approximate computing'. In: *Computer Architecture (ISCA), 2015 ACM/IEEE 42nd Annual International Symposium on*. pp. 554–566.

Kim, J., J. Balfour, and W. Dally (2007), 'Flattened butterfly topology for on-chip networks'. In: *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*. pp. 172–182.

Kim, J., W. J. Dally, S. Scott, and D. Abts (2008), 'Technology-Driven, Highly-Scalable Dragonfly Topology'. In: *Proceedings of the 35th Annual International Symposium on Computer Architecture (ISCA)*. pp. 77–88.

Kim, R. G., W. Choi, Z. Chen, J. R. Doppa, P. P. Pande, D. Marculescu, and R. Marculescu (2017), 'Imitation Learning for Dynamic VFI Control in Large-Scale Manycore Systems'. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*.

Kim, S., D. Chandra, and Y. Solihin (2004), 'Fair cache sharing and partitioning in a chip multiprocessor architecture'. In: *Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques*. pp. 111–122.

Kim, T., X. Huang, H. B. Chen, V. Sukharev, and S. X.-D. Tan (2016), 'Learning-based dynamic reliability management for dark silicon processor considering EM effects'. In: *Proc. of Conf. on Design, Automation & Test in Europe (DATE)*. pp. 463–468.

Kim, Y. (2010), 'ATLAS: A scalable and high-performance scheduling algorithm for multiple memory controllers'. In: *High Performance Computer Architecture (HPCA)*.

Kim, Y., M. Papamichael, O. Mutlu, and M. Harchol-Balter (2010), 'Thread Cluster Memory Scheduling: Exploiting Differences in Memory Access Behavior'. In: *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*. pp. 65–76.

Kirovski, D. and M. Potkonjak (1997), 'System-level Synthesis of Low-power Hard Real-time Systems'. In: *Proceedings of the 34th Annual Design Automation Conference*. pp. 697–702.

Kling, M., S. Misailovic, M. Carbin, and M. Rinard (2012), 'Bolt: on-demand infinite loop escape in unmodified binaries'. *Proceedings of the ACM ...* pp. 431–450.

Kong, J., S. W. Chung, and K. Skadron (2012), 'Recent Thermal Management Techniques for Microprocessors'. *ACM Computing Surveys* **44**(3), 13:1–13:42.

Koren, I. and C. M. Krishna (2007), *Fault-Tolerant Systems*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1st edition.

Koufaty, D., D. Reddy, and S. Hahn (2010), 'Bias Scheduling in Heterogeneous Multi-core Architectures'. In: *Proceedings of the 5th European Conference on Computer Systems*. pp. 125–138.

Krishna, C. M. and Y. H. Lee (2000), 'Voltage-clock-scaling adaptive scheduling techniques for low power in hard real-time systems'. In: *Proceedings Sixth IEEE Real-Time Technology and Applications Symposium. RTAS 2000*. pp. 156–165.

Kulkarni, C., F. Catthoor, and H. D. Man (1998), 'Code transformations for low power caching in embedded multimedia processors'. In: *Proceedings of the First Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing*. pp. 292–297.

Kumar, R., D. M. Tullsen, N. P. Jouppi, and P. Ranganathan (2005), 'Heterogeneous Chip Multiprocessors'. *Computer* pp. 32–38.

Lackey, D. E., P. S. Zuchowski, T. R. Bednar, D. W. Stout, S. W. Gould, and J. M. Cohn (2002), 'Managing power and performance for system-on-chip designs using Voltage Islands'. In: *IEEE/ACM International Conference on Computer Aided Design, 2002. ICCAD 2002.*

LaFrieda, C., E. Ipek, J. F. Martinez, and R. Manohar (2007), 'Utilizing Dynamically Coupled Cores to Form a Resilient Chip Multiprocessor'. In: *Proc. of Intl. Confl. on Dependable Systems and Networks (DSN)*. pp. 317–326.

Laurenzano, M. A., P. Hill, M. Samadi, S. Mahlke, J. Mars, and L. Tang (2016), 'Input Responsiveness: Using Canary Inputs to Dynamically Steer Approximation'. In: *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*. New York, NY, USA, pp. 161–176, ACM.

Lee, H., M. Shafique, and M. A. A. Faruque (2018), 'Aging-aware Workload Management on Embedded GPU Under Process Variation'. *IEEE Transactions on Computers* **67**(7), 920–933.

Lee, J. W., M. C. Ng, and K. Asanovic (2008), 'Globally-Synchronized Frames for Guaranteed Quality-of-Service in On-Chip Networks'. In: *Proceedings of the 35th Annual International Symposium on Computer Architecture*. pp. 89–100.

Lee, S., K. Kang, and C. M. Kyung (2015), 'Runtime Thermal Management for 3-D Chip-Multiprocessors With Hybrid SRAM/MRAM L2 Cache'. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* pp. 520–533.

Lee, S. and T. Sakurai (2000), 'Run-time voltage hopping for low-power real-time systems'. In: *Proceedings 37th Design Automation Conference*. pp. 806–809.

Li, B. and K. Nahrstedt (1999), 'A control-based middleware framework for quality-of-service adaptations'. *IEEE journal on selected areas in communications* pp. 1632–1650.

Li, B., L.-S. Peh, L. Zhao, and R. Iyer (2012), 'Dynamic QoS Management for Chip Multiprocessors'. *ACM Trans. Archit. Code Optim.* pp. 17:1–17:29.

Li, B., L. Zhao, R. Iyer, L.-S. Peh, M. Leddige, M. Espig, S. E. Lee, and D. Newell (2011), 'CoQoS: Coordinating QoS-aware shared resources in NoC-based SoCs'. *Journal of Parallel and Distributed Computing* pp. 700–713.

Li, C., W. Luo, S. S. Sapatnekar, and J. Hu (2015), 'Joint Precision Optimization and High Level Synthesis for Approximate Computing'. In: *Proceedings of Design Automation Conference - DAC '15*. New York, NY, USA, pp. 104:1–104:6, ACM.

Li, M., Q.-A. Zeng, and W.-B. Jone (2006), 'DyXY: A Proximity Congestion-aware Deadlock-free Dynamic Routing Method for Network on Chip'. In: *Proceedings of the 43rd Annual Design Automation Conference (DAC)*. pp. 849–852.

Li, Y., S. Ghose, J. Choi, J. Sun, H. Wang, and O. Mutlu (2017), 'Utility-Based Hybrid Memory Management'. In: *2017 IEEE International Conference on Cluster Computing (CLUSTER)*.

Li, Y. and W. Wolf (1997), 'A Task-level Hierarchical Memory Model for System Synthesis of Multiprocessors'. In: *Proceedings of the 34th Annual Design Automation Conference*. pp. 153–156.

Liao, W., L. He, and K. M. Lepak (2005), 'Temperature and supply Voltage aware performance and power modeling at microarchitecture level'. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* pp. 1042–1053.

Liu, D., T. Chen, S. Liu, J. Zhou, S. Zhou, O. Teman, X. Feng, Z. Xuehai, and Y. Chen (2015), 'PuDianNao : A Polyvalent Machine Learning Accelerator'. In: *Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems*. pp. 369–381.

Liu, J., P. H. Chou, N. Bagherzadeh, and F. Kurdahi (2001), 'Power-aware scheduling under timing constraints for mission-critical embedded systems'. In: *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*. pp. 840–845.

Liu, Y. and H. Zhu (2010), 'A survey of the research on power management techniques for high-performance systems'. *Software: Practice and Experience* **40**(11), 943–964.

Lo, D., L. Cheng, R. Govindaraju, L. A. Barroso, and C. Kozyrakis (2014), 'Towards Energy Proportionality for Large-scale Latency-critical Workloads'. In: *Proceeding of the 41st Annual International Symposium on Computer Architecuture (ISCA)*. pp. 301–312.

Lo, D., L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis (2015), 'Heracles: Improving Resource Efficiency at Scale'. In: *Proceedings of the 42Nd Annual International Symposium on Computer Architecture (ISCA)*. pp. 450–462.

Lo, W. H., K. z. Liang, and T. Hwang (2016), 'Thermal-aware dynamic page allocation policy by future access patterns for Hybrid Memory Cube (HMC)'. In: *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*. pp. 1084–1089.

Lorch, J. R. and A. J. Smith (2004), 'PACE: a new approach to dynamic voltage scaling'. *IEEE Transactions on Computers* pp. 856–869.

Luo, J. and N. K. Jha (2001), 'Battery-aware static scheduling for distributed real-time embedded systems'. In: *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*. pp. 444–449.

Lysne, O., T. Skeie, S.-A. Reinemo, and I. Theiss (2006), 'Layered routing in irregular networks'. *IEEE Transactions on Parallel and Distributed Systems* **17**(1), 51–65.

Ma, K. and X. Wang (2012), 'PGCapping: Exploiting Power Gating for Power Capping and Core Lifetime Balancing in CMPs'. In: *Proc. of Intl. Conf. on Parallel Architectures and Compilation Techniques (PACT)*. pp. 13–22.

Ma, S., N. Enright Jerger, and Z. Wang (2011), 'DBAR: An Efficient Routing Algorithm to Support Multiple Concurrent Applications in Networks-on-chip'. In: *Proceedings of the 38th Annual International Symposium on Computer Architecture (ISCA)*. pp. 413–424.

Ma, S., N. E. Jerger, Z. Wang, M. Lai, and L. Huang (2014), 'Holistic routing algorithm design to support workload consolidation in NoCs'. *IEEE Transactions on Computers* pp. 529–542.

Ma, T. C. L. and K. G. Shin (2000), 'A user-customizable energy-adaptive combined static/dynamic scheduler for mobile applications'. In: *Proceedings 21st IEEE Real-Time Systems Symposium*. pp. 227–236.

Ma, Y., T. Chantem, R. P. Dick, and X. S. Hu (2017a), 'Improving System-Level Lifetime Reliability of Multicore Soft Real-Time Systems'. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* **25**(6), 1895–1905.

Ma, Y., T. Chantem, R. P. Dick, S. Wang, and X. S. Hu (2017b), 'An on-line framework for improving reliability of real-time systems on "big-little" type MPSoCs'. In: *Proc. of Design, Automation Test in Europe Conf. Exhibition (DATE)*. pp. 446–451.

Maggio, M., H. Hoffmann, M. D. Santambrogio, A. Agarwal, and A. Leva (2011), 'Decision Making in Autonomic Computing Systems: Comparison of Approaches and Techniques'. In: *Proceedings of the 8th ACM International Conference on Autonomic Computing*. pp. 201–204.

Mahajan, D., A. Yazdanbakhsh, J. Park, B. Thwaites, and H. Esmaeilzadeh (2016), 'Towards Statistical Guarantees in Controlling Quality Tradeoffs for Approximate Acceleration'. In: *Proceedings of the 43rd International Symposium on Computer Architecture*. Piscataway, NJ, USA, pp. 66–77, IEEE Press.

Mahmood, A. and E. J. McCluskey (1988), 'Concurrent error detection using watchdog processors-a survey'. *IEEE Transactions on Computers* **37**(2), 160–174.

Maiterth, M., G. Koenig, K. Pedretti, S. Jana, N. Bates, A. Borghesi, D. Montoya, A. Bartolini, and M. Puzovic (2018), 'Energy and Power Aware Job Scheduling and Resource Management: Global Survey — Initial Analysis'. In: *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. pp. 685–693.

Mak, T., P. Y. Cheung, K.-P. Lam, and W. Luk (2011), 'Adaptive routing in network-on-chips using a dynamic-programming network'. *IEEE Transactions on industrial electronics* pp. 3701–3716.

Manzak, A. and C. Chakrabarti (2000), 'Variable voltage task scheduling for minimizing energy or minimizing power'. In: *2000 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.00CH37100)*. pp. 3239–3242 vol.6.

Manzak, A. and C. Chakrabarti (2001), 'Variable voltage task scheduling algorithms for minimizing energy'. In: *Low Power Electronics and Design, International Symposium on, 2001.* pp. 279–282.

Marculescu, R., U. Y. Ogras, L. Peh, N. E. Jerger, and Y. Hoskote (2009), 'Outstanding Research Problems in NoC Design: System, Microarchitecture, and Circuit Perspectives'. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

Mars, J., L. Tang, R. Hundt, K. Skadron, and M. L. Soffa (2011), 'Bubble-Up: Increasing Utilization in Modern Warehouse Scale Computers via Sensible Co-locations'. In: *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. pp. 248–259.

Martinez, J. and E. Ipek (2009), 'Dynamic multicore resource management: A machine learning approach'. *IEEE Micro, 29:8–17*.

Meena, J. S., S. M. Sze, U. Chand, and T.-Y. Tseng (2014), 'Overview of emerging nonvolatile memory technologies'. *Nanoscale Research Letters* **9**(526).

Mercati, P., F. Paterna, A. Bartolini, L. Benini, and T. S. Rosing (2017), 'WARM: Workload-Aware Reliability Management in Linux/Android'. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **36**(9), 1557–1570.

Meterelliyoz, M., H. Mahmoodi, and K. Roy (2005), 'A leakage control system for thermal stability during burn-in test'. In: *ITC*.

Millberg, M., E. Nilsson, R. Thid, and A. Jantsch (2004), 'Guaranteed Bandwidth using Looped Containers in Temporally Disjoint Networks within the Nostrum Network on Chip'. In: *Proceedings of the Design Automation and Test Europe Conference (DATE)*.

Misailovic, S., S. Sidiroglou, H. Hoffmann, and M. Rinard (2010), 'Quality of service profiling'. In: *ACM/IEEE 32nd International Conference on Software Engineering*, Vol. 1. pp. 25–34.

Mishra, N., H. Zhang, J. D. Lafferty, and H. Hoffmann (2015), 'A Probabilistic Graphical Model-based Approach for Minimizing Energy Under Performance Constraints'. In: *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*.

Mitra, S., M. Das, A. Banerjee, K. Datta, and T.-Y. Ho (2016), 'A Verification Guided Approach for Selective Program Transformations for Approximate Computing'. In: *Asian Test Symposium (ATS), 2016 IEEE 25th*. pp. 37–42.

Mittal, S. (2016), 'A Survey of Techniques for Approximate Computing'. *ACM Comput. Surv.* **48**(4), 62:1–62:33.

Moazzemi, K., C. Y. Hsieh, and N. Dutt (2016), 'HAMEX: heterogeneous architecture and memory exploration framework'. In: *2016 International Symposium on Rapid System Prototyping (RSP)*.

Moreau, T., F. Augusto, P. Howe, A. Alaghi, and L. Ceze (2017), 'Exploiting quality-energy tradeoffs with arbitrary quantization: special session paper'. In: *Proceedings of the Twelfth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis Companion*. p. 30.

Moreau, T., W. M, J. Nelson, A. Sampson, H. Esmaeilzadeh, L. Ceze, and M. Oskin (2015), 'SNNAP : Approximate Computing on Programmable SoCs via Neural Acceleration'. In: *International Symposium on High-Performance Computer Architecture (HPCA)*.

Moscibroda, T. and B. G. Zorn (2011), 'Flikker : Saving DRAM Refresh-power through Critical Data Partitioning'. In: *Proceedings of International conference on Architectural support for programming languages and operating systems - ASPLOS '11*. pp. 213–224.

Muck, T. R., B. Donyanavard, K. Moazzemi, A. M. Rahmani, A. Jantsch, and N. D. Dutt (2018), 'Design Methodology for Responsive and Robust MIMO Control of Heterogeneous Multicores'. *IEEE Transactions on Multi-Scale Computing Systems*.

Mück, T. R., Z. Ghaderi, N. D. Dutt, and E. Bozorgzadeh (2017), 'Exploiting Heterogeneity for Aging-Aware Load Balancing in Mobile Platforms'. *IEEE Transactions on Multi-Scale Computing Systems* **3**(1), 25–35.

Mukherjee, S. S., M. Kontz, and S. K. Reinhardt (2002), 'Detailed design and evaluation of redundant multi-threading alternatives'. In: *Proc. of Intl. Symp. on Computer Architecture (ISCA)*. pp. 99–110.

Murali, S., G. De Micheli, G. De Micheli, and G. De Micheli (2004), 'SUNMAP: a tool for automatic topology selection and generation for NoCs'. In: *Proceedings of the 41st annual Design Automation Conference*. pp. 914–919.

Muralidhara, S. P., L. Subramanian, O. Mutlu, M. Kandemir, and T. Moscibroda (2011), 'Reducing Memory Interference in Multicore Systems via Application-aware Memory Channel Partitioning'. In: *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*. pp. 374–385.

Murray, J., R. Kim, P. Wettin, P. P. Pande, and B. Shirazi (2014), 'Performance Evaluation of Congestion-Aware Routing with DVFS on a Millimeter-Wave Small-World Wireless NoC'. *J. Emerg. Technol. Comput. Syst.* **11**(2).

Mushtaq, H., Z. Al-Ars, and K. Bertels (2011), 'Survey of fault tolerance techniques for shared memory multicore/multiprocessor systems'. In: *Proc. of Intl. Design and Test Workshop (IDT)*. pp. 12–17.

Nair, R. (2015), 'Big data needs approximate computing: technical perspective'. *Communications of the ACM* **58**(1), 104–104.

Naithani, A., S. Eyerman, and L. Eeckhout (2017), 'Reliability-Aware Scheduling on Heterogeneous Multicore Processors'. In: *Proc. of IEEE Intl. Symp. on High Performance Computer Architecture (HPCA)*. pp. 397–408.

Naithani, A., S. Eyerman, and L. Eeckhout (2018), 'Optimizing Soft Error Reliability Through Scheduling on Heterogeneous Multicore Processors'. *IEEE Transactions on Computers* **67**(6), 830–846.

Nakai, M., S. Akui, K. Seno, T. Meguro, T. Seki, T. Kondo, A. Hashiguchi, H. Kawahara, K. Kumano, and M. Shimura (2005), 'Dynamic voltage and frequency management for a low-power embedded microprocessor'. *IEEE Journal of Solid-State Circuits* pp. 28–35.

Navada, S., N. K. Choudhary, S. V. Wadhavkar, and E. Rotenberg (2013), 'A Unified View of Non-monotonic Core Selection and Application Steering in Heterogeneous Chip Multiprocessors'. In: *Proceedings of the 22Nd International Conference on Parallel Architectures and Compilation Techniques (PACT)*. pp. 133–144.

Nazari, A., N. Sehatbakhsh, M. Alam, A. Zajic, and M. Prvulovic (2017), 'EDDIE: EM-based detection of deviations in program execution'. In: *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*.

Neuwirth, S., F. Wang, S. Oral, and U. Bruening (2017), 'Automatic and Transparent Resource Contention Mitigation for Improving Large-Scale Parallel File System Performance'. In: *2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS)*. pp. 604–613.

Neuwirth, S., F. Wang, S. Oral, S. Vazhkudai, J. Rogers, and U. Bruening (2016), 'Using Balanced Data Placement to Address I/O Contention in Production Environments'. In: *2016 28th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. pp. 9–17.

Nielsen, L. S., C. Niessen, J. Sparso, and K. van Berkel (1994), 'Low-power operation using self-timed circuits and adaptive scaling of the supply voltage'. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* pp. 391–397.

Nilsson, E., M. Millberg, J. Öberg, and A. Jantsch (2003), 'Load distribution with the Proximity Congestion Awareness in a Network on Chip'. In: *Proceedings of the Design Automation and Test Europe (DATE)*. pp. 1126–1127.

Ogras, U. Y. and R. Marculescu (2008), 'Analysis and Optimization of Prediction-based Flow Control in Networks-on-chip'. *ACM Trans. Des. Autom. Electron. Syst.* **13**(1), 11:1–11:28.

Ogras, U. Y., R. Marculescu, and D. Marculescu (2008), 'Variation-adaptive feedback control for networks-on-chip with multiple clock domains'. In: *2008 45th ACM/IEEE Design Automation Conference*.

Ogras, U. Y., R. Marculescu, D. Marculescu, and E. G. Jung (2009), 'Design and Management of Voltage-Frequency Island Partitioned Networks-on-Chip'. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*.

Ouyang, J. and Y. Xie (2010), 'LOFT: A high performance network-on-chip providing quality-of-service support'. In: *Microarchitecture (MICRO), 2010 43rd Annual IEEE/ACM International Symposium on*. pp. 409–420.

Palomino, D., M. Shafique, A. Susin, and J. Henkel (2016), 'Thermal optimization using adaptive approximate computing for video coding'. In: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016*. pp. 1207–1212.

Papazoglou, M. P. and D. Georgakopoulos (2003), 'Introduction: Service-oriented Computing'. *Commun. ACM* pp. 24–28.

Park, J., E. Amaro, D. Mahajan, B. Thwaites, and H. Esmaeilzadeh (2016), 'AxGames: Towards Crowdsourcing Quality Target Determination in Approximate Computing'. In: *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*. New York, NY, USA, pp. 623–636, ACM.

Parloff, R. (2016), 'WHY DEEP LEARNING IS SUDDENLY CHANGING YOUR LIFE'. *Fortune*.

Passos, R. M., J. A. Nacif, R. A. F. Mini, A. A. F. Loureiro, A. O. Fernandes, and C. N. Coelho (2006), 'System-level Dynamic Power Management Techniques for Communication Intensive Devices'. In: *2006 IFIP International Conference on Very Large Scale Integration*. pp. 373–378.

Pathania, A., Q. Jiao, A. Prakash, and T. Mitra (2014), 'Integrated CPU-GPU Power Management for 3D Mobile Games'. In: *Proceedings of the 51st Annual Design Automation Conference*. New York, NY, USA, pp. 40:1–40:6, ACM.

Petrucci, V., M. A. Laurenzano, J. Doherty, Y. Zhang, D. Mosse, J. Mars, and L. Tang (2015), 'Octopus-man: Qos-driven task management for heterogeneous multicores in warehouse-scale computers'. In: *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*. pp. 246–258.

Pillai, P. and K. G. Shin (2001), 'Real-time Dynamic Voltage Scaling for Low-power Embedded Operating Systems'. In: *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*. pp. 89–102.

Pritchett, T. and M. Thottethodi (2010), 'SieveStore: A Highly-selective, Ensemble-level Disk Cache for Cost-performance'. *SIGARCH Comput. Archit. News* pp. 163–174.

Psarakis, M., D. Gizopoulos, E. Sanchez, and M. S. Reorda (2010), 'Microprocessor Software-Based Self-Testing'. *IEEE Design & Test of Computers* **27**(3), 4–19.

Psarakis, M., A. Vavousis, C. Bolchini, and A. Miele (2014), 'Design and implementation of a self-healing processor on SRAM-based FPGAs'. In: *Proc. of IEEE Intl. Symp. on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. pp. 165–170.

Pumma, S., M. Si, W. Feng, and P. Balaji (2017), 'Parallel I/O Optimizations for Scalable Deep Learning'. In: *2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS)*.

Qiao, F., N. Zhou, Y. Chen, and H. Yang (2015), 'Approximate Computing in Chrominance Cache for Image/Video Processing'. *2015 IEEE International Conference on Multimedia Big Data* pp. 180–183.

Qiu, Q. and M. Pedram (1999), 'Dynamic power management based on continuous-time Markov decision processes'. In: *Proceedings 1999 Design Automation Conference (Cat. No. 99CH36361)*. pp. 555–561.

Qu, G., D. Kirovski, M. Potkonjak, and M. B. Srivastava (1999), 'Energy minimization of system pipelines using multiple voltages'. In: *Circuits and Systems, 1999. ISCAS '99. Proceedings of the 1999 IEEE International Symposium on*. pp. 362–365 vol.1.

Radetzki, M., C. Feng, X. Zhao, and A. Jantsch (2013), 'Methods for Fault Tolerance in Networks-on-Chip '. *ACM Computing Surveys* **46**(1), 8:1–8:38.

Raha, A., S. Venkataramani, V. Raghunathan, and A. Raghunathan (2015), 'Quality Configurable Reduce-and-rank for Energy Efficient Approximate Computing'. In: *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. San Jose, CA, USA, pp. 665–670, EDA Consortium.

Rahmani, A., P. Liljeberg, A. Hemani, A. Jantsch, and H. Tenhunen (2017a), *The Dark Side of Silicon*. Springer.

Rahmani, A. M., B. Donyanavard, T. Mück, K. Moazzemi, A. Jantsch, O. Mutlu, and N. Dutt (2018), 'SPECTR: Formal Supervisory Control and Coordination for Many-core Systems Resource Management'. In: *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. pp. 169–183.

Rahmani, A. M., M. H. Haghbayan, A. Kanduri, A. Y. Weldezion, P. Liljeberg, J. Plosila, A. Jantsch, and H. Tenhunen (2015), 'Dynamic power management for many-core platforms in the dark silicon era: A multi-objective control approach'. In: *2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*.

Rahmani, A. M., A. Jantsch, and N. Dutt (2017b), 'HDGM: Hierarchical Dynamic Goal Management for Many-Core Resource Allocation'. *IEEE Embedded Systems letters*.

Rahmani, A. M., K. Latif, P. Liljeberg, J. Plosila, and H. Tenhunen (2010), 'Research and practices on 3D networks-on-chip architectures'. In: *NORCHIP 2010*. pp. 1–6.

Ranganathan, P. and N. Jouppi (2005), 'Enterprise IT trends and implications for architecture research'. In: *High-Performance Computer Architecture, 2005. HPCA-11. 11th International Symposium on*. pp. 253–256.

Rantala, V., T. Lehtonen, P. Liljeberg, and J. Plosila (2008), 'Hybrid NoC with traffic monitoring and adaptive routing for future 3D integrated chips'. *Diagnostic Services in Network-on-Chips* p. 11.

Rehman, S., K. H. Chen, F. Kriebel, A. Toma, M. Shafique, J. J. Chen, and J. Henkel (2016), 'Cross-Layer Software Dependability on Unreliable Hardware'. *IEEE Transactions on Computers* **65**(1), 80–94.

Rinard, M. (2006), 'Probabilistic Accuracy Bounds for Fault-Tolerant Computations that Discard Tasks'. In: *Proceedings of International Conference on Supercomputing - ICS '06*. pp. 324–334.

Rinard, M. C. (2007), 'Using Early Phase Termination to Eliminate Load Imbalances at Barrier Synchronization Points'. *SIGPLAN Not.* **42**(10), 369–386.

Rotem, E., A. Naveh, A. Ananthakrishnan, E. Weissmann, and D. Rajwan (2012), 'Power-Management Architecture of the Intel Microarchitecture Code-Named Sandy Bridge'. *IEEE Micro* **32**(2), 20–27.

Rozo, L., A. M. Landwehr, Y. Zheng, C. Yang, and G. Gao (2018), 'Reliability-Aware Runtime Adaption Through a Statically Generated Task Schedule'. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* **26**(1), 11–22.

Rusu, C., A. Ferreira, C. Scordino, and A. Watson (2006), 'Energy-Efficient Real-Time Heterogeneous Server Clusters'. In: *12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06)*. pp. 418–428.

Saez, J. C., A. Fedorova, D. Koufaty, and M. Prieto (2012), 'Leveraging Core Specialization via OS Scheduling to Improve Performance on Asymmetric Multicore Systems'. *ACM Trans. Comput. Syst.* pp. 6:1–6:38.

Saez, J. C., M. Prieto, A. Fedorova, and S. Blagodurov (2010), 'A Comprehensive Scheduler for Asymmetric Multicore Systems'. In: *Proceedings of the 5th European Conference on Computer Systems (EuroSys)*. pp. 139–152.

Samadi, M., D. A. Jamshidi, J. Lee, and S. Mahlke (2014), 'Paraprox : Pattern-Based Approximation for Data Parallel Applications'. In: *Proceedings of International conference on Architectural support for programming languages and operating systems - ASPLOS '14*. pp. 35–50.

Samadi, M., J. Lee, and D. Jamshidi (2013), 'Sage: Self-tuning approximation for graphics engines'. In: *Proceedings of IEEE/ACM International Symposium on Microarchitecture - MICRO '13*.

Samman, F. A., T. Hollstein, and M. Glesner (2013), 'Runtime contention and bandwidth-aware adaptive routing selection strategies for networks-on-chip'. *IEEE Transactions on Parallel and Distributed Systems* pp. 1411–1421.

Sampson, A., W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman (2011), '{EnerJ}: Approximate Data Types for Safe and General Low-power Computation'. *Proceedings of the 32nd {ACM} {SIGPLAN} Conference on Programming Language Design and Implementation* pp. 164–174.

San Miguel, J. and M. Badr (2014), 'Load Value Approximation'. In: *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*.

Sanchez, D. and C. Kozyrakis (2011), 'Vantage: Scalable and Efficient Fine-grain Cache Partitioning'. In: *Proceedings of the 38th Annual International Symposium on Computer Architecture*. New York, NY, USA, pp. 57–68, ACM.

Saputra, H., M. Kandemir, N. Vijaykrishnan, M. J. Irwin, J. S. Hu, C.-H. Hsu, and U. Kremer (2002), 'Energy-conscious compilation based on voltage scaling'. *ACM SIGPLAN Notices* **37**(7), 2.

Scolari, A., F. Sironi, D. Sciuto, and M. D. Santambrogio (2014), 'A Survey on Recent Hardware and Software-Level Cache Management Techniques'. In: *2014 IEEE International Symposium on Parallel and Distributed Processing with Applications*. pp. 242–247.

Scott, S., D. Abts, J. Kim, and W. J. Dally (2006), 'The blackwidow high-radix clos network'. *ACM SIGARCH Computer Architecture News* pp. 16–28.

Sehatbakhsh, N., A. Nazari, A. Zajic, and M. Prvulovic (2016), 'Spectral profiling: Observer-effect-free profiling by monitoring EM emanations'. In: *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*.

Semiconductor Industry Association et al. (2011), 'International Technology Roadmap for Semiconductors'. http://www.itrs2.net/.

Seo, D., A. Ali, W.-T. Lim, N. Rafique, and M. Thottethodi (2005), 'Near-Optimal Worst-Case Throughput Routing for Two-Dimensional Mesh Networks'. In: *Proceedings of the 32nd Annual International Symposium on Computer Architecture (ISCA)*. pp. 432–443.

Shafique, M. and S. Garg (2017), 'Computing in the Dark Silicon Era: Current Trends and Research Challenges'. *IEEE Design Test* **34**(2), 8–23.

Shafique, M., S. Garg, T. Mitra, S. Parameswaran, and J. Henkel (2014), 'Dark silicon as a challenge for hardware/software co-design: Invited special session paper'. In: *Proc. of ACM International Conference on Hardware/Software Codesign and System Synthesis*. p. 13.

Shafique, M., B. Vogel, and J. Henkel (2013), 'Self-adaptive hybrid Dynamic Power Management for many-core systems'. In: *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*. pp. 51–56.

Shahosseini, S., K. Moazzemi, A. M. Rahmani, and N. Dutt (2017), 'Dependability evaluation of SISO control-theoretic power managers for processor architectures'. In: *2017 IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC)*.

Shamsa, E., A. Kanduri, A. M. Rahmani, P. Liljeberg, A. Jantsch, and N. Dutt (2018), 'Goal Formulation: Abstracting Dynamic Objectives for Efficient On-chip Resource Allocation'. In: *2018 IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC)*. pp. 1–4.

Shamsa, E., A. Kanduri, A. M. Rahmani, P. Liljeberg, A. Jantsch, and N. Dutt (2019), 'Goal-Driven Autonomy for Efficient On-chip Resource Management: Transforming Objectives to Goals'. In: *Proc. of Conf. on Design, Automation Test in Europe (DATE)*.

Shang, L., L.-S. Peh, and N. K. Jha (2003), 'Dynamic voltage scaling with links for power optimization of interconnection networks'. In: *The Ninth International Symposium on High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings*. pp. 91–102.

Sharifi, A., S. Srikantaiah, A. K. Mishra, M. Kandemir, and C. R. Das (2011), 'METE: Meeting End-to-end QoS in Multicores Through System-wide Resource Management'. *SIGMETRICS Perform. Eval. Rev.* pp. 13–24.

Shelepov, D., J. C. Saez Alcaide, S. Jeffery, A. Fedorova, N. Perez, Z. F. Huang, S. Blagodurov, and V. Kumar (2009), 'HASS: a scheduler for heterogeneous multicore systems'. *ACM SIGOPS Operating Systems Review* pp. 66–75.

Shin, Y. and K. Choi (1999), 'Power conscious fixed priority scheduling for hard real-time systems'. In: *Proceedings 1999 Design Automation Conference (Cat. No. 99CH36361)*. pp. 134–139.

Shojaei, H., A. Ghamarian, T. Basten, M. Geilen, S. Stuijk, and R. Hoes (2009), 'A parameterized compositional multi-dimensional multiple-choice knapsack heuristic for CMP run-time management'. In: *Design Automation Conference, 2009. DAC'09. 46th ACM/IEEE*. pp. 917–922.

Shoushtari, M., A. BanaiyanMofrad, and N. Dutt (2015), 'Exploiting Partially-Forgetful Memories for Approximate Computing'. *IEEE Embedded Systems Letters* **7**(1), 19–22.

Shye, A., J. Blomstedt, T. Moseley, V. J. Reddi, and D. A. Connors (2009), 'PLR: A Software Approach to Transient Fault Tolerance for Multicore Architectures'. *IEEE Transactions on Dependable and Secure Computing* **6**(2), 135–148.

Sidiroglou, S., S. Misailovic, H. Hoffmann, and M. Rinard (2011), 'Managing performance vs. accuracy trade-offs with loop perforation'. In: *Proceedings of ACM SIGSOFT Symposium and European Conference on Foundations of Software Engineering - SIGSOFT/FSE '11*. pp. 124–134.

Simevski, A., R. Kraemer, and M. Krstic (2014), 'Increasing multiprocessor lifetime by Youngest-First Round-Robin core gating patterns'. In: *Proc. of NASA/ESA Conf. on Adaptive Hardware and Systems (AHS)*. pp. 233–239.

Singh, A., W. J. Dally, A. K. Gupta, and B. Towles (2003), 'GOAL: a load-balanced adaptive routing algorithm for torus networks'. In: *ACM SIGARCH Computer Architecture News*. pp. 194–205.

Singh, A. et al. (2013a), 'Mapping on multi/many-core systems: survey of current and emerging trends'. In: *Proc. of DAC, 2013*. pp. 1:1–1:10.

Singh, A. K., P. Dziurzanski, H. R. Mendis, and L. S. Indrusiak (2017), 'A Survey and Comparative Study of Hard and Soft Real-Time Dynamic Resource Allocation Strategies for Multi-/Many-Core Systems'. *ACM Computing Surveys* **50**(2), 24:1–24:40.

Singh, A. K., A. Kumar, and T. Srikanthan (2011), 'A Hybrid Strategy for Mapping Multiple Throughput-constrained Applications on MPSoCs'. In: *Proceedings of the 14th International Conference on Compilers, Architectures and Synthesis for Embedded Systems*. pp. 175–184.

Singh, A. K., A. Kumar, and T. Srikanthan (2013b), 'Accelerating Throughput-aware Runtime Mapping for Heterogeneous MPSoCs'. *ACM Trans. Des. Autom. Electron. Syst.* pp. 9:1–9:29.

Singh, N. and S. Rao (2014), 'Ensemble Learning for Large-Scale Workload Prediction'. *IEEE Transactions on Emerging Topics in Computing* pp. 149–165.

Singla, G., G. Kaur, A. K. Unver, and U. Y. Ogras (2015), 'Predictive dynamic thermal and power management for heterogeneous mobile platforms'. In: *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*. pp. 960–965.

Skadron, K., M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan (2003), 'Temperature-aware microarchitecture'. In: *30th Annual International Symposium on Computer Architecture, 2003. Proceedings.* pp. 2–13.

Skitsas, M. A., C. A. Nicopoulos, and M. K. Michael (2016), 'DaemonGuard: Enabling O/S-Orchestrated Fine-Grained Software-Based Selective-Testing in Multi-/Many-Core Microprocessors'. *IEEE Transactions on Computers* **65**(5), 1453–1466.

Skitsas, M. A., C. A. Nicopoulos, and M. K. Michael (2018), 'Exploring System Availability During Software-Based Self-Testing of Multi-core CPU'. *Journal of Electronic Testing: Theory and Application* **34**(1), 67–81.

Smolens, J. C., B. T. Gold, B. Falsafi, and J. C. Hoe (2006), 'Reunion: Complexity-Effective Multicore Redundancy'. In: *Proc. of the Intl. Symp. on Microarchitecture (MICRO)*. pp. 223–234.

Song, W. J., S. Mukhopadhyay, and S. Yalamanchili (2015), 'Managing performance-reliability tradeoffs in multicore processors'. In: *Proc. of IEEE Intl. Reliability Physics Symp.* pp. 3C.1.1–3C.1.7.

Soteriou, V. and L. Peh (2007), 'Exploring the Design Space of Self-Regulating Power-Aware On/Off Interconnection Networks'. *IEEE Transactions on Parallel and Distributed Systems* pp. 393–408.

Srinivasan, J., S. V. Adve, P. Bose, and J. A. Rivers (2004), 'The Case for Lifetime Reliability-Aware Microprocessors'. In: *Proc. of Intl. Symp. on Computer Architecture (ISCA)*. pp. 276–287.

Srinivasan, S. T. and A. R. Lebeck (1998), 'Load latency tolerance in dynamically scheduled processors'. In: *Proceedings. 31st Annual ACM/IEEE International Symposium on Microarchitecture*. pp. 148–159.

St Amant, R., A. Yazdanbakhsh, J. Park, B. Thwaites, H. Esmaeilzadeh, A. Hassibi, L. Ceze, and D. Burger (2014), 'General-purpose code acceleration with limited-precision analog computation'. In: *Proceeding of International Symposium on Computer Architecuture - ISCA '14*. pp. 505–516.

Subasi, O., G. Yalcin, F. Zyulkyarov, O. Unsal, and J. Labarta (2017), 'Designing and Modelling Selective Replication for Fault-Tolerant HPC Applications'. In: *Proc. of Intl. Symp. on Cluster, Cloud and Grid Computing (CCGRID)*. pp. 452–457.

Subramanian, L., D. Lee, V. Seshadri, H. Rastogi, and O. Mutlu (2016), 'BLISS: Balancing Performance, Fairness and Complexity in Memory Access Scheduling'. *IEEE Transactions on Parallel and Distributed Systems* pp. 3071–3087.

Subramanian, L., V. Seshadri, A. Ghosh, S. Khan, and O. Mutlu (2015), 'The Application Slowdown Model: Quantifying and Controlling the Impact of Inter-application Interference at Shared Caches and Main Memory'. In: *Proceedings of the 48th International Symposium on Microarchitecture (MICRO)*. pp. 62–75.

Sui, X., A. Lenharth, D. S. Fussell, and K. Pingali (2016), 'Proactive Control of Approximate Programs'. In: *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*. New York, NY, USA, pp. 607–621, ACM.

Suleman, M. A., O. Mutlu, M. K. Qureshi, and Y. N. Patt (2009), 'Accelerating Critical Section Execution with Asymmetric Multi-core Architectures'. In: *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. pp. 253–264.

Sun, G., C.-W. Chang, and B. Lin (2013), 'A New Worst-Case Throughput Bound for Oblivious Routing in Odd Radix Mesh Network'. *IEEE Computer Architecture Letters* **12**(1), 9–12.

Sun, J., R. Lysecky, K. Shankar, A. Kodi, A. Louri, and J. Roveda (2014), 'Workload Assignment Considering NBTI Degradation in Multicore Systems'. *Journal Emerg. Technol. Comput. Syst.* **10**(1), 4:1–4:22.

Sung, H., J. Min, S. Ha, and H. Eom (2017), 'OMBM: Optimized Memory Bandwidth Management for Ensuring QoS and High Server Utilization'. In: *Foundations and Applications of Self* Systems (FAS* W), 2017 IEEE 2nd International Workshops on*. pp. 269–276.

Tai, J., D. Liu, Z. Yang, X. Zhu, J. Lo, and N. Mi (2017), 'Improving Flash Resource Utilization at Minimal Management Cost in Virtualized Flash-Based Storage Systems'. *IEEE Transactions on Cloud Computing* pp. 537–549.

Tan, C. et al. (2015), 'Approximation-aware scheduling on heterogeneous multi-core architectures'. In: *In Proc. of ASP-DAC,*. pp. 618–623.

Tang, L., J. Mars, and M. L. Soffa (2012), 'Compiling for Niceness: Mitigating Contention for QoS in Warehouse Scale Computers'. In: *Proceedings of the Tenth International Symposium on Code Generation and Optimization (CGO)*. pp. 1–12.

Tang, L., J. Mars, N. Vachharajani, R. Hundt, and M. L. Soffa (2011), 'The impact of memory subsystem resource sharing on datacenter applications'. In: *ACM SIGARCH Computer Architecture News*, Vol. 39(3). pp. 283–294.

Tavakkol, A., M. Sadrosadati, S. Ghose, J. Kim, Y. Luo, Y. Wang, N. M. Ghiasi, L. Orosa, J. Gómez-Luna, and O. Mutlu (2018), 'FLIN: Enabling Fairness and Enhancing Performance in Modern NVMe Solid State Drives'. In: *Proc. of Intl. Symp. on Computer Architecture (ISCA)*. pp. 397–410.

Tedesco, L. P., T. Rosa, F. Clermidy, N. Calazans, and F. G. Moraes (2010), 'Implementation and Evaluation of a Congestion Aware Routing Algorithm for Networks-on-chip'. In: *Proceedings of the 23rd Symposium on Integrated Circuits and System Design (SBCCI)*.

Teodorescu, R. and J. Torrellas (2008), 'Variation-Aware Application Scheduling and Power Management for Chip Multiprocessors'. In: *2008 International Symposium on Computer Architecture*. pp. 363–374.

Tesauro, G. and et.al. (2007), 'Managing Power Consumption and Performance of Computing Systems Using Reinforcement Learning'. In: *Int. Conf. on Neural Information Processing Systems*.

Tesauro, G., N. Jong, R. Das, and M. Bennani (2006), 'A hybrid reinforcement learning approach to autonomic resource allocation'. In: *3rd IEEE International Conference on Autonomic Computing*. pp. 65–73.

Thwaites, B., G. Pekhimenko, A. Yazdanbakhsh, J. Park, G. Mururu, and T. Mowry (2014), 'Rollback-Free Value Prediction with Approximate Loads'. In: *Proceedings of IEEE/ACM International Conference on Parallel Architectures and Compilation Techniques - PACT '14)*.

Tilli, A., A. Bartolini, M. Cacciari, and L. Benini (2015), 'Guaranteed Computational Resprinting via Model-Predictive Control'. *ACM Trans. Embed. Comput. Syst.*

Torng, C., M. Wang, and C. Batten (2016), 'Asymmetry-aware work-stealing runtimes'. In: *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*. pp. 40–52.

Tziantzioulis, G., A. Gok, S. Faisal, N. Hardavellas, S. Ogrenci-Memik, and S. Parthsarathy (2016), 'Lazy pipelines: Enhancing quality in approximate computing'. In: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016*. pp. 1381–1386.

Tzilis, S., I. Sourdis, V. Vasilikos, D. Rodopoulos, and D. Soudris (2016), 'Runtime Management of Adaptive MPSoCs for Graceful Degradation'. In: *Proc. of Intl. Conf. on Compilers, Architectures and Synthesis for Embedded Systems (CASES)*. pp. 5:1–5:10.

Unsal, O. S., R. Ashok, I. Koren, C. M. Krishna, and C. A. Moritz (2001), 'Cool-cache for hot multimedia'. In: *Proceedings. 34th ACM/IEEE International Symposium on Microarchitecture. MICRO-34*. pp. 274–283.

Unsal, O. S. and I. Koren (2003), 'System-level power-aware design techniques in real-time systems'. *Proceedings of the IEEE* pp. 1055–1069.

Unsal, O. S., I. Koren, C. M. Krishna, and C. A. Moritz (2002), 'The minimax cache: an energy-efficient framework for media processors'. In: *Proceedings Eighth International Symposium on High Performance Computer Architecture*. pp. 131–140.

Valiant, L. G. and G. J. Brebner (1981), 'Universal Schemes for Parallel Communication'. In: *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*. New York, NY, USA, pp. 263–277, ACM.

Van Craeynest, K., S. Akram, W. Heirman, A. Jaleel, and L. Eeckhout (2013), 'Fairness-aware Scheduling on single-ISA Heterogeneous Multi-cores'. In: *Proceedings of the 22Nd International Conference on Parallel Architectures and Compilation Techniques (PACT)*. pp. 177–188.

Van Craeynest, K., A. Jaleel, L. Eeckhout, P. Narvaez, and J. Emer (2012), 'Scheduling Heterogeneous Multi-cores Through Performance Impact Estimation (PIE)'. In: *Proceedings of the 39th Annual International Symposium on Computer Architecture*. pp. 213–224.

van den Brand, J. W., C. Ciordas, K. Goossens, and T. Basten (2007), 'Congestion-controlled best-effort communication for networks-on-chip'. In: *Proceedings of the conference on Design, automation and test in Europe*. pp. 948–953.

Vargas, V., P. Ramos, J.-F. Méhaut, and R. Velazco (2018), 'NMR-MPar: A Fault-Tolerance Approach for Multi-Core and Many-Core Processors'. *Applied Sciences* **8**(3).

Vassighi, A. and M. Sachdev (2006), 'Thermal runaway in integrated circuits'. *IEEE Transactions on Device and Materials Reliability*.

Venkataramani, S., V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan (2013), 'Quality programmable vector processors for approximate computing'. *46th Annual IEEE/ACM International Symposium* pp. 1–12.

Vogt, L., Y. Chara, H. Ouannani, and M. Nazih (2007), 'Integrated temperature sensor with digital output for SoC power management'. In: *2007 International Conference on Design Technology of Integrated Systems in Nanoscale Era*. pp. 7–12.

Wang, F., S. Oral, S. Gupta, D. Tiwari, and S. S. Vazhkudai (2014), 'Improving large-scale storage system performance via topology-aware and balanced data placement'. In: *2014 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*. pp. 656–663.

Wang, L., X. Wang, and T. Mak (2016), 'Adaptive Routing Algorithms for Lifetime Reliability Optimization in Network-on-Chip'. *IEEE Transactions on Computers* **65**(9), 2896–2902.

Wang, T. and Q. Xu (2014), 'On the Simulation of NBTI-Induced Performance Degradation Considering Arbitrary Temperature and Voltage Variations'. In: *Proc. of Design Automation Conf. (DAC)*. pp. 169:1–169:6.

Wang, T., Q. Zhang, and Q. Xu (2017a), 'ApproxQA: a unified quality assurance framework for approximate computing'. In: *Proceedings of the Conference on Design, Automation & Test in Europe*. pp. 254–257.

Wang, X. and J. F. Martínez (2015), 'XChange: A market-based approach to scalable dynamic multi-resource allocation in multicore architectures'. In: *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*. pp. 113–125.

Wang, X. and J. F. Martínez (2016), 'ReBudget: Trading Off Efficiency vs. Fairness in Market-Based Multicore Resource Allocation via Runtime Budget Reassignment'. In: *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems (ISCA)*. pp. 19–32.

Wang, Y., H. Li, and X. Li (2017b), 'Real-Time Meets Approximate Computing: An Elastic CNN Inference Accelerator with Adaptive Trade-off Between QoS and QoR'. In: *Proceedings of the 54th Annual Design Automation Conference 2017*. New York, NY, USA, pp. 33:1–33:6, ACM.

Wells, P. M., K. Chakraborty, and G. S. Sohi (2009), 'Mixed-mode Multicore Reliability'. *SIGARCH Comput. Archit. News* **37**(1), 169–180.

Wentzlaff, D., P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. F. Brown III, and A. Agarwal (2007), 'On-chip interconnection architecture of the tile processor'. *IEEE micro* pp. 15–31.

Winter, J. A., D. H. Albonesi, and C. A. Shoemaker (2010), 'Scalable thread scheduling and global power management for heterogeneous many-core architectures'. In: *2010 19th International Conference on Parallel Architectures and Compilation Techniques (PACT)*.

Wolf, M., S. Bhattacharyya, J. Florence, and A. E. Sapio (2016), 'Power and Thermal Modeling for Communication Systems'. In: *2016 IEEE International Workshop on Signal Processing Systems (SiPS)*. pp. 136–141.

Wu, D., B. M. Al-Hashimi, and M. T. Schmitz (2006), 'Improving Routing Efficiency for Network-on-chip Through Contention-aware Input Selection'. In: *Proceedings of the 2006 Asia and South Pacific Design Automation Conference (ASP-DAC)*. pp. 36–41.

Wu, W. and A. Louri (2016), 'A Methodology for Cognitive NoC Design'. *IEEE Computer Architecture Letters* **15**(1), 1–4.

Wu, Y., C. Lu, and Y. Chen (2016), 'A Survey of Routing Algorithm for Mesh Network-on-Chip'. *Front. Comput. Sci.* pp. 591–601.

Xiang, Y., T. Chantem, R. P. Dick, X. S. Hu, and L. Shang (2010), 'System-level reliability modeling for MPSoCs'. In: *Proc. of Conf. on Hardware/Software Codesign and System Synthesis (CODES)*. pp. 297–306.

Xiang, Y. and S. Pasricha (2015), 'Soft and Hard Reliability-Aware Scheduling for Multicore Embedded Systems with Energy Harvesting'. *IEEE Transactions on Multi-Scale Computing Systems* **1**(4), 220–235.

Xu, C., X. Wu, W. Yin, Q. Xu, N. Jing, X. Liang, and L. Jiang (2017), 'On Quality Trade-off Control for Approximate Computing Using Iterative Training'. In: *Proceedings of the 54th Annual Design Automation Conference 2017*. New York, NY, USA, pp. 52:1–52:6, ACM.

Xu, Q., T. Mytkowicz, and N. S. Kim (2016a), 'Approximate Computing: A Survey'. *IEEE Design & Test* **33**(1), 8–22.

Xu, S., B. Fu, M. Chen, and L. Zhang (2016b), 'Congestion-Aware Adaptive Routing with Quantitative Congestion Information'. In: *High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2016 IEEE 18th International Conference on*. pp. 216–223.

Xue, L., F. Li, M. Kandemir, I. Kolcu, et al. (2006), 'Dynamic partitioning of processing and memory resources in embedded MPSoC architectures'. In: *Proceedings of the conference on Design, automation and test in Europe: Proceedings*. pp. 690–695.

Yamamoto, A. Y. and C. Ababei (2014), 'Unified reliability estimation and management of NoC based chip multiprocessors'. *Microprocessors and Microsystems* **38**(1), 53–63.

Yang, H., A. Breslow, J. Mars, and L. Tang (2013), 'Bubble-flux: Precise Online QoS Management for Increased Utilization in Warehouse Scale Computers'. In: *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA)*. pp. 607–618.

Yang, Z., J. Tai, J. Bhimani, J. Wang, N. Mi, and B. Sheng (2016), 'GReM: Dynamic SSD resource allocation in virtualized storage systems with heterogeneous IO workloads'. In: *2016 IEEE 35th International Performance Computing and Communications Conference (IPCCC)*. pp. 1–8.

Yao, F., A. Demers, and S. Shenker (1995), 'A scheduling model for reduced CPU energy'. In: *Proceedings of IEEE 36th Annual Foundations of Computer Science*. pp. 374–382.

Ye, Y., R. West, Z. Cheng, and Y. Li (2014), 'COLORIS: A Dynamic Cache Partitioning System Using Page Coloring'. In: *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation*. New York, NY, USA, pp. 381–392, ACM.

You, D. and K. S. Chung (2014), 'Dynamic power management for embedded processors in system-on-chip designs'. *Electronics Letters* pp. 1309–1310.

Zahedi, S. M. and B. C. Lee (2014), 'REF: Resource Elasticity Fairness with Sharing Incentives for Multiprocessors'. In: *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ISCA)*. pp. 145–160.

Zhang, T., J. L. Abellán, A. Joshi, and A. K. Coskun (2014), 'Thermal management of manycore systems with silicon-photonic networks'. In: *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*.

Zhang, X., S. Dwarkadas, and K. Shen (2009), 'Towards practical page coloring-based multicore cache management'. In: *Proceedings of the 4th ACM European conference on Computer systems*. pp. 89–102.

Zhang, X., Y. Zhang, B. R. Childers, and J. Yang (2017), 'DrMP: Mixed Precision-aware DRAM for High Performance Approximate and Precise Computing'. In: *Parallel Architectures and Compilation Techniques (PACT), 2017 26th International Conference on*. pp. 53–63.

Zhang, Y., M. A. Laurenzano, J. Mars, and L. Tang (2014), 'SMiTe: Precise QoS Prediction on Real-System SMT Processors to Improve Utilization in Warehouse Scale Computers'. In: *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. pp. 406–418.

Zhao, B., H. Aydin, and D. Zhu (2008), 'Reliability-aware Dynamic Voltage Scaling for energy-constrained real-time embedded systems'. In: *Proc. of IEEE Intl. Conf. on Computer Design (ICCD)*. pp. 633–639.

Zhao, Y., J. Rao, and Q. Yi (2016), 'Characterizing and Optimizing the Performance of Multithreaded Programs Under Interference'. In: *Proceedings of the 2016 International Conference on Parallel Architectures and Compilation (PACT)*. pp. 287–297.

Zhou, Y., H. Hoffmann, and D. Wentzlaff (2016), 'CASH: Supporting IaaS Customers with a Sub-core Configurable Architecture'. In: *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA)*. pp. 682–694.

Zhuravlev, S., J. C. Saez, S. Blagodurov, A. Fedorova, and M. Prieto (2012), 'Survey of Scheduling Techniques for Addressing Shared Resources in Multicore Processors'. *ACM Computing Surveys* **45**(1), 4:1–4:28.

Zipf, P., G. Sassatelli, N. Utlu, N. Saint-Jean, P. Benoit, and M. Glesner (2009), 'A Decentralised Task Mapping Approach for Homogeneous Multiprocessor Network-on-chips'. *Int. J. Reconfig. Comput.* pp. 3:1–3:14.

Zong, W., M. O. Agyemen, X. Wang, and T. Maky (2015), 'Unbiased Regional Congestion Aware Selection Function for NoCs'. In: *Proceedings of the 9th International Symposium on Networks-on-Chip (NOCS)*. pp. 19:1–19:8.