Real-time ASIC Monitoring for System-level Power and Thermal Management

Georgios Kornaros

A thesis submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy in
Electronic & Computer Engineering

Doctoral Committee

Professor Dionisios Pnevmatikatos, Supervisor
Professor Apostolos Dollas, Member
Associate Professor Ioannis Papaeystathiou, Member
Professor Georgios Alexiou, Member
Assistant Dimitrios Soudris, Member
Professor Georgios Stamoulis, Member
Associate Professor Nikos Bellas, Member

Technical University of Crete
November 2013

Παρακολούθηση και Διαχείριση Συστημάτων ASIC σε Πραγματικό Χρόνο με Στόχο την Διαχείριση Ισχύος και Θερμοκρασίας

Γεώργιος Κορνάρος

Νοέμβριος 2013

Περίληψη

Η παρακολούθηση και διαχείριση συστημάτων σε chip σε πραγματικό χρόνο αποτελεί μια εναλλακτική μέθοδο στις σημερινές μεθόδους επαλήθευσης που χρησιμοποιούνται κατά την σχεδίαση. Η παρακολούθηση και διαχείριση συστημάτων σε chip κατά την διάρκεια λειτουργίας τους έχει στόχο να εκτιμάται και να επιτυγχάνεται υψηλότερη απόδοση του συστήματος μέσα σε ένα ευρύ φάσμα συνθηκών και πεδίων εφαρμογών και κάτω από συγκεκριμένους περιορισμούς λειτουργίας σε σχέση με την ισχύ και την θερμοκρασία. Η σχεδίαση υπολογιστικών συστημάτων, από συμβατικούς πολύ-επεξεργαστές έως και εξειδικευμένα συστήματα προσαρμοσμένα με μεγάλο αριθμό μονάδων επεξεργασίας, αποτελεί έναν τομέα που εξελίσσεται ραγδαία με συνέπεια οι συμβατικές μεθοδολογίες σχεδίασης και επαλήθευσης να είναι ανεπαρκείς. Νέες τεχνικές παρακολούθησης και διαχείρισης σε πραγματικό χρόνο ενσωματώνονται σταδιακά στον χαρακτηρισμό για επιδόσεις και κατανάλωση ισχύος και συνακόλουθα στην αποτελεσματική βελτίωση της απόδοσης και της ενεργειακής συμπεριφοράς τέτοιων συστημάτων.

Στην διατριβή αυτή αναπτύχθηκε μία μεθοδολογία σχεδίασης που χρησιμοποιεί νέα δομικά στοιχεία σε κύκλωμα και αποδεικνύει ότι είναι εφικτή η ανάλυση και ο χαρακτηρισμός των παραγόντων που δυναμικά αλληλεπιδρούν και επηρεάζουν την απόδοση και κατανάλωση ισχύος σε συστημάτα σε chip. Συγκεκριμένα δομικά στοιχεία σε κύκλωμα για την παρακολούθηση και διαχείριση επιτρέπουν στο λογισμικό διαχείρισης που βρίσκεται σε ανώτερο επίπεδο να συλλέξει την απαραίτητη πληροφορία και να αποκτήσει μέσω αυτής βαθύτερη επίγνωση για τις αιτίες που προκαλούν μειωμένες επιδόσεις. Η διαδικασία αυτή είναι ιδιαίτερα σημαντική σήμερα στα πολυπύρηνα συστήματα σε chip με την μεταβλητότητα που υπεισέρχεται λόγω τεχνολογίας αλλά και λόγω της μη-ντετερμινιστικής φύσης των προγραμμάτων. Σε αντίθεση με άλλες τεχνικές που υιοθετούν συγκεκριμένες και περιορισμένες λύσεις για ένα σύστημα, τα δομικά στοιχεία σε κύκλωμα για την παρακολούθηση και διαχείριση που αναπτύχθηκαν επιτρέπουν σε πραγματικό χρόνο λειτουργίας την ανίχνευση και εφαρμογή των βέλτιστων λύσεων για την χρήση των πόρων του συστήματος και την βελτιστοποίηση της συμπεριφοράς του.

Ταυτόχρονα στην διατριβή αυτή περιγράφονται λύσεις σε επίπεδο μικροαρχιτεκτονικής για την παρακολούθηση και διαχείριση συστημάτων που ενσωματώνουν δίκτυο σε chip. Περιγράφονται τεχνικές με χρήση κυκλωμάτων υψηλής ταχύτητας για ενσωμάτωση ειδικών μονάδων παρακολούθησης και διαχείρισης σε στρατηγικά σημεία σε ένα σύστημα και ειδικότερα στις διεπαφές ενός δικτύου σε chip. Τα κυκλώματα αυτά παρακολούθησης επιτρέπουν τον δυναμικό προγραμματισμό τους και την συλλογή πληροφοριών σε κεντρικούς ή κατανεμημένους διαχειριστές.

Επίσης, τα αποτελέσματα συγκεκριμένων υλοποιήσεων σε πλατφόρμες με αναδιατασσόμενη λογική αποδεικνύουν την χρησιμότητα και αποτελεσματικότητα φίλτρων στα κυκλώματα παρακολούθησης σε πραγματικό χρόνο μέσα σε σύνθετα συστήματα σε chip.

Τέλος, η διατριβή αυτή θέτει τις βάσεις για μελλοντική έρευνα στον χώρο της παρακολούθησης και διαχείρισης πληροφορίας σε πραγματικό χρόνο, όπου η παραγωγική συνεργασία και επικοινωνία μεταξύ των υποσυστημάτων παρακολούθησης και των εφαρμογών επιτρέπει την βελτιστοποίηση της απόδοσης ενώ ταυτόχρονα εναρμονίζεται με τους περιορισμούς λειτουργίας σε θερμοκρασία και ισχύ ενός chip.

Abstract

RUN-TIME MONITORING of Systems-on-Chip offers an alternative to today's largely ad hoc, design-time validation methodologies to estimate and extract the highest performance of the system across a wide spectrum of design corners and applications domains under specific power and temperature constraints. The diversity of designs ranging from conventional multiprocessor machines to designs that consist of a "sea" of programmable arithmetic logic units or other specialized custom units present a dramatically evolving design space that validation methods cannot cover in a conventional manner. New run-time monitoring techniques are increasingly employed to characterize and assist in improving the performance and the energy consuming behavior of such systems.

The developed methodology in this dissertation uses novel designed hardware primitives and shows that detailed analysis and characterization of the different interacting components that dynamically influence the performance and power consumption of SoC is feasible. In particular, hardware monitoring structures allow to the upper management software layer to obtain feedback from the hardware design and better understand and deal with the predominant symptoms of inefficiencies. This becomes especially true today, in the presence of multi- and many- core SoCs, of the technological process variability and of the workload variations caused by the non-deterministic nature of applications. Contrasted with previous solutions that tend to define at design-time a static and finite set of configurations for a system, the designed hardware monitoring agents offer run-time assistance to find and exploit optimal ways to manage resources and improve the system's behavior.

Various microarchitecture alternatives are described for run-time monitoring and management

of network-on-chip based systems. High-speed and energy efficient circuit techniques are deployed for monitoring agents that reside at critical points inside chips, especially at interfaces; these monitoring circuits can be configured dynamically and communicate computed statistics to centralized or distributed hardware monitor managers of different functionality and complexity. The presented results extracted from implementations on reconfigurable devices expose the usability properties of monitoring and provide guidelines for system-level designers, proving the need for flexible and at the same time efficient filters for run-time monitoring inside complex NoC-based SoCs.

This work should lay the foundationfor further research in the domain of run-time system monitoring, where a harmonized synergistic operation between the hardware monitoring subsystem and the applications allow for optimized performance while respecting chip's energy or temperature constraints.

Acknowledgments

I would like first of all to thank my supervisor, Prof. Dionisios Pnevmatikatos for his guidance, support and encouragement, and for numerous conversations that has profoundly influenced my thinking about the subject of this thesis. Also, I would like to express my gratidute to Prof. Dollas and Asoc. Prof. Papaeystathiou for their helpful feedback and constructive suggestions. Finally, I would like to thank the members of my doctoral committee for their comments on improving and extending this thesis.

Lastly, but most importantly, I wish to thank my wife and my parents; my gratitude to them is beyond words.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

PERFORMANCE and energy efficiency have always been the ultimate design goals for all platforms from embedded to High-Performance Computing (HPC). While in embedded mobile platforms one key concern is an extended battery life, in servers or HPC, energy consumption dictates the overall operation costs with significant economic implications for datacenter owners. Embedded applications increasingly require higher performance, low-power processing to support the emergence of multimedia in hand-held devices, high-speed connectivity and faster data-processing. This brings new challenges with multiple tasks required to be executed in parallel. Given a processing algorithm, custom designs always provide the optimal solution, but rising development costs limit application-specific chips to devices that will be sold in very high volumes. Even then, custom circuits cannot meet time-to-market constraints since a design must rapidly adapt to changing market requirements. Extending product functionality and improving their lifetime requires additional features to satisfy the growing customer's needs as well as new market and technology trends.

As a more flexible alternative, programmable domain-specific processors have evolved to exploit particular forms of parallelism common to certain classes of embedded applications. Examples include digital signal processors (DSPs), media processors, network processors, and field-programmable gate arrays (FPGAs). However, full systems often require a heterogeneous mix of these cores to be competitive on complex workloads with a variety of processing tasks. The resulting

devices are complex System-on-Chips that combine multiple, general purpose processors for flexibility, with multiple custom function cores for performance and efficiency. These systems exhibit a highly dynamic behavior due to the multitude of the system components and their interactions.

Designing energy-efficient components fades outs in view of increasing importance of developing energy-efficient systems. Beyond micro-architectural enhancements and improved design flows, a holistic approach that includes also operating systems, libraries, protocol stacks, and compilers is more effective. Designers should design simple energy/performance management hooks in the processor and all other components (RAMs, controllers, chipsets, and interconnections) that can be exposed to the software, beyond voltage and clock scaling. Subsystems with power-performance "gears" to support power management at a higher level are more promising, enabling global energy management and different policies tailored to an application domain or changing system constraints.

Silicon vendors are constantly facing pressure to deliver feature-rich, high-performance, low-power and low-cost chips in as short time as possible. Thus, in the recent decade designers have developed strategies to reuse previously verified circuits called *Intellectual Property (IP) Cores*. The goal has been to reduce the time-to-market, the design costs and the productivity gap. To address these challenges, as figure 1.1 shows, the reuse of IP cores along with the evolvement of EDA tools and the hardware-software co-design methodologies have helped developers producing multi-billion transistor single chips in very short time.

Emerging multicore architectures become more diverse both at the architecture and the programming level, while varying application requirements at run-time compose a spectrum where optimum performance, predictability, fault diagnosis and recovery, combined with power efficiency during operation require radically new design methodologies. To address these needs intelligent system monitoring and runtime control of the state and dynamics of the device is becoming increasingly important. System-on-chip monitoring goals involve, first, to enhance system flexibility to adaptively match system resources on-line to dynamic workloads and second, to achieve energy efficiency and desired performance under power and thermal constraints. These objectives come on top to the traditional use of software and hardware techniques, such as watchdog processors [1, 2]

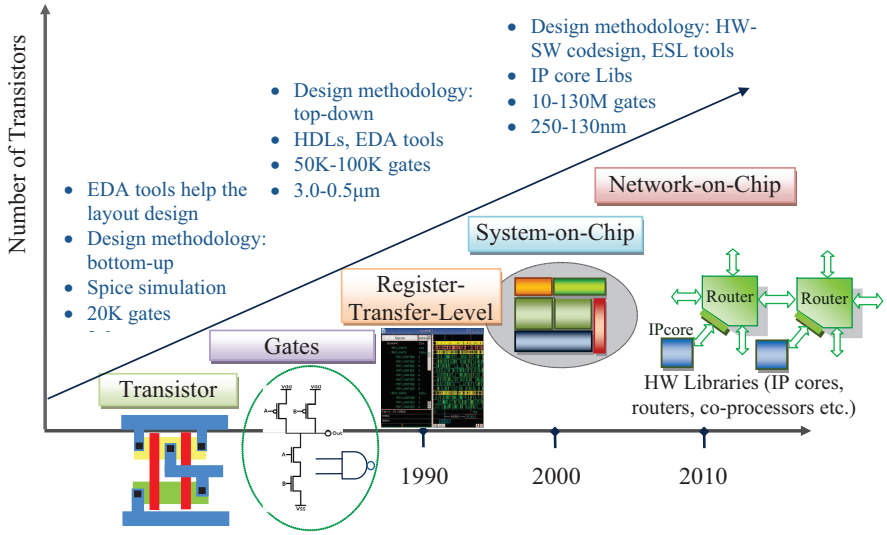

Figure 1.1: Improvements of tools, strategies, silicon manufacturing techniques boosting the design of integrated circuits from few gates to billion-transistor SoCs and NoCs

that exploit special-purpose hardware modules to monitor the control-flow of programs, as well as memory accesses for potential faults. In addition to dependability of on-chip systems that becomes an increasing problem as feature sizes continue to shrink, intelligent system monitoring is required to handle systems' adaptivity and enhance their efficiency.

1.1 Dynamic Power and Temperature Management in Multicore SoCs

Dynamic power management refers to power management schemes implemented while a chip is powered and tasks are running. Research efforts usually focus on power/energy reduction by Dynamic Voltage / Frequency Scaling (DVFS) techniques which control the supply voltage and the clock frequency during execution of a task according to computation requirement of the task. Because dynamic power consumption in a CMOS circuit scales quadratically with the supply voltage,

a significant power reduction is expected by DVFS techniques.

Processors are the center of attention in optimizing chip microarchitectures from the perspective of power and thermal dissipation. However, there is increasing concern about non-processing elements such as caches and on-chip interconnects. Nowadays, power and temperature management techniques need to shift from the era of single microprocessors to modern multi-core architectures. As CMOS technology is continuously scaling, single chip systems integrating a large number of processors, on-chip memories and custom intellectual property cores (IPcores) have become a reality [3]. Innovative architectures that allow different cores communicate to each other via the Network-on-Chip (NoC) paradigm have emerged as a promising alternative to traditional bus-based approaches [4]. By eliminating global wires, the NoC approach provides the needed scalability and predictability, while facilitating design reuse.

With communication emerging as a larger power and performance constraint than computation, it may become necessary to understand and leverage the properties of the on-chip communication subsystem at a higher level. One of the greatest bottlenecks to performance in future Systems-on-Chip is the high cost of on-chip communication through global wires [5]. Power consumption has also emerged as a first order design metric. Nir [6] shows that wires contribute up to 50% of total chip power in some processors. Large-scale chip multi-processors (CMPs) are already on the way by many major chip manufacturers [7], [8]. Multi-threaded work-loads that execute on such processors will experience high on-chip communication latencies and will dissipate significant power in interconnect.

Moreover, heterogenous SoCs are emerging to address the growing concerns of energy efficiency and silicon area effectiveness [9]. Small-scale heterogeneous multicore SoCs have been used in embedded systems for years [10]. Meanwhile, general-purpose processor designers are also advocating such heterogeneous architectures for future multicore or many-core processors to optimize a system's energy efficiency (measured in performance per joule) or area effectiveness (measured in performance per mm^2).

In addition, thermal constraints appear to dominate other physical constraints like pin-bandwidth

and silicon area. Power density of modern SoCs has been increasing at an alarming rate resulting in high and uneven on-chip temperatures. Higher temperature increases current leakage and causes poor reliability. Together with power consumption, clock distribution and process variation, problems in future multiprocessor systems-on-chip (MPSoCs) compose a multi-disciplinary equation. Joint optimization across multiple design variables is necessary. The need for adaptive chip architectures that can dynamically accommodate the full range of workloads, from heavily CPU-bound to heavily memory-bound and communication bound is rising together with the integration level of modern MPSoCs.

The operating system cannot assume all cores to be equal due to process variations or have similar behavior, performance and power profile. Spatial thermal variations are typically larger in heterogeneous MPSoCs due to the intrinsic disparity in power densities across the chip. Localized temperature increase hot spots and can cause transient reduction in overall system performance, unreliable timing delay variations or even permanent damages in the devices [11]. Attaining uniform power distribution in nanometer VLSI devices so as to avoid performance and long-term reliability degradation is not a trivial task. To achieve a reliable and efficient system operation, the MPSoCs need to operate below a maximum temperature value and power budget. Thermal throttling is a mechanism that reduces power consumption when the core temperature exceeds a threshold temperature limit. However, excessive thermal throttling affects the system performance. Determining the operating frequencies or even voltages of the different cores of the MPSoCs, such that the performance of the system is maximized, while satisfying the temperature and power budget constraints, is a challenging task.

The mere objective of each methodology is to achieve the theoretical highest performance within a thermally-safe DVFS configuration for specific or different workloads. This objective can be analyzed to various sub-problems: statically use techniques such as workload profiling to produce predictions, or employ dynamic management mechanisms in software or hardware in cases that a-priori knowledge of workload cannot be safely obtained, or to accommodate for process variation effects. The accuracy of the prediction or minimizations of prediction deviation together with the

need for fast adjustment of the constructed thermal model are a few subsequent problems that need careful consideration.

As the power densities increase with the continuous shrinking geometries, thermal hot-spots and large temperature variations on the die may occur, introducing a number of significant challenges. The main goals for power and thermal-aware management along with associated issues can be summarized in:

- accurate system modeling for power characterization and/or thermal characterization,
- accuracy of power and temperature detection sensors,
- circuits and techniques for real-time safe identification and reaction
- workload behavior characterization and phase prediction,
- accuracy of prediction (coarse or fine grain) algorithm,
- fast and low overhead algorithm/techniques for (a) updating the thermal model at runtime, and (b) predicting system's future behavior
- efficient reaction strategy with the following goals under specific power budget, or thermal constraints:
  - Maximize per-core performance or processed workload which consists of the right mixture of tasks (cold and hot, or urgent and best-effort, or real-time and idle)
  - Maximize system performance or chip performance utilizing the maximum number of functional units or resources,
  - Maximize per-application performance

In general, the design space exploration for run-time power and temperature management involves an extremely large number of parameters in order to fit the best strategy in terms of efficiency (fine or coarse grain, transistor, RTL level and architecture technique or compiler, application and OS level) to the right system, chip, and environment: CMPs, heterogeneous MPSoCs, NoC-based systems, multi-threaded environment, high-performance processors, multi-level memory hierarchies, embedded systems, real-time OS, embedded OS.

1.2 Contributions

This dissertation explores the potential to enhance the run-time monitoring mechanisms in multi-core SoCs and introduce novel system-level monitoring mechanisms, in order to facilitate dynamic optimization of system performance, power and temperature effects. Performance monitoring units are already present in current microprocessors. However, as the number of processors and the heterogeneity levels increase rapidly in modern systems, along with the application complexity, new methods and circuits are required to deliver optimal system operation.

The main contributions of this dissertation are the following:

- It presents a library of monitoring structures. When designing a self-adaptive SoC the architect needs to define the different events (and sources of these events) which the system have to cope with, the sensors to perceive these events, the means to store, transmit and process this information and the policies and reaction mechanisms exploiting the monitoring information. One critical objective of the developed library is the design of efficient monitoring units to deal with intelligent on-the-fly transformations of the captured monitoring data and handling of large amounts of monitoring counters. The design of low-level sensors and calibration is beyond the scope of this dissertation.

- It addresses the challenge for designing a system-level Monitoring-on-Chip (MoC) architecture in multi-core SoCs. Continuous improvements in integration scale have fueled the integration of several processing cores on the same chip, with huge numbers of events and amount of monitoring information to handle. In this context sharing and pre-procesing of monitoring data are investigated and monitoring methods and architectures are proposed to efficiently solve these crucial issues.

- It describes a monitoring framework that is fully automatic and flexible. The framework is also scalable, and additional objectives such as power consumption can be added easily. It introduces a novel infrastructure to address the run-time monitoring of multi-core SoCs in a non-invasive way. This monitoring framework allows for performing on-line optimizations of

the architectural and software parameters through providing programmable hardware agents.

- It describes the implementation of a monitor-aware computing platform. To address the cost-effective issue of monitoring-conscious computing a multi-core system-on-FPGA is designed that integrates pro-active policies for power and temperature management. The focus in the proposed system extends the concept of embedding intelligent monitoring units to mechanisms that reduce power and thermal gradients through using distributed sensors. By exploring the efficiency of various distributed heuristics which achieve accurate and fast predictions the objectives include run-time performance optimizations.

1.3 Thesis Organization

The rest of this thesis is organized as follows.

Chapter 2 presents an investigation and categorization of on-chip monitoring methods and corresponding system-level services which involve a number of trade-offs from architectural point of view, including communication protocols and software interfacing, interaction and interoperability. Among the objectives is to understand the major domains in modern ASICs that require or benefit from dynamic monitoring techniques and identify the needs and resulting impact when bringing hardware monitoring units in assistance to software agents.

In chapter 3 we present monitoring methods and architectures in the context of Networks-on-Chip. We introduce the concept and challenges of monitoring for NoCs along with an analysis of the different layers and approaches to integrate monitoring methods. Then, we discuss implementation tradeoffs of monitoring in NoCs, such as cost and the effects on the design process. NoC monitoring systems and services can aim at different objectives; in order to be efficient though programmability, cost, interfacing and intrussiveness need to be addressed. This chapter discusses these aspects to provide insight as regards the issues that emerge in future NoC architectures.

As today's multicore chips exhibit significant diversity at the architecture and the programming level, it is required to develop monitoring components to match these needs. Chapter 4 describes

the design of various monitoring units, ranging from essential counter-based probes with filtering capabilities, to multi-probe components that maintain arrays of counters for managing hundreds of events. This is deemed mandatory for modern complex SoCs in order to support system-wide monitoring services. Additionaly, chapter 4 presents system-level monitoring structures and system architectures that are monitoring-conscious, discussing potential solutions on sharing and scaling of these components. As the key enabling factor of modern SoCs is the integration of tens or hundreds of IPs in a single chip, another trend that is becoming more common in integration of monitoring functionality is distributed development model. Monitoring services will continue to form and evolve based on domain of expertise and efficient communication of the captured information.

In chapter 5 a system-level monitoring conscious platform is presented. We discuss a multicore architecture that supports coprocessor accelerators per node, drawing kernels power profile on-the-fly, while balancing power and thermal effects at the NoC level. Overall, these techniques analyze a system exploration methodology for run-time adaptive computing. It demonstrates as a proof of concept a monitoring subsystem using actual sensors and a custom communication protocol for developing distributed adaptation policies. Different prediction mechanisms are investigated on top of a multi-FPGA emulation platform showing promising results with a minimum complexity overhead.

Finally, Chapter 6 provides a summary of this dissertation giving a glipse of the future towards monitoring-aware SoC designs. In the upcoming years it rises clearly an emerging need for synergistic design of monitoring subsystems and decision mechanisms for self-optimising computing systems. New circuits and methods are required to balance the use of resources and their cost for the system, and therefore constraining the system to use efficiently only the necessary resources to accomplish the applications goals.

Chapter 2

Taxonomy of On-Chip Monitoring

2.1 Introduction

AGGRESSIVE downscaling of device sizes and ever increasing integration densities result in a fast growing number of processing and storage components in a single chip. This trend gives rise to systems with rapidlly increasing complexity, as it is well reflected in Systems-on-Chip (SoCs) combining multiple Intellectual Property (IP) cores. Running modern demanding applications on a multicore SoC requires matching the needs of every application with the offered system platform services. Run-time adaptive mechanisms are increasingly utilized to address this challenge and achieve key requirements, such as energy efficiency, quality-of-service, predictability, reliability or scalability, with respect to both the hardware and software components of the SoC. Observation and collection of the essential pieces of information plays a central role in creating and using these mechanisms. Designers commonly place custom, application-specific observation probes at various locations in a platform, as depicted in figure 2.1, to dynamically capture critical events, to calculate statistics, or to achieve error-resilient processing and communication. While performance, energy consumption and other important metrics are already managed as optimization goals at component level, the growth of parameters in large multicore SoCs with processors (CPUs), memories, hardware accelerators (ACC), and graphic processors (GPUs) that communicate with a

Network-on-Chip (NoC) makes observation and processing of gathered statistics at system level an important challenge.

Figure 2.1 depicts an example of a multi-core system with hardware observation probes (also interchangeably called *monitors* (M) throughout this work).These are located at critical positions in order to collect the desired data, which are forwarded to a monitor manager, or directly to a processor. The monitor manager must then evaluate the collected information and provide a decision on the basis of pre-defined criteria. In addition, platforms include software agents for less critical or system-level events. Usually, the operating system controls and coordinates these agents dynamically, while striving to reduce side-effects such as performance or energy overheads and perturbations.

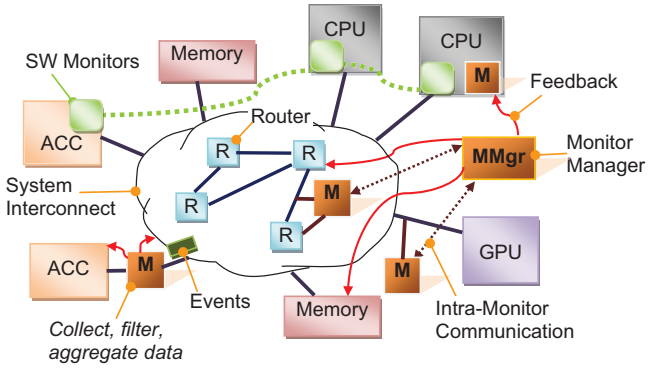

Figure 2.1: Monitoring components in modern systems-on-chip involve hardware probes (M) and monitor managers (MMgr) that cooperate with reactors to optimize system performance metrics, energy efficiency, or reliability. The monitoring process consists of events generation, collecting the monitor data of interest, processing these data to decide and providing the right feedback for adaptation

A monitor manager may also be formulated as a software service at the operating system or

application level, by defining ways to collect and aggregate various events together with reaction policies in order to address potential hazards or optimize system performance. Nevertheless, there is hardly any standard in these methods and in monitoring in general. One of the reasons may be due to that an "event" is not a well defined term; different observers can describe the same event in different terms, and indeed different observers may assume different sources of the cause, or of the location, or of the time of the event, a simple form of the *Rashomon effect* [12]. The subjectivity in observing an event may be due to the different properties of an event that can be of interest to the observer.

Overall, modern SoCs with processors, accelerators, graphic processors, memories and potentially custom units (see figure 2.1), may embed monitoring units in either integral way (as dedicated circuits, and/or as software threads), or as external components. The underlying motivation for employing monitoring is to compensate deficiencies of pre-fabrication simulations, or as countermeasures to dynamic (and unpredictable) environmental changes, or to workload changes. On top, the integration of many heterogeneous components causes complex interdependencies and introduces sources of non-determinism, that often lead to the activation of subtle faults. The challenging process of multicore SoC monitoring is analyzed and discussed in the following sections.

2.1.1 Focus of this Work

This chapter presents a classification of *monitoring* techniques highlighting the distinctions between major methodologies and revealing underlying relationships as well. The primary emphasis throughout this work is on capturing the functional behavior of monitoring architectures, hardware and software, rather than the exact way in which they carry out specific tasks. The intention of this survey is to shed light on attributes, advantages and sources of complexity for different monitoring strategies pairing the suitable technique with the corresponding problem domain as figure 2.2 depicts. Moreover, this work tries to clarify how the monitoring process differs for processors, Networks-on-Chip and memories, or, how effective and what is the appropriate strategy to bring understanding to applications executing on multiprocessors at various levels of abstraction or

virtualization.

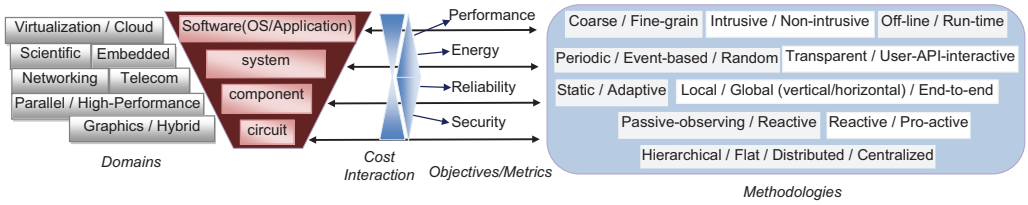

Figure 2.2: Monitoring for systems, a wide spectrum of methodologies endorsed for various different objectives; the versatile concept of monitoring has been employed in many domains and disciplines to assist in cost-effective tackling of various optimization or operational problems inside modern SoCs

Different methods and strategies outlined in figure 2.2 undergo distillation in the following sections to extract the essence of effectiveness in employing monitoring. Finally, although the primary focus is on the component and system-level, we also give circuit examples and references to innovative application-level techniques.

2.1.2 Significance of Monitoring in the Multi-core Era

Emerging multicore architectures are becoming more diverse both at the architecture and at the programming level, while varying application requirements at run-time compose a spectrum where optimum performance, predictability, fault diagnosis and recovery, combined with power efficiency during operation require radically new design methodologies. To address these needs, monitoring and runtime control of the state and dynamics of the entire system are becoming increasingly important. Application monitoring and steering derive their value from their use in understanding and optimizing application behavior and in permitting developers to explore code characteristics that are not easily understood. At the same time benefits from system-on-chip monitoring involve

enhancements of system flexibility to adaptively match system resources on-line to dynamic workloads, and improvements in energy efficiency and desired performance under power and thermal constraints. Power constrainted environments have completely changed the approach to SoC design. Since the highest performance implementations dissipates too much power [13], performance is becoming a secondary concern for most designers. Additionally, as the scale and complexity of systems continues to increase, monitoring the system at run time offers better performance, scalability, and flexibility for multicore designs. Monitoring mechanisms and tools help architects keep pace with new software, potentially using the insights gained to develop fast, robust, representative microbenchmarks for simulation based studies or to design systems accelerating a variety of new applications and usage models.

Therefore, a number of NoC-based multicore systems on a single chip have emerged that bring into focus novel dynamic environments making feasible a wide range of modern applications. To overcome scalability in complex SoCs along with global synchrony and verification costs, while achieving separation of functionality from communication, the network-on-chip paradigm has been introduced [14]. A Network-on-Chip is an on-chip point-to-point distributed interconnection network that the key communication method is to implement interconnections of different IP cores using on chip packet-switched networks. Increasing the NoC's communication observability at runtime is a big challenge. Moreover, congestion or deadlock may appear that needs online monitoring and management, as NoCs become more sophisticated and can even perform adaptive routing algorithms. In industry, complex multicore systems embed tens or hundred of processors, usually in a tiled organization, which adopt Network-on-Chip infrastructures as communication layer. For instance, Tilera TILE64 [15] is built around the iMesh which consists of five 8×8 meshes, where traffic is actually statically divided across the five meshes. Intel TeraFLOPS chip [16] consists of an 8×10 mesh, which runs at an aggressive clock of 5GHz on a 65nm process. Nvidia's Tegra 3 [17] integrates generic and customized processing units to handle demanding workloads dynamically.

These pioneer chips allow user processes or threads to communicate flexibly and dynamically, and at the same time require hardware and software dynamic management, for instance to control

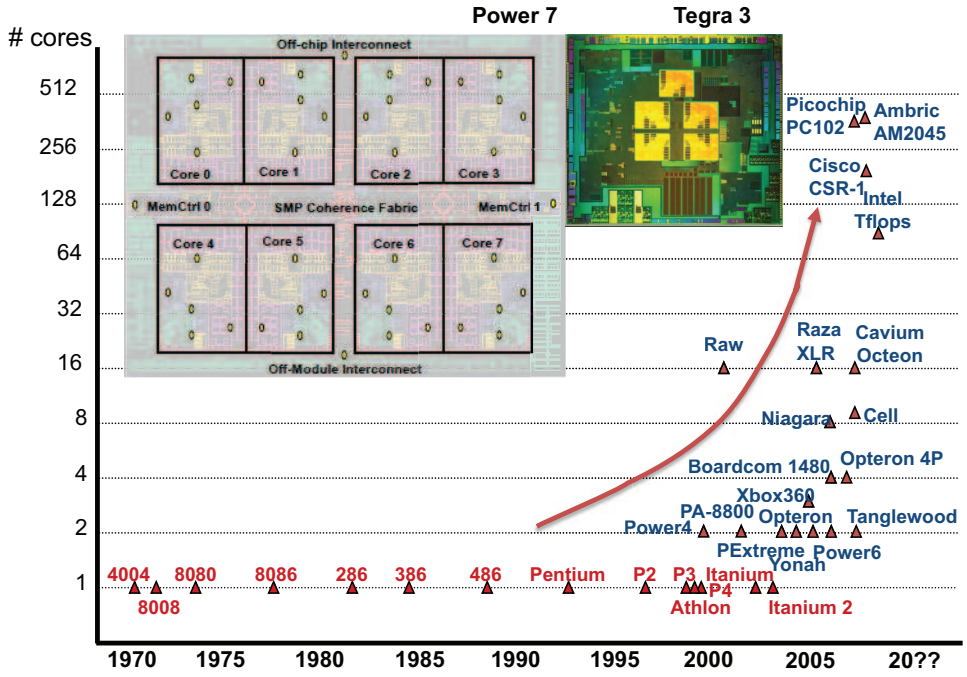

Figure 2.3: Exponential growth of multiprocessor SoCs in the recent decade challenge the design of single-chip systems. Chip layouts are adapted from IBM's Power7 chip [Floyd et al. 2011] and Nvidia's Tegra3 [Nvidia Tegra 3]

operation within the thermal envelope of the chip. For instance, in Power7 chip[18] shown in figure 2.3, critical path monitor circuits combined with thermal sensors provide real-time feedback on the chip's current timing margins. These in turn are used to achieve performance goals through featuring per-core frequency scaling with available autonomic frequency control and per-chip automated voltage slewing. In particular, Power7 chip includes 44 digital thermal sensors (DTSs) using band gap diode voltage comparators and polynomial curve fitting to help the firmware produce the temperature values without costly computations. Additionally, sensor communication is optimized

through packing multiple sensors into single read operations. Nvidia's Tegra3 implements monitoring of its variable Symmetric Multiprocessing (vSMP) technology[17] by individually enabling all five CPU cores (via aggressive power gating) on the basis of the workload.

Figure 2.3 depicts the growth trend for today's multicore designs; in these large systems, finding common architectural methods that address both the system diversity and the dynamic behavior of different applications is extremely challenging. This work demonstrates a taxonomy for ideas and techniques used to monitor and dynamically manage multicore SoCs in the view of an exponential growth of the number of cores. Efficient monitoring, as determined by capturing the correct information at the right time and at the lowest cost in order to fully and precisely understand the root cause of the event, is the key of every monitoring technique. We do not address particular implementations of monitors and circuits, as the primary focus is placed on the strategies for large multicore systems. Response mechanisms are not detailed as well, in the view of endless proposals for optimizations at the circuit, architectural and software level. It is important to note that this survey concentrates on monitoring methods for systems-on-chip; application-specific aspects are considered out of scope.

Close to this work, exploration and classification of various methods and models have appeared in the literature. The growing complexity of modern SoCs has provided a strong incentive for surveying the methods for online failure prediction through runtime monitoring [19] and methods for fault tolerance in the scope of NoCs [20]. Benini et al., has surveyed several approaches to dynamic power management at system-level with a view towards the tradeoffs involved in designing and implementing power-managed systems [21]. Moreover, Kong et al., provided a comprehensive overview of thermal management techniques for microprocessors [22]. So far, in comparison with this work, these works demonstrate little or no concern on the different monitoring methods, the microarchitectural mechanisms to support monitoring for multicore SoCs and the synergies among techniques and across design layers, which is a major contribution of this work. Another contribution of this work is the complete and consistent definition of many terms that are not standardized in the literature and their use to describe the very large space of monitoring methods and techniques.

Section 2.2 gives an overview of basic terms and objectives used throughout this chapter. Section 2.3 develops a functional taxonomy of monitoring techniques designed for processor, chip component and system-level. Additionally, a methodology-based classification is presented in section 2.4 through organizing various approaches for monitoring multi-core SoCs. Finally, open issues and trends appear in section 5.7, which also concludes the chapter.

2.2 System-on-chip Monitoring, Definition and Objectives

Monitoring usually refers to methods that enable observation of a range of phenomena and events and plays an increasingly vital role in design of computer software and architecture. This section gives the definition of key terms used throughout the chapter and consolidating diverse terms found in state-of-the-art system architecture literature. System-on-chip *monitoring* is the process of integrating hardware and software monitor components to adaptively match system resources to dynamic workloads for performance and quality-of-service, to optimize resource utilization, to achieve energy efficiency, or provide increased reliability and dependability through fault detection and recovery (component, communication, power, temperature or soft-error failures). Monitoring involves architectural enhancements, communication protocols, software interfacing, middleware, interaction and interoperability.

The fundamental concept that governs any monitoring process includes observing, identifying and triggering by an event. An *event* is an atomic occurrence in time. From the viewpoint of a system as a finite state machine an *event* can be defined as "a significant change in state" [23, 24]. For instance, an instruction completed event indicates that the registers and processor flags are now updated and that the processor pipeline now has one less instruction in it. For an executing task, an object allocation event means that the system now has less free memory and an additional new object. Some events can have attributes. For example, an object allocation event can have attributes specifying the object size, address, type, etc. Conversely, a cycle event (a clock tick in the CPU) has no attributes. The information that characterizes an event usually consists of: (a) a timestamp giving the exact time the event occurred, (b) a source identifier which defines what the source of the

event is, (c) a special identifier to determine the category that the event belongs to, as well as (d) the information that this event carries. The information regarding the events is called *attributes* of the events, and they consist of an attribute identifier and a value. The exact attributes along with the number of them depend on the category the event belongs to.

Computing systems and sensing devices such as sensors, actuators, controllers, can detect state changes of objects or conditions and create events which can then be processed by a service or system. In this scope, an *event trigger* is a condition that result in the creation of an event. The first logical layer is the event generation, which means the creation of a fact, of a piece of information, anything that can be sensed. Converting the different data collected from the sensors to one standardized data form that can be evaluated is a significant problem in the design domain [25].

Essentially, the monitoring process consists of the following steps (see figure 2.4):

- *Event generation* is caused via periodic sensor sampling or asynchronous event trigger
- *Data capture* involves the transformation of the event (e.g., analog-to-digital conversion) and formatting it to a meaningful representation
- *Event filter* and captured data processing consists of selecting the core information from a large amount of events to store or forward to the decision making component
- *Diagnosis, decision making* involves the identification of the location and root cause of the event that may be filtered
- *A Reaction* policy is activated when the diagnosis results indicate that a deviation from the objectives of the monitoring mechanism has occurred and countermeasures need to be scheduled

Monitoring involves different and often diverging approaches. For instance, the type of events that can be monitored can be categorized *spatially* through the various levels of the system, from application layer, operating system and middleware, to hardware layer, components and circuit level. Monitoring can be employed to identify: (i) system calls, context switches, interrupts, shared variable references, and application-specific events, or, (ii) processor, on-chip communication, memory

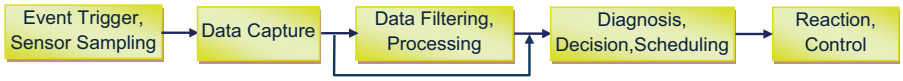

Figure 2.4: Basic components of the on-line monitoring process

hierarchy, and co-processing cores related events. Alternatively, the monitoring process can be based on the objective "why" it is applied, or "how" the monitoring is designed and integrated to the system. Monitoring techniques can span different levels from hardware to software level, albeit mainly requiring the synergy of both.

Monitoring services permit access to system's information at various levels of abstraction. At component level, accuracy is the profound benefit of every hardware monitoring technique which commonly employs counters and memory storage to log important system events. Nevertheless, a limitation of hardware counters is that they depend on the microarchitecture of the processor. While microarchitecture-specific counters can be very useful, if the goal is to understand the behavior of specific components, they do not provide direct metrics when one tries to measure properties of the workload common to different platforms, such as data-reuse patterns.

Additionally, monitoring services can essentially be determined by and defined through the following attributes: access intensity, function parallelism, criticality, dependencies.

Access intensity is the frequency at which the monitor captures an event and generates a notification. These two operations can be distinguished as potential filtering alters the higher level perspective of observations.

Function parallelism of a monitor component is the level of parallelism that can be accomplished from a parallel-based monitor organization to serve a particular function, like power identification or queues utilization of the CPU or of NoC's routers.

Criticality is the level of importance of the service that a monitoring subsystem provides. Often, a system must respect stringent latency constraints, interrupt responses, etc., which rely on the

260pt

$Monitoring\ methods$	$Access\ Intensity$	$Function\ Parallelism$	$Criticality$	$Dependency$
$Software$	L	M	L/M	L
$Hardware$	H	H	H	M/H
$Reactive$	M/H	n/a	H	M
$Pro-active$	L/M	n/a	L/M	M

Table 2.1: Monitoring methodologies covering a mixture of features at different levels, low (L), medium (M) and high (H) are usually employed on basis of the monitoring objective.

monitor components and their communication protocols.

Dependency is the degree of confidence that a collected monitor provides. Monitor values can be simplified to deliver a number of occurrences of an event, like cache misses, while other complex metrics inherently involve reliance to potential or circumstantially insignificant causes. For instance, monitoring thermal effects on a system component should possibly include sensor's leakage currents when operating in high temperatures.

Generally, designing monitoring services involves locating the right set of features such as the ones listed in Table 2.1. It is true though that most adaptive SoCs use application-specific approaches to observe and manage a system dynamically. In order to characterize and understand the appropriate monitoring strategy suitable for a particular system the following sections provide more insight through various solutions.

2.3 Functional Taxonomy

Functional classification focuses on organizing the properties of different methodologies on the basis of the functions or objectives of monitors: debugging, performance, quality-of-service (QoS), resource utilization, power, energy and temperature, fault-tolerance, security and other application specific goals such as dynamic reconfiguration of modern applications or monitoring to assist code migration in a cloud environment. These objectives essentially form the constituents of two major

directions in system development: providing the methods to enable the correct operation of a system and, enhancing it in order to improve its distinct characteristics such as performance or energy. It is important to note that system's robustness against soft or hard errors or deadlock avoidance belong to the first direction, while optimizing quality of service of the on-chip interconnect fabric involves monitoring for the latter consideration.

Additionally, when designing monitor components these goals are not disjoint. For instance, monitors developed for increased reliability, or for security must respect the chips energy constraints and preserve the performance levels intact. In general, dynamic applications with varying behavior in time or in space, as well as general-purpose processing components require monitoring mechanisms of increased complexity. This observation drives developers to provide basic hardware primitives to monitor particular functions and build monitoring services on top of these primitives. Next, we discuss selective approaches and attributes that characterize each category.

2.3.1 Monitors for Debugging

A debug methodology involves observing the (hardware or software) component to debug in its target environment, and controlling its execution (stopping, single stepping, etc.) to efficiently and effectively locate the root cause of any undesired behavior. Monitoring and debugging computation, especially of a single processor, is a mature area for which tools and techniques have been developed [26, 27, 28]. In industry, various solutions are employed; Xilinx developed Chipscope tool [29] to provide on-chip debug and real-time system visibility for reconfigurable platforms. Embedded processor paradigms such as ARM CoreSight [30] and MIPS on-chip instrumentation [31] technologies feature non-intrusive monitor modules integrated within the processor cores, providing logic analysis for AMBA AHB, OCP, and Sonics SiliconBackplane bus systems. Moreover, the IEEE's Industry Standard and Technology Organization has proposed a standard for a global embedded processor debug interface, called Nexus 5001^{TM}[32]. Nexus defines four classes of operation: Class 1,2,3, and 4. Higher numbered classes progressively intend to support more complex debug operations at the cost of increased on-chip resources.

There are several hardware approaches for debugging. If the erroneous behavior is investigated during the development of a device, then searching for bugs in silicon is also referred to as *post-silicon validation and debug*, or just post-silicon validation. Design-for-debug (DFD) techniques are used to localize functional bugs through logic probing, scan chains, and real-time trace collection, commonly referred as *embedded logic analysis*. Various approaches address debug support framework and interfaces standardization for SoCs [33], and debug structures for instrumentation, such as assertion-based on-chip debug checkers, triggers and event counters[34, 28]. In particular, typical post-silicon validation techniques apply recording values of the circuits internal signals into an on-chip trace buffer, feeding the obtained value into a simulation framework to reproduce the erroneous behavior [35]. Due to the vast amount of debug data, it is more efficient to record higher level information, such as instruction footprints for processors[36].

In addition, infrastructures for Networks-on-Chip have addressed debugging issues [37], while earlier El Shobaki [38] presented MAMon for event-based debugging of SoC applications, and Cota et al. [39] described a Test Access Mechanism (TAM) for observing and testing the nodes of a SoC through re-using the network resources to minimize the cost and improve the speed of testing probes.

If on the other hand a “mature” system exhibits unpredictable or unexpected problematic behavior, then debugging is still required, with the goal nonetheless placed on reaching sufficient understanding of the deficiencies (hardware or software) that escaped the developers. Flight data recorder [40], Karma [41] and Rerun [42] are hardware approaches that use a low-overhead hardware recorder in the context of caches or cores, even on deployed systems, essentially to log the minimum thread ordering information that is necessary to play back the multiprocessor execution faithfully after the event. HARD [43] and NUDA [44] provide hardware-assisted race detection approaches, using a non-uniform memory mapping. HARD design is architecture-dependent, while NUDA offers a distributed and simultaneous notify mechanism for debugging control.

Figure 2.5 summarizes the main directions towards embedding special circuitry and mechanisms for debugging and desired goals for each direction.

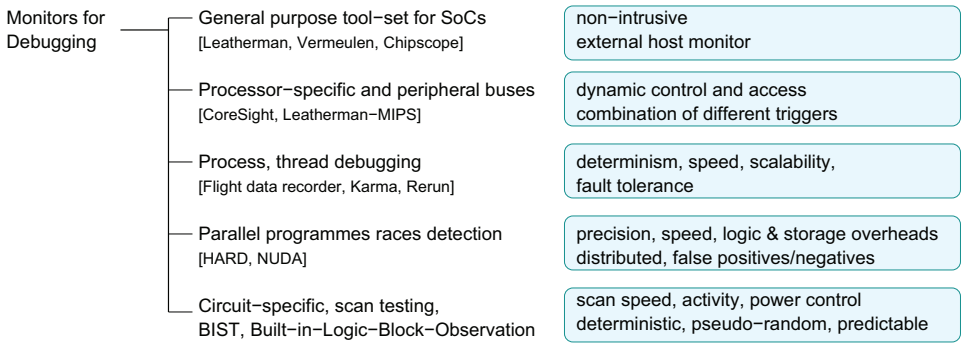

Figure 2.5: Sub-categories of monitors for debug principally relying on hardware support

Even though the power of simulators continues to improve in a linear fashion, the inability to adequately simulate all the possible permutations in pre-silicon has led to alternatives such as emulation and FPGA-based prototyping before design completion. Chip instrumentation is the key innovation to enable more efficient observation and verification of sustained functional integration that is combined with faster internal clock speeds and complex, high-speed I/O. Moreover, runtime debugging is also becoming increasingly important for multi-core systems, especially for detecting race conditions or deadlocks, requiring architectural enhancements and tools support.

2.3.2 Monitors for Performance

The most widely known monitor structure is the performance counter, guiding computer architects and programmers through a challenging spectrum of complex systems and evolving applications. A *performance monitor* observes the behavior of a system, collecting throughput and latency-related statistics which help the run-time system to optimize its weaknesses. Hardware performance monitors are often available inside processors, while additional software performance monitors may be developed inside native libraries or virtual machines like the Java libraries. A performance monitor is a scalar variable with a value that changes over time, while a performance counter is a special

kind such monitor. The value of a performance counter reflects a count, usually a count of events.

Performance counters usually refer to a class of hardware devices contained in most modern microprocessors. These counters can be programmed to count the number of times specified (dynamic) events occur within a processor. One common such event is the number of instructions committed since the counter was enabled. Through exposing these counts to software, they can be used to find and remove software inefficiencies [45], or architectural bottlenecks [46]. Performance counters can easily track the number of events that occur within a processor, but it is difficult to use them to find which instruction causes a particular event. One common method is to set the initial value of a counter to a negative number and take an interrupt to the kernel when the counter turns to zero; then, the state can be observed from within the interrupt handler. In a different approach, the *ProfileMe* framework randomly samples individual instructions and collects cycle-level information on a per-instruction basis [47]. The idea is to provide more accurate data for out-of-order CPUs, since other instructions in the pipeline can affect the previously started counters. Since counters in ProfileMe do not record the number of events triggered but only their presence, it is the Operating System's task to record the data and provide the correct amount to the user.

Conventionally, performance counters have been targeted at internal processor core events, and only recently support is extended to system-wide events [48], even in parallel multiprocessors such as Blue Gene [49]. The number of hardware counters in a processor is much smaller than the number of events that can be measured. This can impact the accuracy since the more events that can be tracked the more accurate the performance model built. Hardware vendors have increased coverage, accuracy and documentation of performance counters making them more useful than before. For instance, hundreds of events can be monitored on a modern Intel chip[50], representing a three-fold increase in a little over a decade. Despite these improvements, it is still difficult to realize the full potential of hardware counters because the costly methods used to access these counters perturb program execution or trade overhead for loss in precision. Programmability, parametric, dynamic adaptation, simultaneous activation and monitoring of multiple performance counters are still difficult to achieve, mainly due to increased complexity.

2.3.3 Monitors for Quality-of-Service

The design of modern SoCs involves integration of many different components (IPs) such as processors, graphical accelerators, audio/video decoders, on-chip memories, that cooperate to achieve the overall performance needs of different applications. Usually each one of these IPs has drastically different bandwidth and latency requirements, and these requirements must be guaranteed to achieve proper operation. Monitoring subsystems are increasingly employed in such multicore SoCs to effectively manage allocation of on-chip resources. One of the primary objectives of monitors for quality-of-service includes the interconnect QoS which is responsible for providing a suitable set of services to satisfy these requirements. Service class, virtual channel and dynamic bandwidth controllers have been proposed [51, 52, 53] to provide efficient Quality-of-Service in advanced SoCs. One use of such mechanisms involves to observe and control the number of priority tokens on a given connection, and thus its real-time properties to provide connection oriented guaranteed services. For example, figure 2.6 shows monitor units located in NoC routers, which monitor congestion by estimating link utilization [54], or counting how many packets are stored in buffers. When thresholds are surpassed, a notification is sent to a controller or a traffic shaper to adjust the packet generation rate [51]. In a different QoS-aware monitor scheme, adaptive routing/path allocation algorithm or on-demand buffer assignment can be applied for adaptive NoCs where the subset of tasks and their mapping may change during run-time [55, 56, 57]. Monitors are responsible to track spatial change of task execution, that is, relocation of application tasks to different processing elements, as well as the possible introduction of new tasks and reconfigure the NoC routing algorithm.

Many modern embedded and distributed systems (including real-time and non-real-time) employ utilization monitors, rate modulators, model predictive or workload controllers with feedback control techniques in order to provide quality-of-service [58, 59]. In response to workload variations in unpredictable environments, dynamic resource allocation is needed to achieve high processor utilization while still meeting real-time constraints, to enforce appropriate schedulable utilization bounds, or to handle system dynamics caused by load balancing for large-scale server clusters.

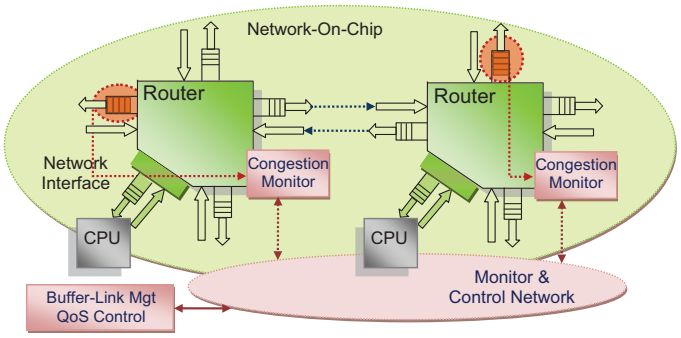

Figure 2.6: Distributed approach for QoS control of a NoC through monitoring buffer or link utilization

From a different angle, monitors help to avoid saturation of processors, which may cause system crash or severe service degradation. However, these management approaches based on feedback control require having an accurate system model, which may be difficult to obtain for realistic distributed or multi-core embedded systems.

Monitoring for sustaining QoS can be inherent in particular application domains. For instance, servers providing adaptive video streaming service exploit the inherent adaptiveness of video applications to perform controlled and graceful adjustments to the perceptual quality of the displayed MPEG video stream in response to fluctuations in the QoS delivered by the underlying infrastructure. Increased efficiency though is attained when the monitor service is not platform-agnostic; in embedded system design where the platform cost and its resources are bounded, quality monitoring and control assisted by the system and in synergy with the applications is inevitable [60].

2.3.4 Monitors for Power, Energy and Temperature

As the power densities increase with the continuous shrinking geometries, thermal hot-spots and large temperature variations on the die may occur, and must be avoided or at least mitigated. Monitor mechanisms have a fundamental role when building adaptive chip architectures that struggle to

achieve the theoretical highest performance within a thermally-safe dynamic voltage and frequency (DVFS) configuration for specific or different workloads.

The past decade has witnessed many research efforts in this domain since the design space exploration for run-time power and temperature management involves a very large number of parameters. Figure 2.7 shows a state of the art summary of approaches for a single processor and some in the multicore era [61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74]. The main goal is to fit the best monitoring strategy (fine or coarse-grain, at transistor, RTL and architecture level technique, or compiler, application and OS level) in terms of efficiency, to the right chip, system, and environment: CMPs, heterogeneous multicores, NoC-based systems, multi-threaded environment, high-performance processors, multi-level memory hierarchies, embedded systems, real-time, or, embedded OS.

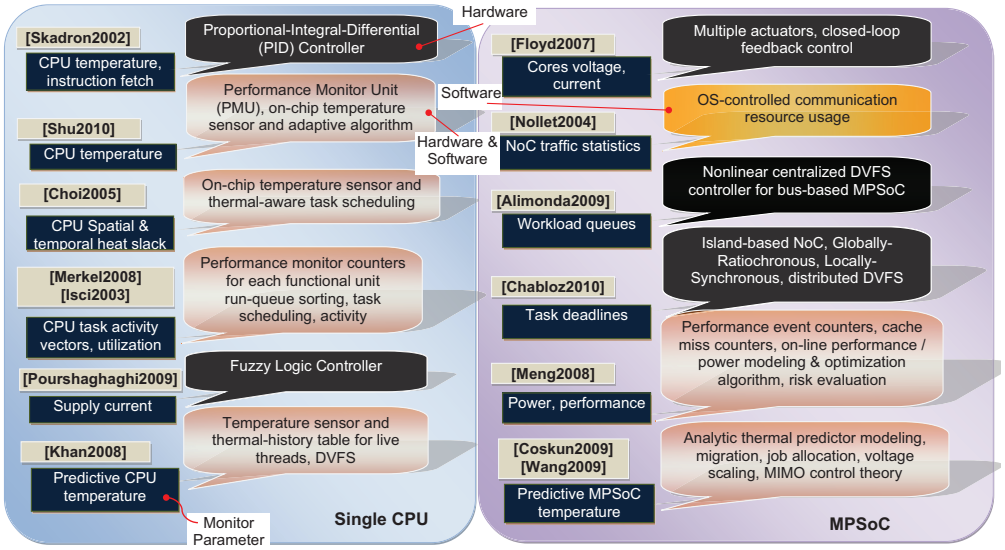

Figure 2.7: Monitor-based approaches for energy and temperature management for single- and multi- processor systems

An intrusive, or non-intrusive design approach involves exploiting the performance counters that exist in most processors, which can be used to monitor activity data (access count) of most on-chip functional units and therefore allow interpretation of these counter values also for localized temperature sensing across a microprocessor [65]. Through performance monitoring counters the functional units utilization can be determined as a way to infer energy requirements and corresponding temperature [75]. Accurate monitor information is of prime importance to develop *task activity vectors*, as defined in [64], as a metric for characterizing a task by the functional units it uses. Then, the scheduler of the operating system considers this metric, to arrange, for example, the tasks in a processors run queue in a way that tasks using complementary resources are scheduled successively and thus reduce hotspots.

In a different perspective, monitor controllers are utilized in feedback control theory based approaches that tune the system performance while using predictive lookup tables for forecasting temperature and workload dynamics, and applying proactive thermal management techniques [61, 73, 76].

In addition to performance monitor counters and adaptive feedback controllers that are exploited for energy and thermal system management, other types of monitors are also integrated in modern SoCs, such as thermal, delay, or wearout monitors. Most thermal sensors are build by simple ring oscillators or diode-based circuits. Arrays of voltage and thermal sensors are interconnected with on-chip controllers to capture chip environmental conditions. Monitor information is then used to control system clock frequency, voltage and bandwidth allocation [77, 78, 68].

The majority of power and temperature-aware monitoring schemes have both hardware and software components. Two important parameters involve the design of low-cost sensors and their strategic distribution throughout the chip. At the same time, monitoring alone is not the only challenge as it is tightly connected to the system's capacity to react efficiently. Modern processors commonly utilize instruction-level-parallelism (ILP) techniques for mild stress situations and DVFS for emergencies [79]. Therefore, they can provide reconfiguration capabilities for localized throttling and reduction in capabilities of resources such as queues, buffers and tables, as well as the

ability to reduce the width of the major components of the machine such as, fetch, issue and retirement units. Software on the other hand, running at the OS level or as a Virtual Machine Monitor (VMM), should minimize the interaction between the management method and the SoC and reduce the number and duration of reaction periods or time spent in a thermal crisis, while increasing the system performance. A software monitoring layer can maintain thermal history tables for live threads in the system [67], and invoke predictive DPTM policies or allow the operating system to control CPU activity on a per-application basis, to perform task migration, or to manage the SoC communication[69].

Software monitors allow designers to save silicon area by reducing the number of hard sensors and their controllers without sacrificing the tracking accuracy. Moreover, the exact location of hot spots may not be known a priori at manufacturing time or depend on workload behavior. Thus, hard sensor temperature indications can be weighted through software monitors to consider runtime behaviors and physical distance.

2.3.5 Monitors for Fault Tolerance and Reliability

Ensuring reliable systems involves multiple abstraction layers from circuit and micro-architecture up to algorithm and application layer and thus various monitor methodologies have been developed across these layers to provide runtime self-diagnosis, adaptivity, and self-healing. The majority of all mechanisms assume an *asymmetric reliability*, in a conservative view that some components are almost fault-free and thus these mechanism need to protect individual system components like buses, Network-on-Chip, memories, datapaths etc. against transient or permanent errors. Additionally, monitoring and diagnostic units have modest performance requirements, so they can operate at low voltage and frequency, and at the same time they usually use aggressive built-in redundancy, making it effectively immune to failure [80]. However, centralized monitor schemes represent a dependability exposure due to their single point of failure vulnerability.

On-line monitors intend to extract metrics for accurate determination of aging models or for complete reliability analysis [81], or for providing reliability characteristics of components. Future

architectures will contain components that exhibit varying dependability characteristics, like processor cores with different arithmetic precision and memories with different reliability guarantees. Applications will be allowed to trade-off high hardware reliability for high performance through switching modes in mixed-mode multicore systems with reconfigurable dual-modular redundancy [82]. Moreover, using reliability annotations, compilers can create a mapping that ensures the assignment of reliability-critical code and data objects to the most reliable components of a system, or operating systems can perform proactive, reliability-driven thread migration and shadowing (e.g., shadow threads are assigned to dependable cores with low thermal stress).

Monitor circuits are essential in designing with deep sub-micron technologies for dynamic detection and correction of errors. A combination of *in situ* error-detecting circuits and micro-architectural recovery mechanisms enhance system robustness in the face of various errors [83]. Most adaptive techniques employ tracking circuits, such as tunable replica circuits [84], to compensate for manifestations of voltage droops, of PVT variations or fast-changing transient type of events, such as coupling noise and phase-locked loop (PLL) jitter.

In the context of Networks-on-Chip, monitoring systems can provide communication observability about system events, even at transaction level. Similar to macro-networks, techniques are increasingly developed to provide fault-tolerant on-chip communication through codes (Hamming code and CRC) for error detection and correction and end-to-end retransmission mechanisms [85], or by supporting multi-path communication, adaptive routing and reconfiguration. Monitor probes which are attached to network interfaces or to routers are responsible to provide error detection indications after checking each transmitted packet.

Beyond typical problems with monitor processing in other domains, such as high volume of events, complexity overheads, topological organization, or real-time monitor service delivery, additional points in reliability monitoring involve detecting global or composite events.When tracking events sent from multiple monitor agents, transient faults can result in missed or out-of-order events that negatively impact a predictable and concise global system view. Potential monitor vulnerability to the propagation of faults prevents scaling to large-scale systems. Thus, fault-aware monitors and

monitor managers need to emerge for fault-optimized platforms. Adaptive monitor organizations and traditional mechanisms from distributed systems are promising solutions. Checkpointing and rollback recovery are well known to provide fault tolerance in distributed systems and in multiprocessors [86]. Each monitor domain can save its state on stable storage and, when an error is detected, the execution is rolled back and resumed from earlier checkpoints. To reduce full-state comparison that is costly in terms of bandwidth and storage, *fingerprinting* can provide a concise view of the architectural state of a processor by, for example, a hash-function computation, monitoring the committing instruction results of the processor [87]. These methods along with robust on-chip monitor communication can help to ensure monitoring for reliability.

2.3.6 Monitors for Security

Security forms a separate dimension, next to constraints on area, performance, and power. This domain includes security monitoring subsystems which are the cornerstone components in most system architectures to verify that the processor indeed performs the operations that it was intended to. Anomaly and intrusion detection is usually performed by comparing behavior against a model. A monitoring subsystem operates in parallel with the processor and use per-block hash values, or control flow information to detect deviations in the program execution [88]. For example, custom signature verification units [89] are proposed to assist when fetching instructions from memory. Any attack would disturb the pattern of execution steps and thus alert the monitor.

Architectural monitoring support for security commonly concentrates on implementing tamper-resistance and cryptography. When implementing secure processing and monitoring there are endless choices of what characteristics to monitor. Particularly intuitive patterns for monitoring involve control flow, address and load/store information. Those patterns though consume memory space, and thus hashing schemes are often used. Several pieces of information such as instruction address and instruction word can be compacted to smaller hash values. Monitor processing in this domain can take the form of firewall-like structure that filters unauthorized memory access request, such as a data protection unit (DPU), employed in a network interface on-chip, which is attached to shared

memory [90]. Monitor co-processor units have been proposed, such as a Reliability and Security Engine (RSE) [91], using dedicated hardware modules and logic for detecting various faults including both buffer overflow based and transient faults. When a security threat is detected there may be more than one recovery mechanism to follow: terminate the current process and inform the operating system about the fault by returning a trap signal, or try to continue execution of the program in a safe way.

New programming models have also been developed by leveraging a multicore platform and designing special cores and runtime software monitors to support one or more protected processing components (called resurrector cores) that are insulated from remote attacks [92]. The key idea is an insulated component that provides fine-grained concurrent state monitoring and efficient state backup. Network-on-Chip based systems in safety critical environments require increased levels of defense not only at processor level, but also at system level, like security agents and manager components as a mean to detect and prevent propagation of attacks [93].

2.3.7 Application-specific Monitors

Application-specific monitoring involve mechanisms developed to satisfy any of the objectives discussed so far, but are further customized to consider the behavior of a particular application. For instance, this category includes monitor techniques for on-line exploration of the best performance/energy parameter values that could be customized related to an application's memory usage characteristics. Run-time monitors and managers have proposed for extensible processors with customized instruction set that achieve to utilize the most appropriate special instruction for an H.264 video encoder [94]. The main idea of this online monitoring approach is to estimate the quality of usage of the special instructions that are reconfigured in the processor datapath. The method employs counters to reflect the execution count of these custom instructions for each iteration of a computational-intensive loop.

Moreover, in embedded system development, Application-Specific Instruction-set Processors (ASIPs) have gained popularity as they combine the flexibility of software with the energy-efficiency,

and scalable computational performance of dedicated hardware implementations. Through modifying the development methodology for ASIPs, monitoring can enhance the system observability and its adaptation capabilities. For instance, Ragel et al., [95] describe a methodology for monitoring routines to check insecure operations through the addition of extra micro-instructions within vulnerable machine instructions. ASIPs can be synthesized targeted for different applications, which can employ different security monitoring methods (Instruction Memory Data Bus Monitoring, Data Path Monitoring, Return Address Stack Monitoring and Branch Instruction Monitoring).

In a different perspective, a monitoring approach is considered *application-specific* when it is customized for a specific computational subsystem. Instead of utilizing widely adopted counter-based techniques and sensors, some run-time monitoring methods employ sophisticated circuitry. Gordon-Ross et al., [96] describe a hardware cache-tuning module that non-intrusively monitors an application's memory access patterns and analytically predicts the best cache configuration for those patterns. Then, if the predicted best cache configuration differs from the configuration presently in use, a cache tuner reconfigures the cache directly to the new best cache. Qureshi and Patt [97] developed particular cache monitoring circuits that examine the correlation of the benefit that applications get for an amount of cache with the demand. Then these monitors guide the partitioning of the cache among competing applications. In addition, the utility monitor circuits can be extended to compute utility information for prefetched data or estimate CPI, which can help in providing quality-of-service guarantees. Moshovos et al., [98] examined the potential for filtering remote snoop requests by checking them against a small *Jetty* table to avoid tag lookups and reduce on-chip power consumption induced by remote cache misses. Clearly, various forms of speculation are routinely employed to reduce the latency of cache misses and to overlap data fetch and transmission latency with checking for cache coherency.

At system level, dedicated connections between physical cores, processors and memory increasingly tend to be replaced with dynamic connections between virtual on-chip resources [14, 99]. Thus, adaptive virtualization services can extend interconnect intelligence to the application, enabling fabric-wide application service level monitoring and management that automatically reacts

to changes in virtual system workloads. This approach enables the fabric to dynamically allocate shared resources as changes occur in the data flows between virtual cores. If congestion occurs (or is predicted), the interconnect fabric can adjust bandwidth and other resources according to defined service levels, helping to ensure that higher-priority workloads receive the resources they need.

2.3.8 Summary

In general, monitoring techniques fundamentally change the ways in which the designers and system developers address performance, energy, reliability and other optimization objectives in the multicore domain. New techniques are urgently required to understand the exact robustness, performance, and complexity characteristics of integrating diverse hardware components together with software tasks with varying behavior. Although many researchers advocate static exploration and simulation-based solutions on complexity grounds, run-time monitoring strategies are evolving with an increased dynamism becoming a key part of systems design.

2.4 Methodology-based Taxonomy

Monitoring methodologies can be classified into categories according to the desired level of sophistication, complexity, response time, precision, scalability, cost or other criteria. This section provides a taxonomy of methodologies for monitoring, each method determined at different level by the above impact criteria:

- Monitor Probe Implementation: direct and indirect measurement circuitry,
- Monitor Implementation Technique: software, hardware, or hybrid mechanisms,
- Sampling frequency: continuous, periodic, on-demand, and adaptive (relates to trading overhead for loss in precision or sensitivity),
- Real-time monitoring: logging, transmission and processing of monitor information,

- Intrusiveness: intrusive and non-intrusive, whether the monitor behavior can perturb program execution, or the data flow,
- Organization: hierarchical, centralized and distributed monitors,
- Reactivity: reactive and pro-active management
- Biologically-inspired monitoring for adaptive systems-on-chip

In this section we give an analysis of various monitor strategies and discuss the scope and reasons for adopting each methodology with the resulting impact.

2.4.1 Direct Monitoring Circuits

Monitoring through various-purpose on-chip integrated sensors has slowly been developing for more than twenty years now but has found renewed interest as designers are looking for new advanced ways to construct analog and digital circuits in fine-line CMOS processes. Sensors and monitor circuits are advancing to address traditional challenges in VLSI chips such as signal integrity quality, power supply noise, along with emerging ones due to evolving nanometer processes. Nowadays, process variation effects have become more acute, the sensitivity of device lifetime to operating conditions has increased, and hundreds or even thousands of sensors are required to estimate bounds on overall chip performance degradation, or physical characteristics of the chip. Intelligent designs improve not only the analog or digital features of the sensors, but additionally the associated direct or indirect impact of their utilization, such as the number and on-die location of the sensors, the costly calibration stage, or the consumed silicon area.

Figure 2.8 shows a traditional digital thermal sensor (DTS) using a thermal on-die diode (i), and a fully digital delay-based CMOS temperature sensor that converts inverter delays to digital temperature outputs (ii). A number of circuits of different type are also developed to sense different issues and can be summarized in: temperature, current, power or leakage power sensors, signal integrity loss (or quality) sensors, power supply noise or noise detector sensors, jitter sensors, process variation sensors (including aging, reliability, NBTI, gate oxide degradation sensors) edge or transition

detection and delay sensors.

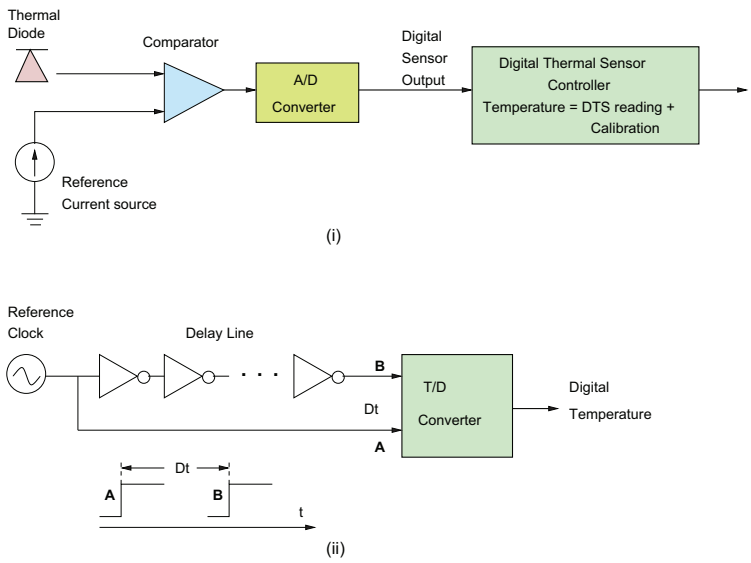

Figure 2.8: Digital temperature sensors with (i) a thermal diode, analog-to-digital (A/D) converter and calibration, and (ii) a delay-locked loop using inverters and a time-to-digital (TD) converter

The following list outlines novel research works linked to sensor developments along with advanced associated monitoring techniques.

- Novel methods and solutions involve on-chip *reliability* monitors and sensors to address the effects of aging process, the negative/positive bias temperature instability (NBTI/PBTI), the hot carrier injection (HCI) and time-dependent dielectric breakdown (TDDB). Kim et al., [100] develop a beat frequency [1] detection scheme to capture the effects of both DC and AC stress signals to NBTI aging. They utilize fully digital differential measurements based on

[1] *beat frequency* is the difference of the two oscillator frequencies which is attributed to the V_{th} shift due to NBTI.

free-running ring oscillators, with minimal calibration and sub-picosecond sensing resolution ($<$0.02% or $<$0.8ps) for a 4 ns period ring oscillator. Singh et al., [101] propose compact sensors for devices undergoing NBTI and defect-induced oxide breakdown. Common practice using ring oscillators is replaced with a voltage controlled delay line. The oxide degradation sensor monitors the change in gate leakage under stress conditions. The innovation lies on small size, low-power sensor designs, since compact sensors can be implemented in large numbers to collect high-volume data on device degradation.

- Bull et al., [102] develop timing *transition-detector* sensors to track errors due to dynamic variations: both slow-changing variations such as thermal hotspots, and fast-changing variations such as Ldi/dt droops, capacitive coupling and PLL jitter. They employ the *Razor* technique and the designed sensors to optimize performance by speculatively operating the processor without the full timing margins and achieving 52% energy savings while enabling runtime adaptation to PVT variations. To monitor circuit-level performance within the presence of dynamic parameter variations similar techniques are emerging such as the all-digital dynamic variation monitor (DVM) [103] that contains a tunable replica circuit, a time-to-digital converter, and multiplexers to measure circuit delay or frequency changes while capturing clock-to-data correlations.

- Sensor circuits to detect *signal integrity* undershoots and overshoots are proposed by Champac et al., [104] showing an adapted multi-signal monitor with area cost that is distributed among the signals to test. The proposed sensor is based on a cross-coupled differential amplifier and uses coherent sampling to obtain the information of the signal under test.

- Ha et al., [105] demonstrate a new on-chip *temperature* sensor that operates with simple, low-cost one-point calibration, while it uses delay-locked loops (DLLs) to convert inverter delays to digital temperature outputs. The use of DLLs enables low energy (0.24 J/sample) and high measurement bandwidth (5 kilo-samples/s), facilitating fast thermal monitoring. Trading energy, linearity and resolution for small area and conversion rate researchers incorporate more

elaborate calibration techniques and/or higher precision circuits. For example Chen et al., [106] describe a higher resolution but slower temperature sensor composed of a temperature-dependent delay line (TDDL) to generate a delay proportional to the measured temperature, combined with a binary-weighted adjustable reference delay line which is controlled by a successive approximation register-based scheme. A low-cost alternative for a temperature-sensitive timer is based on a four transistor (4T) DRAM cell [107]. The time it takes to discharge because of leakage is a measure of temperature.

- Instead of measuring the voltage drop across the package resistance to estimate power consumption of the entire chip, due to limited spatial resolution as block-wise power consumption cannot be calculated, new methods are proposed to sense power. For example, a real-time on-chip power sensor with high temporal and spatial resolution can utilize the voltage drop across a sleep transistor to generate a current proportional to the load current, which is used to charge a capacitor and reset it at different speeds [108]. The pulse density of the output waveform is proportional to the load current.

Major challenges in efficient integration of on-chip sensors involve sensor imprecision and reading corruption by noise. Multi-sensor arrangements reduce such effects but the sensor placement error and the cost of adding a large number of sensors can be prohibitive. Thus, techniques have been proposed for optimizing sensor placement [53], or for providing accurate temperature readings on a given chip while maintaining a reasonable overhead in terms of sensor data collection. The latter can be achieved by interpolating to reduce the average errors for a given distribution, and by a dynamic selection method using only a relatively small fraction of embedded sensors per core to provide reasonable accuracy [109]. To this end, Zhang et al., [110] propose a statistical scheme for reading noisy sensors that can be implemented with very low overhead either as a hardware module or as a software kernel in operating system. Instead of relying on temperature sensors that are vulnerable to noise and process variation, indirect methods have been proposed on the basis of *model-based* estimation approaches as discussed in the following section.

2.4.2 Indirect Monitoring Circuits and Techniques

Performance counters are the most common monitoring structures utilized to infer that the power behavior of a particular chip component varies accordingly and henceforth also construct the thermal profile of the component. The collection of architectural statistical information, such as cycles per instruction, cache misses, memory references and on-chip communication bandwidth utilization, together with application-level metrics and patterns can be exploited to optimize system performance, performance-per-watt, energy and temperature requirements. The indirect monitoring approach usually involves building a power model for a processor [65], or for the GPU architecture and the GPGPU kernels [111]. The following equation gives the fundamental power model as introduced by Isci and Martonosi [65].

$$Power = \sum_{i=0}^{n}(AccessRate(C_i) \times ArchitecturalScaling(C_i) \times MaxPower(C_i) + NonGatedClockPower(C_i)) + IdlePower$$

It consists of the idle power plus the dynamic power for each hardware component (C_i), where the *MaxPower* and *ArchitecturalScaling* terms are heuristically determined. For example, *MaxPower* is empirically determined through running training benchmarks that stress the desired architectural components. Access rates are obtained from performance counters.

Monitoring is also used through employing predictive functions to model the evolvement of particular system behavior for indirect control of performance or power. For example, Wang et al., [112] shows a simple history-based prediction approach to determine the shader resource requirements of the next frame in order to apply power gating mechanisms to save leakage power in a GPU. Mochocki et al., [113] observed imbalances among Geometry, Triangle Setup, and Rendering stages to motivate the use of DVFS, and they proposed a signature-based workload prediction scheme that estimates the next frame's workload from the frame history and attributes of the current frame (e.g., the triangle count and the triangle size).

2.4.3 Software Monitoring

The *software monitoring* concept addresses the development of monitoring methodologies (though hardware or software instrumentation), to resolve software issues or provide insights into application behavior and to optimize its performance. Software monitoring determines the software constructs, objects, data structures, tools and techniques used to analyze system's behavior whether it refers to software performance or processor's efficiency.

Software based monitoring schemes, use program instrumentation techniques [114, 115] to instrument the original application with additional code that is able to perform the monitoring. Unfortunately, the main issues with software monitoring have been its speed and inefficiency in dealing with multithreaded programs.

In order to provide better insight about application behaviors holistic approaches gain momentum relying on investigation and analysis of phenomena across system layers, rather than on observations of a single layer like operating system intricacies or cache organizations only. For example, Hauswirth et al., [23] describe methods under the term *vertical profiling*, that capture behavioral information about multiple layers of a system and correlate that information to find the causes of performance phenomena. By incorporating vertical profiling into a programming environment the programmer will be able to understand how the application interacts with the underlying abstraction levels, such as application server, VM, operating system, and hardware. Hauswirth focuses on observing application, virtual machine and operating system layers through hardware performance counters capability that exists in Jikes RVM [116] in order to gather aggregate performance data.

Software systems are comprised of multiple layers, each of which can be an effective source of instrumentation that serves as input for application optimization or problem diagnosis.

Application-aware instrumentation refers to modifications to the application code in order to extract the instrumentation data. Particular understanding of the semantics and functionality of the application is necessary in order to intercept the appropriate function calls, to record the right data and objects and inject monitoring code into the right point in the software system. In this respect this approach reduces its value as a generic technique. Nevertheless, application-aware instrumentation

does have its advantages; inevitably, application-specific data are often required to trace a problem back to root-causes as specific as line of source-code.

Application-agnostic instrumentation or black-box instrumentation is transparent to the application, does not require application-level modifications, and is free from the need to understand application's semantics or structure. In general application-agnostic instrumentation takes the form of performance data collected by the operating system, or by hardware performance counters, such as context-switch rate, CPU usage, network-traffic rate, or page-fault rate. Since black-box instrumentation is generic, it can be readily employed for any application in the same manner.

Hybrid instrumentation takes a middle ground. These gray-box approaches, usually at the library or middleware level, focus on instrumentation that can potentially be reused for an entire class of applications. For example, instrumentation that collects garbage-collection statistics at the Java Virtual Machine (JVM) level can be reused for any Java application. Other examples of gray-box instrumentation include path-tracing based on modified communications layers and management metrics provided by middleware.

Instrumentation for Program Debugging

Traditionally program steering involves on-line or post-mortem exploration of program trace or output data with the objective to debug programs or to profile its run-time behavior. Profiling techniques are widely investigated by researchers to acquire accurate metrics of code and provide a comprehensive analysis to the programmer.

In understanding application performance, the profiling tools help to identify hotspots in the application, or view relationships between functions, analyze application's I/O patterns, or analyze inter-task communication patterns. Accessing the hardware performance counters enables to perform low level analysis of an application, including analyzing cache utilization and floating point performance.

Instrumentation of Parallel Applications

Thread concurrency in the context of parallel programs is essential from parallel machines to

modern multicore SoCs. Deterministic execution of parallel applications is a difficult problem attracting the efforts of researchers towards efficient debugging methodologies and tools.

The on-line management of parallel or distributed applications has provided great improvements in many domains. Examples range from automatic configuration of task fragments for achieving real-time response in uni-processor systems to on-line adaptation of functional application components for managing reliability versus performance trade-offs in parallel and real-time applications [117], and to load-balancing or program configuration for enhanced reliability in distributed systems [118].

Monitoring parallel applications presents several difficulties, and a severe one is the requirement to preserve original behavior of the parallel application. Toward this end two issues arise: (i) monitoring can perturb application execution, and (ii) the monitoring techniques for event collection do not guarantee the preservation of actual time ordering of events being produced, stored and processed. In particular, since events are first captured and stored, the local monitoring threads are inherently operating in a parallel manner, and thus ideally must be perfectly synchronized to deliver events to the central monitor with in-order guarantees. In off-line monitoring events can be sorted. However, for on-line monitoring event re-ordering must be performed on-line and with suitable efficiency. Furthermore, reordering must be performed so that the causal order of events exhibited by the running application is preserved and enforced.

Restructuring application code for efficient memory access can result in a big performance payoff, even though using hardware counters derives measurements which can be very processor-specific. For example, cache tuning and thus code modifications that result in better performance on one processor may degrade performance on another processor.

Recently hardware acceleration is employed to facilitate practical methods that use instruction-grain lifeguards to monitor a parallel application by time-slicing the multiple application threads onto a single processor and analyzing the resulting interleaved instruction stream sequentially. These hardware schemes are lifeguard accelerators called inheritance tracking, idempotency filters, and metadata-TLBs [119]. Through parallelization it is shown that fast online parallel monitoring

of multithreaded parallel applications can be achieved [120].

Monitoring in Invasive Applications and Architectures

Given the availability of chips with vast on-chip processing resources, new ideas introduce the concept of invasive programs that allow themselves to explore and claim hardware resources in order to achieve optimum performance [121, 122]. Teich proposes a self-organizing computing paradigm, named *ivasive programming* in order to manage and control parallel execution in SoCs with hundreds of processors. Running programs can manage and coordinate link configurations and processing elements themselves. However, this paradigm imposes serious headaches to operating system and to typical compiler technology, as hardware abstraction layers or security levels are violated. In contrast to this idea, agent-based approaches that distributes tasks over processor resources is more appealing to system architects.

2.4.4 Hardware Monitoring

As current nanometer technologies suffer within-die parameters uncertainties, varying workload conditions, aging, and temperature effects that cause a serious reduction on yield and performance, system-on-chip designs increasingly integrate multi purpose monitors. In addition, dynamic power and thermal management (DPM and DTM) emerge as solutions to avoid spatial and time distributed hotspots while sustaining current performance improvement trends. Hence, architectures adaptable to variations of all kinds tend to establish a new ecosystem. These architectures rely heavily on information gathered from in situ monitoring circuits. Multi-use sensors and their hardware controllers that can monitor performance, degradation, temperature or power consumption as the circuit ages are becoming a fundamental sub-system of multi-core SoCs [80].

Hardware monitoring methodologies have enabled *recovery-driven* processors. These are processors optimized to deliberately produce timing errors at a rate that can be gainfully tolerated through an error recovery mechanism. Razor [123] and error-detection sequential (EDS) [124] are popular hardware methods for error detection and correction that utilizes circuit-level timing speculation. Circuit-level timing speculation-based techniques detect errors by sampling the same

computation twice; once using the regular clock and again using a delayed clock. An error triggers a correction when the outputs do not match. At this circuit-level the advantages of EDS designs include: elimination of datapath metastability, simple static-CMOS design, low clocking energy and a less-intrusive error-detection approach that does not affect critical-path timing.

In addition, there has been significant research on hardware support for run-time monitoring of tasks. These efforts have resulted in hardware-based monitoring tools [125, 126] which are fast but they require specialized hardware support in the form of wholesale changes to the processor pipeline, memory management and the caches.

2.4.5 Hybrid Monitoring

Hardware improvements in monitor processing usually work synergistically with software tools developed to assist in monitor information collection and filtering and to provide standard interfaces or precise sampling methods. Through these tools, users can extrapolate measurements obtained from samples collected either at predetermined points in the application or during sampling interrupts triggered by user specified conditions e.g., N cache misses. In general though these methodologies introduce error inversely proportional to the sampling frequency.

Hybrid monitoring determines the methodology that combines advantages of both hardware monitoring, like non-intrusiveness and minimum latency, and of software monitoring. In the later case it is relatively easy to relate event traces obtained from the measurements to the system under steering. However, this monitor processing constitutes an extra workload, often with undesirable impact to the behavior of the object system. To address these challenges proposals have been made for improving the performance counter infrastructure, such as through compression and optimized hardware-software communication [127], and for sampling and processing of profiling data [128]. These monitoring overheads which are experienced with monitor invocations may be controlled by use of different monitor types: sampling, tracing, or extended monitors, on the basis of the amount of detailed information that is collected trading off accuracy for performance or latency. Tracing monitors can generate timestamped event records in addition to simple samples, which may be used

immediately for program steering or stored for post analysis. Extended monitors can perform simple processing such as filtering and combining before producing output data. It is obvious that sampling inflicts less overhead on latency and storage requirements than the other types. However, combined use of all types may enable users to balance low monitoring latency against precision requirements.

Hybrid monitoring methodologies may as well present balance across more than two different axes: (i) design-time exploration techniques combined with run-time monitors (ii) hardware and software monitor methods including probes and target-specific agents, and, (iii) time and spectral techniques to balance monitor sensors, amount of extracted information and accuracy of measurements. Various works [129] describe frequency-domain signal representations to explore compressive thermal sensing techniques and traditional signal analysis techniques as means for thermal characterization. They consider the spatial temperature as a space-varying signal and they utilize the Nyquist-Shannon sampling theory to devise methods that can reconstruct the full thermal status from the measurements of the thermal sensors.

Hardware event sampling can be used to estimate a basic block profile and derive an estimated edge profile, thus reducing the overhead of profile collection since no instrumentation code is inserted [130]. Towards a different direction Choi et al. in [131] describe a hybrid methodology and infrastructure to profile and optimize the performance and energy consumption of multi-threaded applications for embedded multiprocessors. The collected performance data are analyzed by two analyzers, performance analyzer and energy analyzer. The performance analyzer classifies and arranges the performance profiling results while the energy analyzer applies the energy model to the performance profiling data to estimate the energy consumption.

2.4.6 Event Sampling

Time-based sampling term is used when system cycles, rather than events, are counted [132, 130]. For example, in code profiling blocks with instructions of higher latency and higher average cycles per instruction (CPI) are favored, as they have a higher relative probability of being sampled. Alternatively, when the number of certain events is summed and normalized by the number of total

events, or by the number of events in a local area or time window, then this approach to sampling has been called *frequency-based sampling*. This gives equal weight to all events. Because of this difference, time-based profiles and frequency-based profiles of the same activity are not identical.

Processors provide basic hardware support for collecting statistical frequency-based profiles in the form of an interrupt, driven by a countdown event counter. This can be done by sampling the program counter every period of N instructions, rather than sampling on a time interval. Alternatively, optimizations on the basis of utilizing performance monitoring units can deploy variable sampling rates to minimize sampling overheads. In order to amortize the cost of the interrupt over multiple samples systems have been designed for low overhead hardware-based continuous profiling, which buffer multiple samples in the hardware using hash tables before invoking an interrupt [133].

From a slightly different perspective, by bringing methods and terms from data networking to on-chip interconnects [134], sampling approaches for monitoring can be classified into two categories: (a) component-level passive monitoring, and (b) end-to-end active monitoring. Passive monitoring techniques involve deployment of monitors at each component to periodically collect system metrics like CPU utilization and memory usage. Active-monitor-based techniques send test transactions (e.g. ping, trace routes) through the network to infer network and components health. Passive monitoring-based techniques provide fine-grained metrics, but fail to provide end-to-end view of the system. Active-monitor-based techniques, on the other hand, can provide end-to-end metrics, but introduce additional traffic in the communication interconnect and fail to provide fine-grained analysis.

For each of these classes of sampling methods a number of different sampling mechanisms can be utilized with respect to sampling rate. The most important of those algorithms mainly in the context of Networks-on-Chip are the following:

- Systematic packet sampling, which involves the selection of packets according to a deterministic function. This function can either be count-based, in which every k-th packet is saved for monitoring purposes, or time-based, where a packet is selected every constant time interval.

- Random sampling, in which the selection of packets is triggered in accordance to a random process. Based on the simple such algorithm n samples are selected out of N packets, hence it is sometimes called n-out-of-N sampling.

- Adaptive sampling schemes, that employ either a special heuristic for performing the sample process or certain prediction mechanisms for predicting future traffic and adjusting the sampling rate. Those schemes have some inevitable disadvantages: there is always some latency in the adaptation process and in case of unanticipated NoC traffic bursts, the monitoring module will be possibly saturated. In order to avoid it the NoC monitoring designer would have to allow a certain safety margin by employing systematic under-sampling (at the cost of lower accuracy obviously).

A monitoring process can also be called *adaptive* on the basis of the amount of the collected information per sample. For example, for statistical purposes the monitoring process does not need the details of the events, while in the case of debugging, when deviating from "normal" behavior, more data may be required to increase the level of observability [135]. However, dynamically ranging the awareness level usually entails latency and complexity overheads.

2.4.7 Real-time Monitoring: Logging, Transmission and Processing

This section focuses on real-time collection of monitor data and post-analyzing them, or full real-time monitoring.

Most monitoring infrastructures try to capture the correct data to help identify potential problem sources. Profiling oriented techniques usually achieve low overheads through statistical sampling of system activity and by deferring processing of profile samples for off-line processing. It is more difficult at run time to find the correlation of different metrics to detect or uncover indications on the direction of causality. For example, if both instructions completed per clock cycle (IPC) and some other metric, such as memory accesses, change by about the same percentage and in the same direction, and memory access counts is a measure of work, this often indicates that the IPC is the cause for the change in memory access counting (less work can be done because less instructions

are completed per cycle). If the change in IPC is moderate, but the change in the other metric is extreme and in the opposite direction, this can indicate that the other metric is the cause for the change in the IPC (an extreme amount of extra work had to be done, and thus less instructions could be completed per cycle).

Additionally to capturing raw data, techniques are needed for the on-line processing of the captured data. In [44] a debugging co-processor is responsible to gather and organize collected information as monitored record in a histogram-table. When the co-processor receive a command packet invokes the indicated debugging operations, such as stop, step, continue the execution. However, the repercussion of on-line processing of monitor data on complexity is an important parameter that designers should carefully consider. Monitor information is usually reduced through filtering or compression, and hooks are developed to give the needed flexibility by software agents to handle the processing.

Monitors for fault detection and tuning circuits that are responsible to observe synchronization of signals or capture sub-clock cycle errors, such as *Transition Detector* in Razor-based processor [102] are examples of continuous management of the underlying system to ensure simultaneously higher performance at lower power consumption, while mitigating the impact of rising variations. Multi-purpose circuits are also deployed in SoCs that can monitor second order effects: performance, degradation, temperature or power consumption as the circuit ages, which are critical and their additive impact is essential for the system. The operation of these monitors though, can be relaxed and experience periodic activation to reduce overheads.

Based on the selection of the desired interconnection scheme for the transmission of the actual measurements in a NoC-based system with monitors, the NoC data can be categorized as:

- *In-band traffic*: in this case the NoC traffic is transmitted over the NoC links either by using Time Division Multiplexing techniques or by sharing a network interface.
- *Out of-band traffic*: when hard real-time diagnostic services are needed or when the NoC capacity is limited by communication-bounded applications then a separate inter- connection scheme is used and the NoC monitoring traffic is considered ”out of-band”.

Monitoring traffic can use either the existing NoC inter-communication infrastructure with typical support for mixed-priority traffic, or an organization which is implemented only for covering the requirements of the NoC monitoring system. The former has the advantage that no extra interconnection system is needed, but on the other hand it introduces additional traffic in the actual NoC; if this traffic fails to satisfy real-time constraints then, a dedicated NoC monitoring interconnection infrastructure is employed. Monitor data latency and application traffic need detailed architectural exploration, especially when irregularly distributed monitors or monitors of varying functionality are utilized, combined with dynamic workloads [136]. To provide on-chip monitoring of temperature effects Zhao [136] proposed an independent interconnect, MNoC, which supports routers with high, or low-bandwidth monitors. The MNoC can also be interfaced to a monitor executive processor to provide a software layer. Zhao also investigated alternatives through intermixing monitor data with application traffic over a shared NoC. The expected result is to increase the latency of applications' data, since the monitor information increase resource contention in the NoC.

On-chip monitor communication protocols can have more versatile behavior by adopting software-based monitor policies. Thus, in a more flexible approach a monitoring service can dynamically adapt its monitoring policy at run-time, based on environmental limits, such as temperature, voltage, and faults, or changes in priorities. For instance, similar to the strategy proposed in [137], the monitor can collect system notifications from distributed nodes and dynamically adjust the frequency of the notifications, based on system load. The higher the load (e.g. more tasks in the system), the lower the frequency of notifications. Another example of a software-oriented dynamic monitor is the adaptive system monitor described in [138]. This monitor attempts to reduce monitoring overhead by pre-selecting and focusing on key metrics. Only when an anomaly is detected in one of these key metrics, does the monitor adapt, by increasing the number of related metrics that are continuously monitored. Essentially, this monitor is able to "zoom in and out" of areas when problems are detected.

2.4.8 Monitoring Intrusiveness

Methodologies for monitors are inherently determined through the following directions:

- instrumentation of the running OS or application; impact to their behavior, and to the programming model
- impact, invasiveness to the architecture of the hardware components, to their performance and incurring cost in terms of silicon and energy
- level of impact to the system design flow

One of the categories of monitoring that can be performed on a hardware component or on an application is event tracing. Since the goal is to optimize this component through monitoring the designer or the programmer commonly try to use non-intrusive probes or instrumentation. In event tracing though, the high frequency of events generated coupled with the potential of monitoring a large number of components, can result in large amounts of event trace data. Transfer of these data to secondary memory can have multiple negative impact. If the system interconnect is used to transfer monitor traces along with the application-related communication, then it is not uncommon that the event tracing can significantly perturb the very application behavior being monitored. Moreover, when system memory, cache, or buffers are used for trace data, side-effects can appear as the storage normally available for the application or the OS is not in its original state. Finally, transferring monitor data to remote storage location, in a relaxed-consistent heterogeneous, multi-clock domain, dynamic system can cause issues to preserving the time order of events. Hence, the designer usually must address the trade-off to allow fine-grain monitoring of a component for a relative long time while minimizing perturbations, or coarse-grain sampling with decreased accuracy in determining the component behavior.

Fault tolerant monitoring as well as fault tolerant computing with the assistance of monitoring, which determine the ability of a system to identify and to respond seamlessly (with minimal disruption) to an unexpected hardware or software failure, can affect performance and put a risk to

system's energy constraints. As common practices include 'mirroring' of operations, communication or storage, or introducing new system safe states, hardware and software layers are customized in favor of producing a more robust system.

The design methodology for integrating monitoring and optimization circuits for dynamic parameter variations to enhance performance, energy efficiency and resiliency, typically includes additional steps beyond the standard design flow. To integrate designs for timing-error detection: embedded error-detection sequential (EDS) and tunable replica circuit (TRC), as described in [139], the additional steps are inserted into a standard register-transfer-level to layout synthesis flow. The flow consists of RTL synthesis, timing analysis and place and route with extraction and timing convergence. The updated RTL is run through loops of synthesis, floor-planning and timing analysis flow until max- and min- delay margins are satisfied.

The impact to the design flow at architecture level when inserting monitors in a SoC is strongly related to the level of sharing of system resources to the user tasks and to monitoring process as well. For example, in a NoC design flow described in [140], if the application NoC is extended with the monitoring resources, then, in addition to topology, mapping, and slot allocation that are computed for the user NoC the monitoring communication requirements and the required monitoring IPs and router links must be taken into account early in the design phase. Alternatively, the design flow is not affected when a monitoring solution is non-intrusive, as only the monitoring sub-network consisting of dedicated links is used for transporting the monitoring data. However, in general, by sharing any system resource, non-intrusiveness is potentially not guaranteed and must be enforced.

2.4.9 Hierarchical Monitoring

In a broad sense developers employ a hardware-software layering methodology to observe and manage complex SoCs. Conceptually a SoC design abstraction stack focuses on the system application design and on the platform design on one hand, and on the hardware IP-block and interconnect design and integration on the other hand, as shown in figure 2.9. At the top, service-level, protocols

and algorithms define the overall system resource monitoring layer and subsequent task management. In addition, this abstraction layer is necessary for usability and programming efficiency in modern multi-core systems where it is highly unlikely all blocks to perform the same work requested by the application simultaneously. For processors and SoC components, designers develop monitoring solutions that functionally include performance optimization, fault tolerance and power or thermal management. The monitoring operations can be assigned to hierarchically organized communication in the system.

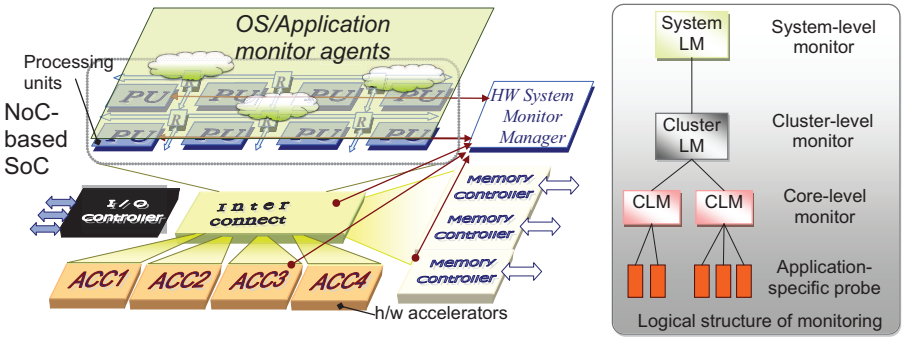

Figure 2.9: Hierarchical monitoring in a multicore SoC at software and platform level. The logical organization can be mapped to either level. Software monitoring provides greater flexibility and scalability but often worse performance, while the hardware approach yields better performance but requires more design effort.

In particular, hierarchy in monitoring infrastructure is essentially defined providing consideration for:

- *communication*, mainly two-level hierarchies are proposed to tackle management of monitoring probes and collecting traces in large systems-on-chip.

- *filters*, which implement various conditioning to the collected raw information, to extract the

information of interest and to reduce the amount of monitor data.

- *software agents*, which commonly consist of a low-level driver providing an intermediate API, and an upper monitoring service usable to the OS or to the application.

Distributed hierarchical monitoring architectures have significant advantages over centralized (monitoring is performed in one location) and decentralized (monitoring is performed at the critical end-points) architectures: it improves the performance significantly since the monitoring load can be distributed among monitor agents, and it scales well with the increase of components and generated events. Moreover, this monitoring architecture avoids single-point of failure, and it reduces the amount of event flow as events classification are localized in the area from which they are originated or generated.

As monitoring is used for processor and multicore SoC environments to manage a wide spectrum of metrics, from low-level hardware health to high-level service compliance, communication refers to intra-monitor communication protocols for scalable solutions, and additionally, to the infrastructure rules that govern gathering of monitor data and transmission of monitor control reaction commands. Various protocols have been proposed to transmit monitoring data information and notification of events. NoC-based systems promise efficient communication and QoS guarantees. Common schemes to achieve these goals include distributed or global management to monitor mapped tasks, congestion conditions, or deadlock symptoms. In [57] monitor packets are classified to forward *data* packets and back-propagated *alarm* packets. The protocol opens a monitoring session for each source-target path and the routers in turn register the congestion levels for each hop. If a specified threshold is reached then an *alarm* packet is send back to compute a new path. In contrast, Gratz et al. [141], although they also bring a global view of congestion, they use a separate network to propagate congestion information, different from the one used to transmit application data.

Exposing monitor information to upper system layers, middleware or operating system, presents varying characteristics, stemming from the different resource that is observed, or the different methodology that is applied to extract the useful information. Multiple shared event sources, unforeseen interactions and dependencies, between these sources can affect the end-to-end viewpoint

the software obtains. Services can decouple monitors' information of a single shared source of event, and each task can reason about its services independent of other tasks. Similar to resource virtualization in multicore SoCs, monitor services can be designed to offer diagnostic, performance, energy-aware services independent of the number, location, or interface of various monitor probes allocated in the system. Although services do not remove the interdependence of resources and the events they cause, such as buffers overflow as a result of multiple requests routed over a single path, there are several advantages when services are adopted. Implementation details are independent of the policies that can be developed and software tasks can rely on each service isolated from others, as they use their own resources.

2.4.10 Monitoring for Reactive and Pro-active Management

Performance monitors or workload characterization monitors have a special appeal to system and software developers who try to identify the true bottleneck in the system to be optimized. In this respect, in order to identify and adaptively build and consult an accurate model, monitor methodologies usually do not focus on fast active response mechanisms. When the monitor objectives include real-time on-chip bandwidth management, avoidance of power gradients in a SoC, or deadlock detection for example, then, low latency reactive schemes are usually preferred against time- or resource- consuming anticipation algorithms. Designers may also apply over-provisioning methods to maximize the amount of detected slack in the system.

The adoption of monitor-based resource and energy management in the design of multi-core embedded SoCs has opened the opportunity for adaptive real-time pro-active management [73, 142]. Runtime resource management techniques have emerged, as shown in figure 2.10 that employ predicting of the best hardware configuration for any phase of a program to maximize energy efficiency. The target is to devise pro-active techniques in order to achieve significant performance and energy improvements, potentially with finer control of workload processing, or of each chip's domain voltage and frequency, rather than managing the chip's single voltage and frequency, as in traditional chips with global DVFS. The main objectives of this management strategy include:

- Using real-time monitors to develop speed and accuracy of adaptive (real-time) workload estimation methods and circuits (traditional and novel)

- User-level and/or system-level cooperation for adaptive (real-time) pro-active energy and temperature management. An application can see a virtual environment which enables vertical interaction and dynamic service of requests for higher priority (i.e. processor time, scheduler time slots), for memory or network bandwidth

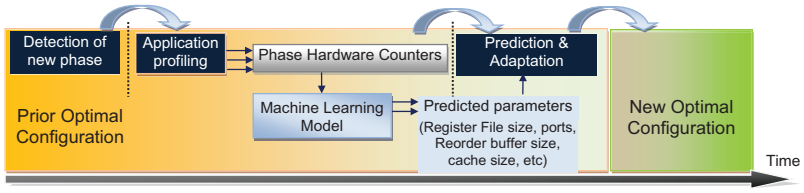

Figure 2.10: Overview of machine-learning model-based technique for microarchitectural adaptivity control that predicts the best configuration utilizing hardware counters collected at runtime (adapted from Dubach et al. 2010])

The goal of each prediction policy is to minimize the interaction between the prediction method and the multicore SoC and reduce the number and duration of reaction periods or time spent in crisis, while increasing the system performance. On the other hand, extended periods of throttling or policing access to system resources affects the system performance.

Alternatively to dynamic prediction mechanisms, one can extract monitor information and utilize system simulators to solve a set of differential equations using numerical methods. However, to determine the optimal point of operation through simulating different dynamic physical effects, like electromigration, NBTI, or even temperature is expensive. Moreover, when developing complex applications that are mapped on a NoC-based multicore system it is usually hard to simulate ahead various data distributions and loads at the design phase. Even by using good statistical functional

modeling and traffic generators dynamic behavior of particular applications (e.g., video processing) is difficult to capture. To simplify control one can use a fixed time period and attempt to adjust voltage/frequency at the end of the interval. However, this entails the risk to miss opportunities to respond to large activity swings inside the interval.

Monitoring mechanisms following reactive or prediction policies are invaluable in modern SoCs, since, for example, chips that simulations prove them to be fault-free in nominal conditions can develop temperatures up to 125^0C under high traffic that may result in 4% -10% fault probabilities [143].

2.4.11 Biologically-inspired Monitoring for Adaptive Management

The use of biologically-inspired computational techniques increasingly play an important role in developing complex adaptive systems covering a large variety of applications, each of them providing a specific adaptation: adaptive communications, adapting to changing environment and interferences, adapting to power limitations etc. Inspired by biology, *self-organization* of systems has been proposed and researched in the scope of distributed systems and artificial intelligence [144]. Emergent intelligent technologies such as *evolvable hardware* [145], or *immune-based systems* adopt a fundamental philosophy based on an agent-based framework where each agent has its own intelligence and autonomy in order to cope with the needed complexity.A design paradigm of an immunity-based system that is biased to robustness rather than to efficiency has the following properties: (i) a self-maintenance system with monitoring not only the non-self but also the self, (ii) a distributed and adaptive system with autonomous components capable of mutual evaluation [146]

Adopting biological concepts, hierarchical and distributed monitoring infrastructures have been proposed for online state collecting and subsequent evaluation of information [147, 148, 149, 150]. The example of the Digital On-Demand Computing Organism comprises several processing elements (cells) connected through a peer-to-peer network that integrate independent monitoring functions [147]. Aggregate monitoring can be realized using dedicated monitoring cells or monitoring “organs” created from arbitrary processing elements. The essential idea is to free the developer

from the burden of redefining evaluation criteria for the collected monitor information at design time; learning phases can be employed to dynamically redefine these criteria automatically.

Biologically-inspired massively parallel architectures appear recently as well [151][152], which set as ultimate goal to simulate the behavior of aggregates of billion neurons in real time. In the SpiNNaker system prototype chip [151] that integrates sixteen ARM processors, one of them is elected as the *monitor* processor after start-up to perform system management tasks. This is the only one processor that is responsible to run all the higher-level protocols, along with fault-tolerant protocols, while the rest operate as application processors.

Drawing inspiration from the behavior of ant colonies, swarm intelligence or other organisms, as well as research on pervasive computing, on wireless ad-hoc and sensor networks and on other fields is emerging, which however extends out of the scope.

2.4.12 Summary

This section presented a view of several methods towards employing monitoring in multicore chips. In particular, as monitors are a very promising foundation for investigating internal and environmental effects, building systems cost-effectively and in a predictable manner is a major engineering challenge. Optimizing each methodology and adapting it to emerging requirements that may be unknown at design time, especially from the perspective of multi-monitor and multi-agent approaches, has become an important consideration for system designers. Powered by sophisticated circuits, sensing techniques can be promisingly combined with new strategies, such as enabling novel knowledge acquisition with intelligent agents that collaborate through communicating learned rules [153].

2.5 Open Issues and Conclusions

When monitoring a SoC the objective of introducing and applying monitors does not clearly define the appropriate metrics to use in all cases. Monitor probes that are inserted for debugging or for fault detection are directly designed to these objectives. However, apart from these, researchers

propose various metrics arguing about the accuracy or strong correlation between a proposed measurable effect/quantity and the target to manage. Processor statistics such as fetch toggling, branch direction and address prediction accuracy, data and instruction cache hit rates, execution bandwidth, or IPC, correlate at a different degree to power dissipation of the processor. Similarly, occupancies of queues in a NoC, memory access patterns are used at system level. The reason is either because the infrastructure to collect these statistics is already in place, or because some architectural metrics are more robust. Sensor readings for example at peak temperatures are more susceptible to thermal noise and errors than some architectural statistics. In addition, accurate thermal modeling such as Hotspot-based methods are inefficient to embed at run-time when the focus is to manage systems that exhibit dynamic behavior. Overall, multiple methods, invoked in combination or on an application-specific basis are considered to more closely correlate to the objective that the user needs to monitor [80]. Discovering the proper balance between estimation accuracy and area or power overhead is of utmost importance to providing value through a system-level approach.

Table II summarizes a few proposals for efficient monitoring techniques giving an overview of the overheads incurred in the system. The cost of sensing and monitoring has been addressed extensively in networking domain (e.g., [154]) by exploring algorithms that combine polling and event reporting aiming at minimizing the overall communication cost, and in distributed systems (e.g., [155]). The multitude of works illustrate a strong need for system and hardware-software engineering process models, methodologies for building monitor-aware systems despite any overheads, usually minimal, that monitoring may induce.

In an increasing complexity multi-core chip ecosystem it becomes important to pair the most efficient monitoring technique with the constraints of each environment. For instance, in a NoC-based system there is extremely limited amount of message buffer space that monitoring messages must wisely utilize; additionally, transient traffic fluctuations make thing worse. Monitor configuration and transfer of collected data during system operation inevitably need valuable communication resources. Creating an optimal run-time time-slot allocation scheme requires a centralized, or global traffic view which is not scalable. On the other hand, defining a compile-time time-slot allocation

Table II. Overheads of various monitoring mechanisms for a multitude of objectives

Technique	HW(1) SW(2) HYB(3)	Objective	Overheads - Perturbations	Description
Roadnoc [Al Faruque 2007]	1	Observability, Buffer utilization of NoC routers	0.7% of the total link capacity, 41 slices (negligible)	Counter-based monitoring component attached to a router
ADAM [Al Faruque 2008]	2	Optimization: Task Mapping on a NoC	Increased communication volume: 13.3% in average compared to the exhaustive mapping algorithm	Run-time application mapping on a NoC in a distributed manner using an agent-based approach
MAMon [El Shobaki 2001]	1	Debugging	Integrated Probe Unit (IPU) slices: 181/3276(5.5%) flip-flops: 254/2580 (9.85%) Event FIFO for 16 events (12 bytes each)	Monitoring System for Hardware-Accelerated Real-Time Operating Systems
Vicis [Fick 2009]	1	Reliability for NoC	51% in area	Built-in self-test at each router diagnoses the number and locations of hard faults, includes ECC, a crossbar bypass bus, port swapping.
NUDA [Wen 2012]	1	Debugging (race detection)	Slowdown : 0.1-3.06%, Area: 0.37%, 64KB extra SRAM, 1KB extra CAM	Non-intrusive debugging frame-work for many-core systems, operates in parallel to the original data interconnection, enabling “non-intrusive" debugging methods
HARD [Zhou 2007]	1	Debugging (race detection)	Slowdown : 0.1 – 2.6%, Area: 0.98%, 64KB in L1, 1MB in L2 (cache coherency required)	The lockset algorithm in hardware to exploit the race detection capability of this algorithm using bloom filters
Isci and Martonosi [Isci 2003]	3	Performance, EDP	<0.1% in application performance using 100ms sampling	Counter-based power model with linear and piecewise linear combinations of event counts. Run-time phase analysis of threads
[Weissel and Bellosa 2004]	2	Power and thermal management	Performance loss <1% for reading event counters, 4.85μs for estimating temperature with standard error of 0.843	Counter-based energy accounting scheme as a feedback for OS directed power management using thread time extension, clock throttling.
[Alimonda 2009]	1	Energy	~1ms for PLL reprogramming, frequency adjustments cannot occur at rate > 0.2 per sec	Nonlinear control approaches to feedback-based DVFS
[Meng 2008]	3	Energy-delay product (EDP)	Stalls all cores, The time for system call is in the order of 0.01ms dominating the reconfiguration time overhead.	Run-time NoC channel width reconfiguration through performance counters and a prediction model
[Coskun 2009]	3	Temperature	Negligible performance scheduling	Proactive scheduling using an autoregressive moving average (ARMA) predictor for forecasting future temperature
[Shu and Li 2010]	3	Temperature	20% performance loss	Performance Monitor Unit (PMU) and on-chip temperature sensor to collect statistics and apply DVFS approach.
[Ciordas 2005]	3	Debug for NoCs	Area increase of a NoC from 17-24%, monitoring traffic two orders of magnitude less than total NoC bandwidth	NoC hardware monitoring services, event generator, sniffer components, monitor network interface
[Arora 2005]	1	Secure execution of programs	Maximum overhead is 9.07%, average overhead is 5.59% as a percentage of the CPU area, intrusive to the design flow	Hardware monitor that observes the processor’s dynamic execution trace, application-specific methodology using the configurable h/w monitor
[Tschanz 2009]	1	Resilience to timing errors (CPU performance,energy)	Error-detection sequentials (EDS): 2.2% area overhead, TRC-based error detection: <1%.	Circuit techniques for detecting timing errors : error-detection sequentials and tunable replica circuits
HPS [Mousa and Krintz 2005]	3	Accuracy in profiling	0- 2.5% with an average of 1% for the most aggressive sampling rate (1/100) and ~0.2% for high quality (90% accurate) profiling	HPS: Hybrid profiling support, a hardware approach for macro-expansion of dynamically executing instructions for profiling applications
Valgrind [Nethercote 2007]	2	Program analysis, Debugging	30 times slower than the normal execution on average (according to [NUDA])	Dynamic binary instrumentation (DBI) framework

for NoC application data and for monitoring data can potentially provide sub-optimal solutions.

Monitor communication protocols differ from typical NoC interconnect protocols, since monitors can be very application-specific, or require irregular topologies with sensor-aware communication schemes. Due to particular characteristics of monitoring operations and topologies, such as concurrency or filtering of captured data, statistical or critical data, isolated monitor network, or shared NoC, communication protocols may provide a different trade-off between performance and flexibility.

Consistency of monitoring communication protocol involves embedding consistency messages to handle exception situations: (i) asymmetric paths to the monitor manager may generate out of order synchronization issues, (ii) stalls, or context switching of the upper management processes must be considered to avoid memory penalty in case of upfront unknown amount of monitoring messages, (iii) reaction mechanisms in monitoring subsystem, such as interrupt triggering should take into consideration not outdated critical events. A consistency-aware protocol might protect system resources, like buffer memory overfilled, or interrupt time, through notifying monitors at the leaves of the hierarchy of the stall conditions.

Bringing hardware monitoring units in assistance to software agents is a cumbersome task. When benchmarking each mechanism a number of attributes affect the comparison and adoption of the appropriate scheme. Response time, interference, energy cost, residual dependencies, scalability are some important properties to determine the system monitoring policies. Moreover, these policies should be independent from implementation and safely identify the event-causing data. Monitoring process and communication semantics should be clear and well-defined to interpret the monitor data properly.

In summary, the development of on-chip monitoring subsystem and corresponding system-level services involves a number of trade-offs from architectural point of view, including communication protocols and software interfacing, interaction and interoperability as well. Through combining different methodologies monitoring the system behavior is a valuable vehicle to capture the changes in the behavior of the system and enable mechanisms to adapt to these changes. Moreover, new proposed methods for monitoring are likely to provide better advantages when treated as cross-layer

approaches. These should not be assessed in isolation and have value in combination only. Monitoring benefits are expected to increase as the demand for short time to market forces developers to design their SoCs with an early list of features, and rely on reconfiguration and programmability to complete the feature list during the product lifetime instead of before the product creation.

In the last decade we witnessed change in the direction of computer architecture. Power and thermal issues have become dominant, and the trend toward integrated multicore processors to improve performance via parallelism is expected to continue, while each core will be operating at a few GHz at the most. Researchers work towards the next step in increasing performance through moving to a technology with multiple active "tiers" in the vertical direction, developing three-dimensional integrated circuits. In addition, new enabling technologies accomplish to give designers considerable flexibility. The emerging availability of scalable Thermal Design Power (TDP), which measures the maximum amount of power that the design's cooling system must dissipate is enabling more flexible and powerful multicore processors to be used in mainstream and in embedded platforms. Offering the means, methodology, and tools to integrate adaptive computing is a new promising direction to essentially enable optimizations for performance, energy and reliability of systems-on-chips. Moreover, in VLSI design methodology, a shift from deterministic design to probabilistic and statistical design eases the impact of transistor variations on circuit performance [156]. The mindset of developers to handle only one or two objectives, namely performance and power is slowly changing, moving toward multivariable design optimizations that account for performance, dynamic and leakage power, reliability and process variation effects. A shift towards adaptive systems and chips by runtime monitoring is actually happening and will become more enhanced in the upcoming years.

Chapter 3

Hardware-assisted Monitoring Services for Networks-on-Chip

NETWORK monitoring is the process of extracting information regarding the operation of a network for purposes that range from management functions to debugging and diagnostics. Originally started in bus-based systems for the most basic and critical purpose of debugging, monitoring consisted of probes that could relay bus transactions to an external observer (being a human or a circuit). The observability is crucial for debugging so that the behavior of the system is recorded and can be analyzed, either on- or off-line. When the behavior is recorded into a trace, the run-time evolution of the system can be replayed, facilitating the debugging process.

Research has already produced valuable results in providing observability for bus-based systems, such as ARM's Coresight technology [157]. Also First Silicon's on chip instrumentation technology (OCI) provides on-chip logic analyzers for AMBA AHB, OCP, and Sonics Silicon-Backplane bus systems [158]. These solutions allow the user to capture bus activity at run-time, and can be combined in a multicore-embedded debug system with in-system analyzers for cores, e.g. for MIPS cores.

Since buses offer limited bandwidth, these simple bus-based systems at first evolved using hierarchies of multiple interconnected buses. This solution offered the required increase in bandwidth

but made the design more complex and ad-hoc and proved difficult to scale. As systems increase in number of interconnected components, communication complexity and bandwidth requirements, we have seen a shift towards the use of generic networks (networks-on-chip) that can meet the communication requirements of recent and future complex System on Chip (SoC). Figure 3.1 shows the use of a regular topology for the creation of a heterogeneous SoC. An example of how a heterogeneous application can be mapped on this SoC is also depicted. Of course the topology need not be regular as shown in figure 3.2.

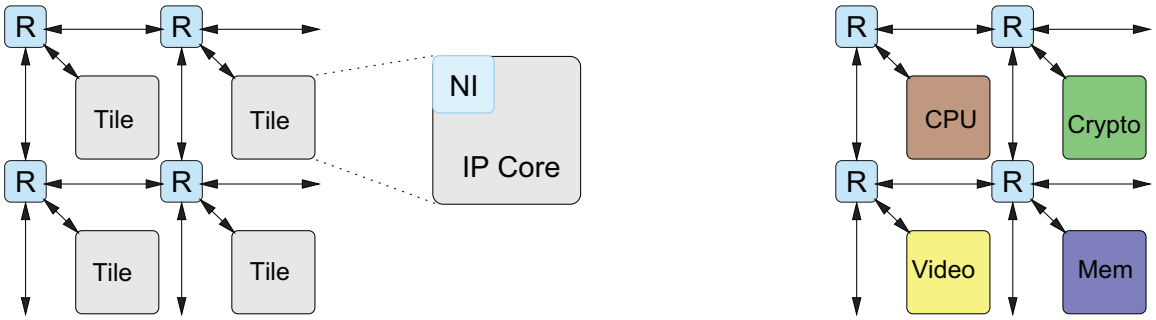

Figure 3.1: Network on Chip based on a regular topology, and an example with a heterogeneous application. Each node (or tile) is connected to a router, and the routers are interconnected to form the network. The nodes can be identical creating a homogeneous system (i.e CPUs), or can differ leading to a heterogeneous system.

However, this change increased dramatically the complexity of monitoring compared to the simpler systems for several reasons. First, the sheer increase in communication bandwidth of each component increases the amount of information that needs to be monitored or traced. Second, the structure of the system does not provide the single, convenient monitoring central location any more. As communication in most cases is conducted in a point-to-point and not broadcast fashion, monitoring recent and future systems is a distributed operation.

Despite all these difficulties that must be overcome for successful monitoring, the complexity of SoCs offer additional opportunities as well. To deal with short time to market, increasing fabrication

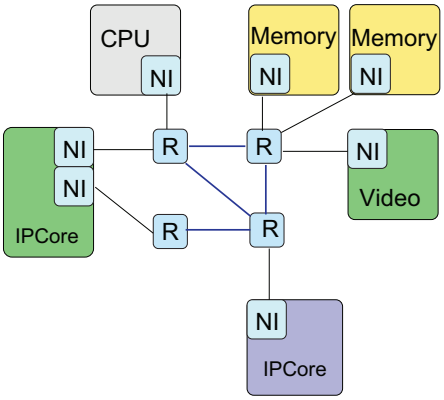

Figure 3.2: Network on Chip based on a irregular topology. The nodes are again connected to (possibly multiple) routers, but the routers are interconnected in an ad-hoc basis in order to customize the network to the application demands and achieve better cost-performance ratio.

(mask) cost, incomplete specifications at design time, and changing customer requirements, drives to increased complexity SoCs which are designed with increased versatility so that they can cover more application space and increase the product lifetime. These two factors, increased complexity and increased flexibility, lead to a dynamic system behavior that cannot be known in advanced at design time. This opens the possibility for dynamic system management where an applications behavior is monitored and adjustments to the system and its operation can be made either to improve the application function (e.g. provide better quality of service, etc) or to optimize the system's operation (e.g. consume less energy). Exploiting these opportunities depends on knowing the run-time characteristics of the system's operation and for the network this can be achieved using network monitoring.

Furthermore new opportunities open up as we move towards deep sub-micron implementation technologies. In these future technologies device reliability is an issue as devices are susceptible to a range of post manufacturing reliability failures. The consequence is that the designer has to deal not only with static faults, but also transient faults, wear-out of devices, etc. To address this

challenge future systems need to support redundant paths and resources and the ability to rearrange their operation to isolate and bypass failures are they occur. This operation requires the quick identification of the existence of a problem and its location so as to determine the correct repair action. Regarding the NoC resources, both these functions can be achieved by network monitoring.

In the next two sections of this chapter we first discuss in detail the objectives and the opportunities of network monitoring and their applications and then we cover the type of information that needs to be monitored and the required interfaces to extract information from the distributed monitor points. Then we describe the overall NoC monitoring architecture, and we discuss implementation issues of monitoring in NoCs, such as cost, the effects on the design process, etc. In the last two sections we present a case study where we present several approaches to provide complete NoC monitoring services, followed by our conclusions.

3.1 Monitoring Objectives and Opportunities

Monitoring can be used to provide information for many different applications that relate to the overall SoC management. The following section detail the main uses of network monitoring.

3.1.1 Verification and Debugging

This is the traditional use of monitoring, which is to add observability into the internal points of the complex system. The observability is crucial to enable the designer to track the system's operation and determine if and when something goes wrong. To this end, tracking the maximum possible amount of information is desired as it provides the best flexibility to the user, who is then responsible to focus on the exact needed bits and pieces.

For testing the nodes of the SoC, one approach is to provide a Test Access Mechanism (TAM) re-using the network resources [159] so as to minimize the cost and improve the speed of testing probes. The key observation is that due to its role, the NoC is a central piece of the SoC. TAM interfaces with test wrappers built around the cores to apply test vectors to the cores to be tested and also collect and deliver the possible responses. However, this type of operation is intrusive

and useful only for off-line testing. A more important usefulness of monitors is for debugging when the system is in operation and we want to extract information so we can track the system progress without affecting its operation (i.e. in a non-intrusive manner). To achieve this goal, the testing wrappers should provide the necessary information to the monitor infrastructure that can then deliver it to the tester without affecting the application's behavior.

3.1.2 Network Parameter Adaptation

Monitoring can be applied in a parameterized network to provide information for the update of the configuration or run-time parameters. For example, when the NoC uses adaptive routing, updates to the way path choices are made can better distribute the load, reduce the latency variation that is caused by congestion, and improve the overall system's performance. NoC monitors can provide the necessary input to a decision making algorithm that then updates the routing tables in the network to this end.

A similar application is to detect permanent network link malfunctions and errors (either permanent or even transient) and re-adjust the routing tables so as to avoid these defective links. This notion can be carried out even at the node level, where monitoring can detect defective processing nodes and provide feedback to the run-time system. Depending on the application, the run-time system should thus avoid the use of the defective node and migrate the processing to other, functional nodes at the possible cost of operating at reduced capacity. The mechanisms to support the isolation of defective links or nodes are basically the same as the ones used in adaptive routing (i.e. updates in the routing tables), and in a way we can think of defective links or nodes as permanently saturated areas that need to be avoided.

3.1.3 Application Profiling

For applications with dynamically changing behavior, monitoring their network patterns can offer insight on their overall operation. This can be useful for the purpose of application profiling, a process to collect information about it's run-time behavior. Profiling information can then be used

to better map the application on the existing resources.

A similar application is when the network supports Quality of Service (QoS) guarantees and can allocate specific portion of link bandwidth to certain communication pairs. Monitoring can detect when the QoS contract is violated, and this information can be used as feedback to either to simply detect the problem, or better yet to take actions (i.e. adjust the QoS parameters) to fix it if possible.

Obtaining information regarding an application's NoC usage can also be used to enable intelligent power management of the NoC resources. NoC monitoring can detect statistically important changes in the communication patterns, and can readjust the speed of uncongested portions of the NoC in order to save power. When links and routers do not support multiple voltage and corresponding speed levels, the identified routers can be shut-down, and their (presumably non-critical) traffic can be re-routed via other low utilization routers.

3.1.4 Run-Time Reconfigurability

Similar to readjusting the parameters is the use of run-time reconfigurable NoC systems. This approach has been explored as a promising way to overcome the potential performance bottlenecks because communication parameters cannot be estimated beforehand since communication patterns vary dynamically and arbitration performs poorly. As a result dynamic reconfiguration is used to change the key parameters of the NoC and eventually the communication characteristics can be tuned to better meet the current requirements at any given time. Such run-time reconfigurable NoC systems have been proposed in [160, 161, 162, 163, 164].

Moreover, as the silicon devices are getting more and more complex the testing of the NoC structure is becoming more difficult. In order to provide the means for NoC testing different Design-for-Testability (DfT) approaches have been proposed (as the one in [165] for example). However, a recent trend which is very promising is the use of certain run-time reconfigurable structures that can be used for both ordinary operations as well as for testing of the NoC structures such as the one in [166].

In order for all those run-time reconfigurable NoCs to adapt their structures on run-time an efficient on-line Monitoring system is required (such as the one in [167]). Those systems can mainly be based on reconfigurable Networking interfaces that in the ordinary case will route the traffic, coming from the IPs that are connected to them, and just keep statistics regarding the different characteristics of the traffic. The main difference between those monitoring systems and the others that are described in this chapter is that the former can take advantage of the run-time reconfiguration aspect in the following manner: Whenever scheduled, the interfaces will be altered in run-time and instead of sending the ordinary data, they will be connected to a separate networking infrastructure over which the monitoring data will be transferred to the main monitoring and reconfiguration module. Those run-time reconfigurable monitoring interfaces have the advantage that they utilize the same hardware resources with the standard NoC interfaces thus reducing the overall overhead of the monitoring schemes.

3.2 Monitoring Information in Network-on-Chip

3.2.1 A High Level model of NoC monitoring

As in every monitoring system, a NoC monitoring scheme should collect samples that may scale from simple bit-level events up to whole messages. The system designer or ultimately the real-time service may need to trace fine grain information such as interrupt notifications for instance, or even protocol messages and data. Testing a multiprocessor SoC obviously calls for a verification strategy which needs to consider the inherent parallelism, the on-chip network structure and the task attributes. Only a high level approach can tackle such issues. Abstraction via filtering of large amount of traced messages can be the key approach for a realistic monitoring service.

Events

In those high level schemes the data collected are modeled in the form of what they are called *events* [168]. Based on this approach all the events have specific pre-defined formats and they are

most frequently categorized since they may have different meanings. According to [169] "an event is a happening of interest, which occurs instantaneously at a certain time". Therefore information characterizing an event consists of: (a) a timestamp giving the exact time the event occurred, (b) a source id which defines what the source of the event is, (c) a special identifier specifying the category that the event belongs to, as well as (d) the information that this event carries. The information of the events are called "attributes" of the events and they consist of an attribute identifier and a value. The exact attributes as well as the number of them depend on the category the event belongs to.

Regarding the classification of the events, Ciordas *et. al.* have grouped them in five main classes: user configuration events, user data events, NoC configuration events, NoC alert events, and monitoring service internal events [168].

- The user configuration events are initiated by the IP modules that are connected to the NoC so as to configure the different NoC monitoring components accordingly. They are formatted in such a way that they present a system-level view of the requested information, hiding NoC implementation details. The information contained in such events can be large, when many details are needed for the configuration action or small when the subsystem to be configured does not need any specific information except probably of the timing of the communication and the communication modules.

- The user data events carry the monitored data from the NoC. Collecting the data can be through sniffing from either the various NoC interfaces or from the actual NoC links.

- The NoC configuration events are employed in the programming or configuration statically or dynamically, in a centralized or distributed way, of the underlying NoC. They are usually employed in NoC debugging and optimization since they carry all the requested information regarding the configuration of the NoC. So for example such events are produced whenever the actual routing protocol or routing state change.

- The NoC alert events are generated whenever emergency situations are triggered. Those situations include buffer overflows, internal or edge congestion, or even missing a hard deadline

in a real-time system. Through those events and based on the statistics of the utilization of various NoC resources, the NoC monitoring/administration system can be alerted that an abnormal situation is about to occur (or more importantly has already occurred).

- The Monitoring service internal events are issued by the monitoring service mechanism itself for various reasons such as synchronization, ordering, data losses etc.

Programming Model

Another critical constituent of the high-level description of a NoC monitoring system is the programming model of the system. Such a model describes in detail the procedure needed for setting up the various monitoring services. In general it consists of a sequence of basic tasks for configuring the NoC monitoring subsystems as well as a detailed reference description of the implementing of those tasks. For example, Radulescu *et. al.* have proposed a memory-mapped I/O programming model for configuring the different sub-modules of a NoC monitoring system [170].

The programming model should address the critical issue of NoC monitoring configuration time. In general, a NoC monitoring system can be configured at three possible points in time: (a) at NoC's initialization time, (b) at NoC's reconfiguration time, or (c) at run-time. Furthermore, the programming model should define the events that would be generated, the categories of the events that would be supported, the attributes of those events as well as a global timing/synchronization scheme and ways to start/freeze/stop the monitoring system.

The programming model also defines whether the NoC monitoring system will be centralized or distributed. In a centralized monitoring service, all the monitoring information is collected in a central point; this approach is simple yet efficient for small NoCs. However, in the state-of-the art SoCs with hundreds of different sub-modules, the collection of all the monitoring information to a central point may become the bottleneck of the NoC monitoring system. On the other hand, in a distributed monitoring service, the monitoring data are collected by concentrating components, and those few concentrators are interconnected together in order to be able to take a decision based on the global state of the NoC. Obviously, this approach, although more complicated than the centralized

one, removes the possible bottlenecks of the centralized approach while it is also significantly more scalable.

Traffic Management

The traffic management is another component of most of the abstract models of NoC monitoring systems. In general it is separated into the sub-system that manages the "configuration traffic" and that covering the "event traffic".

The Configuration Traffic includes all the messages/events required to setup and configure the monitoring scheme, such as the events to configure the NoC monitoring hardware sub-systems and the traffic for setting up connections for the transport of data from the actual NoC to the NoC monitoring processing system. The event traffic management system on the other hand deals with all the traffic generated after the NoC has been thoroughly configured.

NoC Monitoring Communication Infrastructure

It should be noted that the monitoring traffic can use either the existing NoC inter-communication infrastructure or an organization which is implemented only for covering the requirements of the NoC monitoring systems. The former has the advantage the no extra interconnection system is needed but on the other hand it introduces additional traffic in the actual NoC; if this traffic causes performance problems, a dedicated NoC monitoring interconnection infrastructure is employed. Based on the selection of the desired interconnection scheme for the transmission of the actual measurements in a NoC monitoring system, the NoC data can be categorized as either:

- In-band traffic. In this case the NoC traffic is transmitted over the NoC links either by using Time Division Multiplexing techniques or by sharing a network interface.
- Out of-band traffic. When hard real-time diagnostic services are needed or when the NoC capacity is limited by communication-bounded applications then a separate interconnection scheme is used and the NoC monitoring traffic is considered "out of-band".

Considering that the employed monitoring services are used for debugging or performance optimization purposes or for power management, it is clear that the choice of the appropriate interconnection structure is very critical since it may affect the overall NoC's efficiency towards the opposite direction to the desired objective.

A self-adapting monitor service could encompass programmable mechanisms so as to adjust the generated monitoring traffic in a dynamic manner. This means that using a hybrid methodology the distributed NoC monitoring subsystems or the central monitor controller can support an efficient traffic management scheme and regulate the traffic from the NoC to the central diagnostic manager. However, placing extra functionality increases the overhead of the monitoring probes, in terms of area, or energy consumption.

3.2.2 Measurement Methods

One of the main problems in NoC monitoring is that processing the entire contents of every packet imposes high demands on packet probes and their hardware resources. For this reason probes usually captures only the initial part of the packet which contains valuable information. Even so, and since the current NoCs work at extremely high speeds the amount of data collected is huge. One way for reducing the volume of the data is by utilizing certain techniques for filtering, aggregation and sampling just as it is done in the case of telecommunication network monitoring [171].

When sampling is employed and within a NoC monitoring infrastructure, a number of different sampling mechanisms can be utilized as described in [172]. The most important of those algorithms are the following:

- Systematic packet sampling which involves the selection of packets according to a deterministic function. This function can either be count-based in which every k-th packet is saved for monitoring purposes or time-based where a packet is selected every constant time interval. As described in [173], the count-based approach gives more accurate results in terms of the estimation of traffic parameters than the time-based one.

- Random Sampling in which the selection of packets is triggered in accordance to a random

process. Based on the simple such algorithm n samples are selected out of N packets, hence it is sometimes called n-out-of-N sampling. A certain algorithm of random sampling is what it is called probabilistic sampling. In this technique the samples are chosen in accordance to a pre-defined selection probability. When each packet is selected independently with a fixed probability p the sampling scheme is called uniform probabilistic sampling each packet is. Whereas when the probability p depends on the input (i.e. packet content) then this is non-uniform probabilistic sampling.

- The Adaptive sampling schemes employ either a special heuristic for performing the sample process or certain prediction mechanisms for predicting future traffic and adjusting the sampling rate. Those schemes have some inevitable disadvantages: there is always some latency in the adaptation process and in case of unanticipated NoC traffic bursts the saturation of the monitoring module will be possible. In order to avoid it the NoC monitoring designer would have to allow a certain safety margin by employing systematic under-sampling (at the cost of lower accuracy obviously).

Another way of reducing the traffic recorded by the network monitoring scheme is the use of filtering. In a formal definition filtering is the deterministic selection of packets based on their content. In practice, the packets are selected if their content matches a specified mask. In the general case, this selection decision is not biased by the packet position in the packet stream. This approach may also require a relatively complex packet content inspection, since depending on the NoC protocol employed, packets can have different formats and thus a fixed length mask cannot be applied.

Finally, there are the hybrid techniques which are based on combining a number of packet selection approaches. For instance, Scholler in [174] proposes to add packet sampling into the packet and create a scheme combining certain advantages of filtering and sampling. Another example is Stratified Random Sampling. In this approach packets are grouped into subsets according to a set of specific characteristics; then the number of samples is drawn randomly from each group.

Regarding the advantages and the disadvantages of each scheme, it is obvious that the systematic

sampling ones can very easily be implemented in hardware, whereas, as it has been demonstrated in [174] and [175], for the general networking environments it often performs better than random sampling. The main disadvantage is that it "is vulnerable to bias if the metric being measured itself exhibits a period which is rationally related to the sampling interval" [176]. This problem can be overcome when random sampling is utilized.

Depending on the specific SoC that a NoC is employed in, the hybrid sampling schemes can trigger the optimal point in the trade-off between the amount of data collected and the accuracy of the monitoring scheme. Such schemes can allow building accurate time series of different parameters and improving the accuracy of the NoC's traffic classification into flows, groups, etc.

On the other hand the filtering methods can be tuned so as to collect whatever NoC data are needed at each specific case, with the drawback of being relatively complex to be implemented mainly because of the extremely high speeds of today's NoCs,

Regarding the actual transmission of the sampled/filtered data to the Monitoring management processing system there exist mainly two approaches:

- *Raw Data Transfers*: if the probing components need to be simple enough and a centralized Monitoring management unit is employed or,
- *Filtered Data Transfers*: incorporating some manipulation on the sniffed data on-the-spot; if the system design can afford extra area then usually statistic functions can be used that measure traffic for instance: average, min/max, stddev.

3.2.3 NoC Metrics

There are several metrics reported which can be used so as to characterize the NoC monitoring systems in terms of a number of parameters. The most important of those metrics are the following:

- *BW - bandwidth requirements of each NoC Monitoring module*: since the traffic characteristics of the NoC's individual links vary so is the generated information by each NoC monitoring subsystem. Additionally even if two such systems collecting the monitoring data are

identical, they may require different network due to their different configurations; for instance a particular module can be configured to generate coarse grain statistics for diagnostic services, while another identical one which is used for debugging, may supply large amount of data.

- *NP - path coverage*: the number of paths in a NoC that can be monitored when the NoC monitoring subsystems are place in a specific location in the NoC organization.
- *RH - resource history*: This is, in other words, the depth in time that the monitoring info will be saved. Depending on the NoC monitoring scheme, in order to deduce a valid result at transaction-level or at higher abstraction level, the required storage at the different NoC monitoring subsystems may be significant.
- *RTR - real time response requirements*: The time between issuing a trigger event and getting back the requested information. For fault tolerant systems, or hard real time environments this is a very critical metric.
- *ND - NoC dependencies*: Those dependencies are determined in terms of interfacing and protocol compatibility of the monitoring traffic and the standard on-chip NoC traffic.
- *AD - Application dependencies*: Those dependencies cover the mapped application on the NoC and the associated NoC monitoring requirements in terms of resilience to faults, performance, power consumption etc.

3.3 NoC Monitoring Architecture

To aid the development of large complex SoCs, it is necessary to efficiently assist the real-time debugging of the complete design. This can be achieved by embedding hardware components that are usually application- or circuit-specific, as they are tailored to the verification needs of a particular circuit. One common approach is to attach monitoring components to a bus (as in [157]). Monitoring at this level consists of signal observation or event identification; events may be conditioned

to be activated only under pre-specified conditions. Several research and commercial solutions are built around this principle so as to achieve real-time debugging of designs; using in-circuit emulators is also a methodology that gains popularity.

System-on-Chip designs are growing larger and larger, and integrate a number of IP cores connected using a Network-on-Chip consisting of small routers and dedicated communication links. In this environment data are transferred in the form of messages or packets. Packet switching in turn is commonly based on the wormhole approach, where a packet is broken up into flow control units that are called flits, the smallest unit over which the flow control is performed, and the flits follow the header in a pipeline way.

The monitoring messages described in the last section should not block or even interfere with the typical user traffic, and should be follow the on-chip communication protocol. In terms of interfacing, the monitoring component to the NoC should also comply with the router and network interfaces. The aim of the monitoring services ranges in goals and functionality and correspondingly the type of data extracted to support meaningful conclusions varies. The type of data can be categorized as follows:

- measurements, for example statistics using counters that are usually event-triggered,
- content based measurements performed by sniffing in the content of the data units (flits, packets, messages) transferred on the NoC,
- filtered data extracted from the content of the data units,
- headers from the packets,
- payload data meeting particular conditions (filters),
- configuration data exchanged between routers.

The use of monitors to collect network interface statistics in order to assist the operating system controlling the NoC has been proposed in [177]. Such performance monitors can be used to optimize

communication resource usage so as to control the interaction between concurrent applications. Router-attached performance monitors on the other hand are used in [178] to keep track of the network utilization. This information is used by the network manager to adjust the Quality-of-Service levels of the running applications.

Associated with the type of required monitor information is the probing method, which can be categorized as *cycle-level* or *transaction-level* probing. A probing component with the capacity to filter the sniffed data and identify messages up to transaction level is called *transactor* throughout the following sections. Independent of the type of sniffed data, a cycle-level transactor operates on every clock cycle, and its bandwidth requirements depends on the number of signals to observe. At transaction-level however the transactor completes each observed datum at the end of each transaction. Thus, processing and possibly increased storage may be required on-the spot allowing each probe/transactor to communicate only the higher view of an interconnect link. A cycle-level probe needs to use either high bandwidth links, or a compression scheme to reduce the rate of monitor data. A monitoring component can potentially include both options as shown in 3.3 unless the area cost cannot be afforded.

At a higher level, all these strategies derive from the event model. The monitored information can be modeled in the form of events, with an event model specifies the event format, e.g. time-stamped events or not. Event taxonomy helps to distinguish different classes of events and to present their meaning.

A general-purpose monitor for on-chip interconnects is usually based on a layered structured approach, similar to the one depicted in 3.4. A first step to embed a monitor-sniffer with dummy manipulation of data includes only the most basic functionality, such as only copying the observed data on an on-chip link. However, observing all NoC transfers even for a single link may generate large amounts of data that require further analysis. Especially this is a significant issue since the collected data have to be processed at a different location, either on-chip or off-chip. The complexity is increasing dramatically according to the number of monitors and the amount of required captured data.

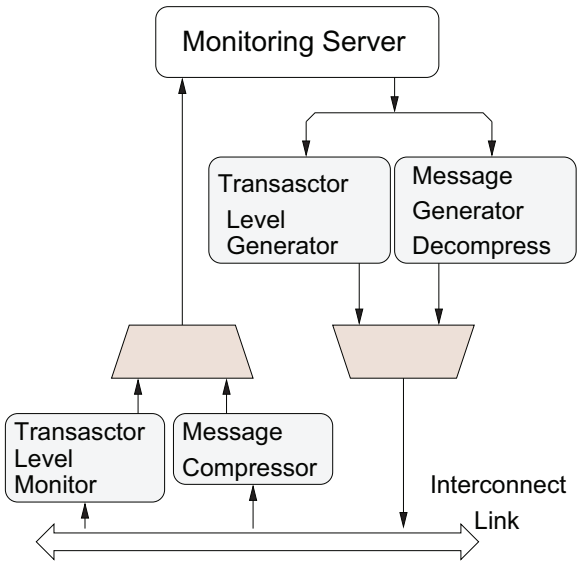

Figure 3.3: Monitoring component combining the two alternatives: sniffing data and filtering up to transaction level, and streaming the messages using compression to reduce transferring large amounts data.

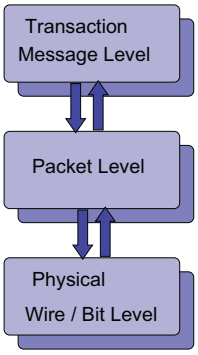

Figure 3.4: Layered organization of a monitoring component

Second, the post-processing of the sniffed data should be done by a more advanced *transaction* level analyzer in order to deduce useful conclusions. In between the two layers, it is necessary to

place another layer to filter and classify the data. In particular, since each message conveyed from end-to-end consists of header and payload, the monitoring component must perform de-capsulation in the same way as a Network Interface. The flits transferred over a communication link may be interleaved by the network interface or the routers. The monitors must therefore collect and construct messages from the right flits. In addition, connections of different bandwidth demands should be identified and treated accordingly. The connections of interest are usually a subset but in the worst case all may be monitored.

Hence, the monitoring service inside a NoC-based System must deal with the following main issues:

- Conditioning / Filtering of sniffed data
- Prioritizing critical vs. non-critical sniffed data (i.e. for statistic purposes)
- On-the spot analysis and possibly reaction based on the analyzed data.

Filtering is based on a priori knowledge of the type and the format of data and in some cases also of the timing of the data of interest. Dimensioning the filters though is a tradeoff between flexibility and increased area cost. It is feasible to use masks and even more intelligent event-based filters as long as the total overhead is affordable. The benefits of appropriate conditioning focus on reducing the traced data to only the critically needed pieces of information.

Monitor services can be characterized as Best Effort (BE) assuming either that the probing of a link is done periodically or that the messages sent and henceforth the reaction to them is not strictly real-time. This type of services is useful for observing liveness of a core or of a link. Moreover, since prioritization of on-chip connections is a usual mechanism to differentiate and ensure quality of service for on-chip user traffic, the same prioritization should be ensured also for the monitoring services.

Meanwhile monitoring services that need Guaranteed Accuracy (GA) may be required when exact piece of information is needed; for example to calculate throughput based on bytes send over a link or for debugging purposes. In addition, hard real-time performance necessitates the quality

of GA services in terms of low latency and complete view of the traffic or capacity to sustain monitoring traffic at full throughput. Guarantees are obtained by means of separate physical links or by means of TDMA slot reservations in NoC interfaces.

While this is a modular approach, it may suffer from transferring and possibly keeping in memory large amounts of data. If the memory refers to on-chip memory resources then the issue that can be raised is the amount of available memory, while if off-chip storage is used then the issues shift to bandwidth needs, pad limitations or augmented system complexity. Additionally, multiplexing with already used memory interfaces affects the available bandwidth and may raise redesign considerations.

If on-the-fly analysis is desired then such a monitoring must be application specific (i.e. hard monitor). Alternatively an embedded software solution can also perform such an analysis provided that software latencies can be tolerated. This is a viable option in low bandwidth configurations of a NoC or for a monitoring application that is not critical in a real-time environment.

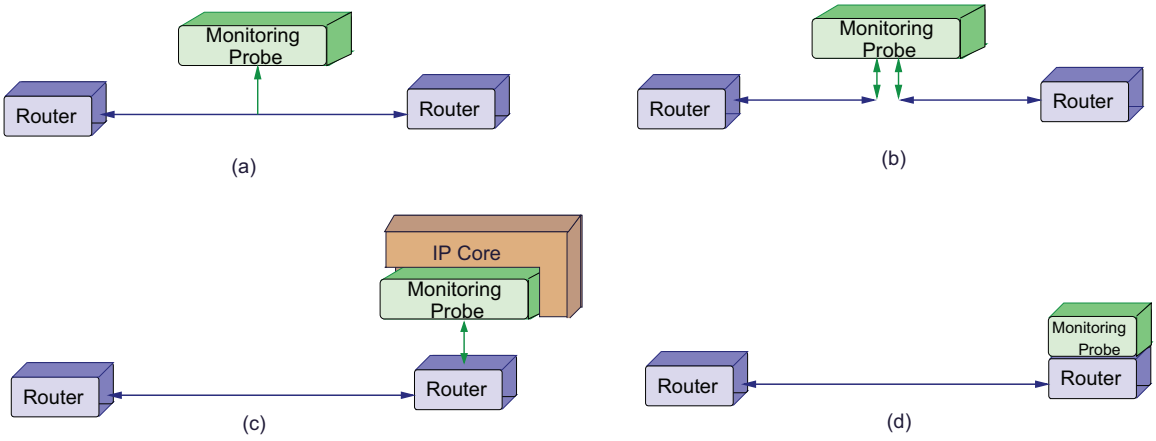

Figure 3.5: Attachment options of a Monitoring component: (a) sniffing packets from a link, (b) operating as a bridge observing and even injecting packets, (c) collecting data also from the core of an IP, (d) accessing also the internal status of a Router

While attaching a monitoring probe to a link serves for real-time observation as shown in figure 3.5 (a), the alternative organization shown in (b) can handle total failures of nodes. If the monitoring probe is also attached to other link then the system live-ness is ensured in case of a dead router by replacing the router functionality with the monitoring services, as long as it can be supported. In the case shown in (c) then the monitoring probe embedded at the "edge" of an IP Core can also collect information from an the link to the router and from the IP Core. Finally, in (d) the monitoring probe has also access to the core functional part of a router. This configuration has the potential to allow also the observation to the internal state of the router and its port interfaces.

3.4 Implementation Issues

This section describes various monitoring implementation alternatives and the impact on the total NoC-based SoC. The communication requirements of the monitoring probes some times are not known beforehand. Only after the NoC to be monitored has been designed, or at least after some steps in the NoC design flow have been performed is it convenient to make decisions about the location and functionality of the probes. The mapping of the application to the NoC nodes has to be finalized so as for the designer to have an overall view of the needs of the system for monitoring and evaluate the requirements and costs of each option. These are closely related problems. Once a NoC architecture has been near the completion of the development phase some issues have to be evaluated regarding the integration of the monitoring probes: the location and the number of the monitor probes, the communication requirements among them and their impact to the total area and energy consumption.

3.4.1 Separate Physical Communication Links

Using a non-shared interconnect scheme allows a non-intrusive independent possibility for transferring the monitoring data at high speed. Thus, the original NoC remains intact and the user data are not affected in any way, latency, or blocking side-effect. This solution comes though at the expense of increased area. Although any interconnect may be used, depending on the area cost margins of

the system, the bandwidth requirements and the scalability potential, a separate physical NoC may be adopted due to its scalable attributes. This option is depicted in Figure 3.6 (a). One probe (M) and a Monitor Router (Rm) is added per router of the NoC. In a simple scenario this monitoring probe is attached to the router, however it could be attached to the network interface (NI) of each IP Core, or to the communication link. This implies and the range of capabilitites of the probe:

- connecting a probe to the NI an IP Core can also generate whole messages and retrieve information from the IP Core itself while,
- attaching it to the router can only collect data in the form of flits.

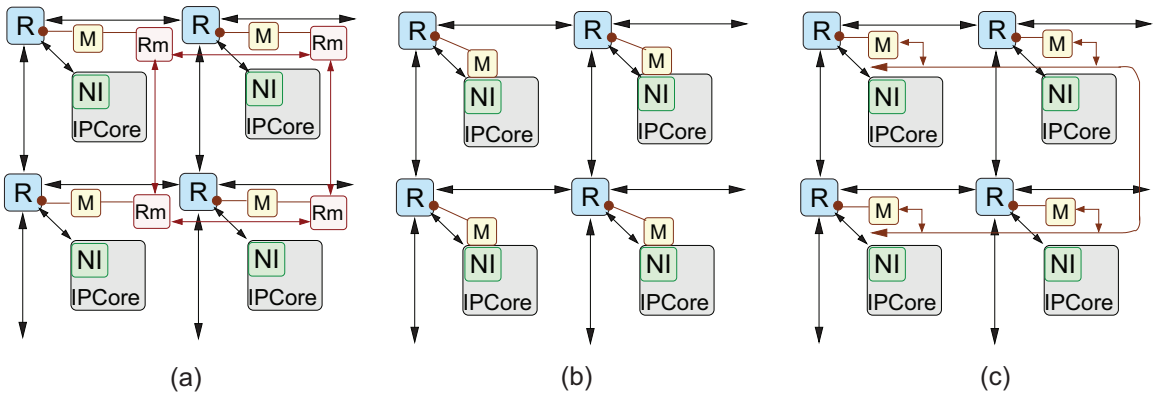

Figure 3.6: Monitoring architectural options: (a) use a separate monitoring NoC for transfering monitor traffic, (b) share the user NoC also for monitoring (c) use separate bus-based interconnect.

Another costly yet high-speed solution that also uses individual non-shared interconnect is to employ point-to-point links to the monitoring manager of the SoC. Thus, the multi-hop latencies are avoided and the system monitor manager can instantly send and retrieve data from the probes. While at the other end, a shared bus or set of busses can be used. Still the non-intrusive integration of the probes is achieved since no interference is possible via the physically disjoint interconnects, while a separate shared bus amortizes the cost of extra area.

3.4.2 Shared Physical Communication Links

An alternative is to use the existing NoC infrastructure both for the user traffic and for the monitoring traffic as well. The NoC resources can be completely shared by the integrated monitor, or depending on the NoC protocol implement a subnetwork or virtual flow for the monitoring traffic. Of course, in principle the user requirements for bandwidth must be respected and in fact they should be considered at the development and mapping design phase of the NoC.

Consequently, injecting the monitoring data also via the NoC interconnect may affect the number of ports of the routers, or the switching capacity of the routers and even the maximum bandwidth that can be sustained by each link. From a different viewpoint this design choice drives to considering NoC routers with monitoring probe capabilities. Nevertheless, if the probe needs to operate at transaction-level a separate probe block attached to the NI is a more structured approach. Although in this option of using the NoC resources in common, the interconnect cost is amortized in terms of area, the cost of the probes still remains the same as in the previous alternative.

3.4.3 Impact of Programmability to Implementation

A monitoring probe is usually a modular component, that consists of the NoC interface, the event generator and the interface to a centralized monitoring manager either on- or off-chip. By identifying packets, messages, local filtering and classification of messages per connection is possible; when raising the filtering to transaction level then detailed inspection of transactions can be achieved. Apparently, each level increases the complexity of the monitoring component-probe. Along with the needed logic is also the storage that is required; this is proportional to the message size, to the number of simultaneous connections and to the depth level of inspection. If the headers of each message is the only part of interest and examination of the payload is not required then the necessary storage can be minimized.

Figure 3.7 shows a modular approach of the transaction monitoring probe as described in [179]. The monitoring data path starts at the sniffer, that captures the raw data from the router links and

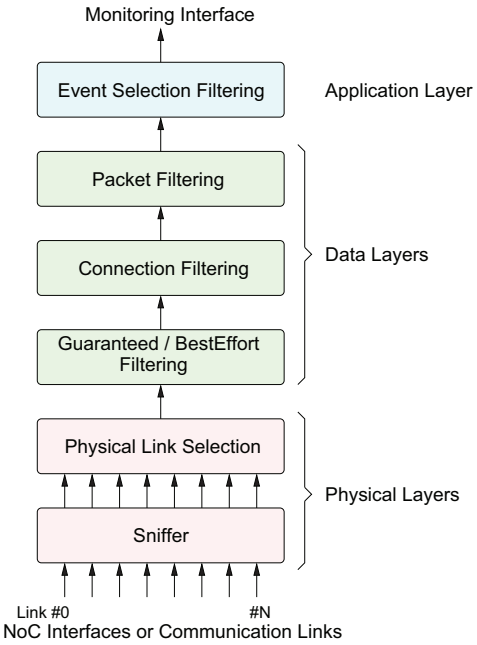

Figure 3.7: Layered organization of a Transaction Monitor: Each Filter layer can be configured at run-time via a command-based interface. The required functionality defines the number of layers of a Monitoring probe.

provides them to the transaction monitor. The link of interest can be selected at runtime by configuring the first filter. The flits can be further filtered as BE or GT in the second filtering block. Further filtering of flits is done by identifying a single connection from the set of connections sharing the same link, in the next filter. Transactions are composed of messages. Message identification allows to see, from within the NoC, when a write or a read message has been issued and from where or to which of the IPs or memories. Messages are payload packed in packets. Therefore, message identification requires depacketization, a procedure usually done at the NI. Finally, the Monitoring Server must be notified by the Transaction monitor according to the pre-configured format. Regarding the implementation impact of such a modular probe, it is reported that in a 0.13μ CMOS technology the implementation of a transaction monitor supporting the first four filtering stages costs $0.026mm^2$ in

silicon. Assuming that no filtering/abstraction is done locally at the monitor, the bandwidth requirements of the transaction monitors are comparable with the bandwidth of the monitored connection. This example demonstrates the potential to provide intelligent services to the system designer for monitoring a NoC at runtime.

3.4.4 Cost Optimizations

If the area budget of the SoC is limited then sharing a monitoring probe between two or even more routers of the user NoC may be an option. A transaction-level monitoring probe as described in [179] requires significant processing that is performed in layers so as to de-packetize a message for example. Hence, assuming the bandwidth requirements of the user and of the monitoring services are tolerable the same probe with this processing engine could be shared among many routers, thus saving useful silicon area. In the same direction a monitoring component may collect information from the IPcore as well. Thus, acquiring a view of the future transactions to be performed by the IPcore the monitoring probe can reserve space, or do a more intelligent conditioning of the events. For example, one function is tracking the messages passing via a NoC router depending on the status of the IPcore. However, the benefits and the total impact of every architectural option has to be evaluated in the environment of real applications mapped on the NoC.

3.4.5 Monitor-NoC co-design

Integrating Monitors efficiently into a NoC is not a straightforward task. Depending on the type and extend of monitoring, the monitors and their traffic will affect the performance of the NoC. Hence, the simple approach of designing the NoC and subsequently adding the necessary monitoring infrastructure often will lead to sub-optimal results.

There are several steps in the design process where monitoring support is interweaved and affect the traditional NoC design. The most obvious is the network dimensioning. If the monitors use the same network to transfer their information, the offered load at the links will be higher depending on the monitoring demands. This will in turn affect the decision for link speed, or the dimensions of

the network so that the total traffic can be supported and the desired maximum and average latencies are met.

Even if monitoring uses a different network for its uses, the two networks co-exist and share the same area resources. Hence, the physical distance between nodes is increased even when monitoring uses a dedicated network. This increase leads to longer point-to-point latencies, affecting the NoC clock cycle. The optimal NoC topology depends partly on the relative position of its nodes as well as on the availability of wires to connect the corresponding routers, so even when a separate dedicated monitoring network is used, the regular NoC design is affected.

When the monitors are fully deployed at every node (or more generally at a regular topology with respect to the rest of the nodes or network), placing the monitors largely depends on the physical location of the nodes, and the corresponding requirements are known in advance. When the monitors are placed irregularly close to nodes that are deemed important for a particular application (i.e. the monitors are customized for that application), then the effect of their placement cannot be determined in advance. Further complications may occur when the application itself can be mapped into SoC nodes in many different ways. In such a context, *Application Aware Monitor Mapping* [179, 180] is required to achieve good results.

A design flow for monitor placement should also attempt to optimize the required number of monitors, since according to the exact placement of the initially prescribed monitors, in certain cases the functionality of multiple monitors may be merged into a single one. The conditions for such a merge are many: the total aggregate monitoring traffic should not exceed the monitoring link (or channel when the network is shared) capacity, the merged monitor should have access to all the monitored information (being transactions, events, etc.) and the monitor programmable resources (if any) should be sufficient to satisfy the tasks of all the merged monitors. When merging is possible, the benefits are direct or indirect: smaller overall area is a direct consequence, but this also indirectly improve latencies and clock rates.

Figure 3.8 shows an integrated design flow similar to the one proposed in [180, 179] for shared

network use between the monitors and regular communications. The input to these flows is the description of the application network requirements (i.e. communication flows, bandwidth, possible latency requirements for QoS, etc), and, for the case with monitors, the monitoring requirements. While the main steps that determine the structure of the network remain the same, they are intermixed with a step that determine monitor insertion and mapping. The second step in the flow *Monitor Insertion*, places virtual monitors in the locations specified by the user. These virtual monitors are later materialized in the *Monitor Placement* step that can optimize both the number and location of the monitors, ensuring that the overall monitoring functionality is preserved. Finally, the dimensioning of the network is shown separated from the topology selection in the traditional NoCs since the overall network requirements are modified (increased) by the number and placement of the monitors. Finally, the process is iterated when the initial parameters (topology, etc) do not lead to a feasible system that meets all the requirements.

If the monitors use a separate network the NoC design is simpler and consists of solving the NoC for communications (Figure 3.8(a)) and solving in parallel a separate problem for the monitors using a simplified version of the integrated flow of Figure 3.8(b). In this case, convergence is much easier as the only interaction in the design of the regular NoC design and the monitoring and its network is through the increased area to accommodate the monitors and the monitoring network.

3.5 Case Study

In this section we describe existing approaches that target different goals using network monitoring to achieve them. This section is meant to motivate the breadth of applications that can use monitoring functionality.

3.5.1 Software assisted Monitoring Services

Kornaros *et. al.* propose a hybrid monitoring scheme for NoCs that features the flexibility of a software manager inside a customized embedded CPU, enhanced by small agent components in hardware to guarantee very high response time; these reside at the “edge” of the NoC [181].

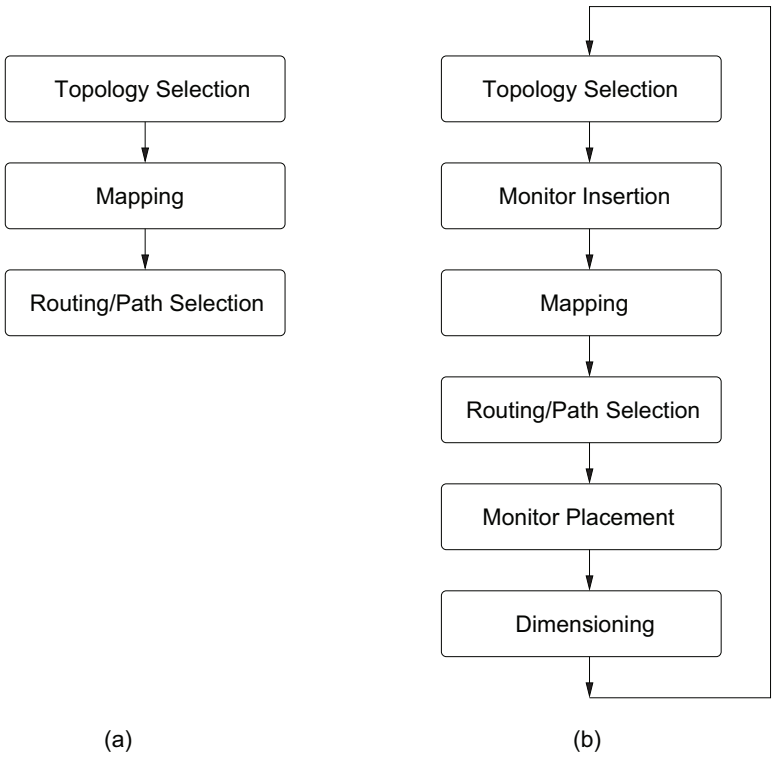

Figure 3.8: Integrated NoC-Monitor Design Flow. Part (a) shows a simplified flow for simple NoCs, while part (b) shows how it is changed to integrate monitor placement and optimization.

The proposed system features the following sub-systems:

- Hardware agents which are responsible to provide information to the embedded CPU and to perform the reconfiguration operations commanded by this CPU.
- An interrupt controller to communicate with the agents, arbitrate the accesses of the agents to the corresponding CPUs data cache and interrupt the CPU when necessary.
- A specialized RISC-CPU core optimized for the monitoring application, thus supporting high performance covering very small silicon area and having very low power consumption.

In this this work, which is shown in figure 3.9, a centralized scheme is adopted to manage the

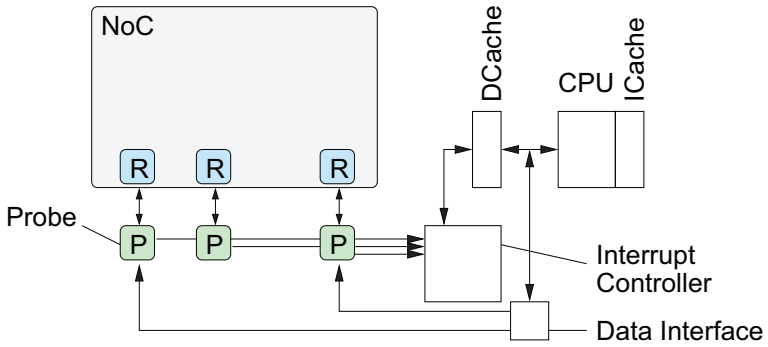

Figure 3.9: Architecture of the hybrid monitoring system of a software monitoring manager assisted by hardware interface accelerators

traffic of the on-chip interconnect, by controlling the limits of guaranteed throughput and best-effort priority classes. Nevertheless, special hardware mechanisms are employed to offload the centralized CPU from complex calculations. A hardware data structure is located at each network interface (NI) which logs the activity of the flits and the calculated statistic measurements. Programmable event generators are assisting to supporting of finer grain interrupts if this is desirable, or to masking out selected events. A master block implements additional timers with varying resolution. The desired objective for the management system is to react fast enough, when the fluctuations of traffic are identified. Even when the NoC is scaled to larger number of routers, the added complexity is shifted to the special hardware components that interface the CPU. The proposed monitoring system is implemented in a Xilinx Virtex4 device (FX100) occupying 12K slices and operating at 100 Mhz. A rather large crossbar is used as an inter-connect under monitoring; an 8-by-8 64-bit wide buffered crossbar was used as an interconnect, which is operating at 120Mhz on the same device without monitors. The simulations show that interrupt handling is achieved in less than 100 cycles for all the monitoring tasks.

Finally, this monitoring system can additionally discover and overcome defects on the NoC. Diagnostic and failure analysis may be periodically performed by the software to ensure operational integrity.

3.5.2 Monitoring Services Interacting with OS

The general problem of mapping a set of communicating tasks onto the heterogeneous resources of a platform on a chip, while dynamically managing the communication between the tiles is an extremely challenging task. Nollet *et. al.* describe a system where each node of a packet-switched NoC includes a traffic statistic monitor probe, a simple interface to the packet switched data NoC called data network interface component (dNIC), and a control network interface component (cNIC) that assists the Operating System to control the NoC [177]. The OS is able to control the inter-processor communication in the NoC environment matching the communication needs so as to provide the required quality of service. The OS can optimize communication resource allocation and thus minimize interaction between concurrent applications. The three main OS tools provided by the NoC are:

- the ability to collect data trace statistics,
- the ability to limit the time interval in which a processing element (PE) is allowed to send (called injection rate control) and finally,
- the ability to dynamically adapt the routing in the NoC.

Although the experimental setup presented in [177] includes only a few nodes, the layered organization of the cNIC provides a well-structured approach. The main role of the cNIC is to provide the OS with a unified view of the communication resources. For instance, the message statistics collected in the dNIC are processed and supplied to the core OS by the cNIC. Additionally, it allows the core OS to dynamically set the routing table in the data router or to manage the injection rate control mechanism of the dNIC. Another role of the cNIC is to provide the core OS with an abstract view of the distributed processing elements. Hence, it can be considered as a distributed part of the OS.

In summary, the three main OS tools provided by the NoC monitoring are: the ability to collect data traffic statistics, the ability to control the injection rate of a processing element (PE), that

is, the ability to limit the time interval in which (PE) may send messages and, last, the ability to dynamically adapt the routing in the NoC.

3.5.3 Monitoring Services at Transaction Level and Monitor-Aware Design Flow

Ciordas *et. al.* presents in detail in [179] how monitoring services can be taken into account at design time and integrate the monitoring functionality and placement in the NoC in the design flow of the system. The proposed direction is towards a unified approach by automating the insertion of the monitors whenever their communication requirements are known, thus leading to a monitoring aware NoC design flow. The proposed flow is exemplified with the concrete case of transaction monitoring, in the context of the AEthereal NoC and UMARS design flow. The objective in the methodology presented is the mapping of transaction monitors to routers such that a full coverage of user channels is achieved. Hence, they extended the coupling of mapping, path selection and time-slot allocation from the original flow to also include the monitoring probes.

In addition, in their research the cost of the complete monitoring solution is quantified; this cost includes the the monitors, the extra network interfaces (NIs), NI ports or enlarged topology needed to support monitoring in addition to the original communication infrastructure. Results show an area efficient solution for integrating monitoring in NoC designs. Monitors alone do not add much to the overall area numbers as the designs remain dominated by the area of NIs. In their work it is also considered the run-time reconfiguration of the monitoring system, showing acceptable reconfiguration times.

It is worthwhile that both approaches are explored, i.e using a separate NoC for the transportation of the monitoring messages and at the other end sharing the monitoring NoC with the application NoC. In the first option, assuming the bandwidth requirements are known, usually it is more expensive in area; however it features more degrees of freedom for the location and topology of the monitoring interconnect. In the shared case the combined communication requirements may not fit on the existing user NoC. In this case, it is clear that a new NoC must be generated, e.g. by increasing the topology and repeating the process. By increasing the topology, the number of

user NoC routers increases and in turn the number of required transaction monitors may increase as well (e.g. if probing all routers is required). This leads to the recomputing of the monitoring communication requirements and monitoring IPs. This means that the NoC monitoring flow must be revised. The reason for investigating this option is that the developed NoC with this approach is mainly the cheapest one.

Evaluating these options they report the following results using a 0.13μm CMOS technology. In the case of choosing a separate physical interconnect for monitoring the total NoC area cost of 3.82mm^2 (2.35mm^2 original + 1.47mm^2 extra) was determined based on the addition of 7 NIs for the 6 probes and one Monitoring access point (MSA), and of 6 routers. When the user NoC was shared with the monitoring components then the total NoC area cost was 2.75mm^2 (2.35mm^2 original + 0.4mm^2 extra). This was based on the addition of 7 network interface ports, 6 for connecting the probes and 1 for the MSA. The added monitoring traffic fitted completely in the original network.

The evaluation of the proposed monitoring methodology is done with benchmarks based on the AEthereal NoC (see [182]). The AEthereal NoC runs at 500 MHz and offers a raw link bandwidth of 2GB/s in a 0.13μm CMOS technology. AEthereal offers transport layer communication services to IPs, in the form of connections, comprising best effort (BE) and guaranteed throughput (GT) services. Guarantees are obtained by means of TDMA slot reservations in NIs. The target is to investigate how the monitors affect the mapping, routing and slot allocation in the design flow and the impact to the coverage of probed routers, to the complexity of the design flow and the resulting area implications.

They have used two real applications, *mpeg* and *audio*. Mpeg is an mpeg2 encoder/ decoder using the main profile (4:2:0 chroma sampling) at main level (720x480 resolution with 15Mb/s), supporting interlaced video up to 30 frames per second. This application consists of 15 processing cores and an external SDRAM, and has 42 channels (with an aggregated bandwidth of 3GB/s). *Audio* is an application that performs sample rate conversion, MP3 decoding, audio post-processing and radio. The application consists of 18 cores and has 66 channels all configured to use guaranteed

throughput. They have also combined these two applications into four cases to be used as examples: mpeg (Design1), mpeg + audio (Design2), 2 mpeg + audio (Design3), 4 mpeg + audio (Design4).

They have also generated synthetic application benchmarks for testing the proposed design flow. These benchmarks are structured to follow the application patterns of real SoCs. They created two benchmarks:

- *Spread communication benchmarks (Spread)*, where each core communicates to a few other cores. These benchmarks characterize designs such as the TV processor that has many small local memories with communication evenly spread in the design.

- *Bottleneck communication benchmarks (Bottleneck)*, where there are one or multiple bottleneck vertices to which the core communication takes place. These benchmarks resemble designs using shared memory/external devices such as the set-top boxes.

For the synthetic benchmarks the average area cost is almost 15%, while for the real examples the total area increase ranges from 2.2% up to 16.1%. The concluding result is that in all cases the area of the transaction monitors is insignificant relative to the total area of the designs, dominated by the area of the NIs. In the AEthereal NoC the amount of cores connected affects the number of NIs and the associated channels and not the routers. Thus, full coverage requires a large number of transaction monitors attached to the NIs. In other NoCs with cores attached to the same channel will require a lower number of transaction monitors. From the real examples the area-efficient solutions was when all routers were probed. Finally, in all designs the area of the monitors is several times lower than the area of the routers involved.

It must be noted that in the case of bottleneck designs the number of routers was inevitably increased. The situation might be even worse assuming that an irregular topology might be in use, or if TDMA was not employed. The benchmarks showed a dependence between the slot table size and the NoC topology; a mesh comprised of fewer routers required larger slot table size. Even most important, it is noticeable that the monitoring service itself is not considered in the design stage. It could dynamically affect and ultimately alter the application which is mapped on the NoC,

so as to discover and avoid bottleneck situations or hotspots at runtime. There is also very little research done regarding other synchronization, arbitration mechansisms in NoCs and the impact of monitoring traffic to it. Additionally, the transaction monitors in all these studies follow a centralized organization; The MSA for example configures the monitor function layers and collects the sniffed data. A distributed control monitoring scheme will obviously deviate from the previous conclusions and needs investigation.

3.5.4 Hardware Support for Testing NoC

Correa [183] and Cota [159] analyze methods to test a packet-switched network model named SOCIN (System-on-Chip Interconnection Network) by reusing of the NoC access channels to avoid the inclusion of extra hardware at system level. Originating form test strategies for on-chip multi-processor architecture where the processors are connected in a network-based model, the test of the routers exploits the similarity of those blocks by using broadcast messages throughout the network, showing that the test time can be minimized. Particularly, they focus in the test of NoC wrappers, and the strategy to shorten its design time based on the available network parallelism. In this case the wrappers are homogeneous, but the cores may be heterogeneous. NoC switching is based on the wormhole approach, where a packet is broken up into flits. With their methodology the externally generated vectors are transformed into messages to be sent through the network so that each wrapper is tested separately. The area overhead is minimal; however in this strategy the objective it to reuse the NoC for system testing while it is not in normal operation. It is interesting though the analysis on the hardware requirements of the test wrappers and on the methodology to achieve increased test coverage.

3.5.5 Monitoring for Cost-Effective NoC Design

Kim, Kim and Yoo propose the use of reconfigurable prototypes to achieve optimal NoC design [184]. Faced with the need to determine all the NoC architectural design parameters such as IP mapping, network topology, routing, etc, they find that many of these choices are affected by the

actual on-chip traffic patterns. Thus to achieve good results, the NoC design requires refinement steps based on knowledge of traffic patterns. To obtain these traffic patterns the designers can use analytical evaluation, simulation or actual execution.

An analytical approach is very quick but assumes that an accurate theoretical model exists for the application and its traffic pattern. In most case in NoC design this assumption is not valid. A Simulation-based approach provides accurate internal traffic observation at the cost of very long simulation time for detailed evaluation. This problem becomes worse as the SoC complexity increases in terms of interconnected nodes, but also since the processing power of each of these nodes is increasing. Another alternative is the use of H/W emulation, i.e. executing the actual application on hardware, but emulators do not provide the observability they required to measure the various NoC parameters.

The final option the authors consider is execution of the application on hardware coupled with the use of monitors to capture all the necessary information. This approach provides accurate NoC evaluation and enables the determination of design parameters based on a real traffic patterns. Since this approach is many orders of magnitude faster than simulation, iterative design refinement is feasible and can be used to achieve better results. They constructed a system that allows the measurement of the following traffic parameters: End-to-end latency, Backlog, Output conflict status, Total execution time, Bandwidth between IPs and Link/Switch/Buffer utilization. Using this system they investigated the best settings for buffer size assignment, network frequency selection, and run-time routing path modifications, while additional applications such as IP mapping and routing path selection, etc, are also possible.

They applied the NoC run-time traffic monitoring system and the collection of dynamic statistics on a portable multimedia system running a 3-D Graphics application, and found that through more accurate determination of the application needs they can reduce the NoC buffer size by 42%. They also found that using adaptive routing based on the run-time monitoring results can reduce the path latency by 28%. They also report using monitoring to choose the lowest frequency that meets the desired processing and communication bandwidth.

3.5.6 Monitoring for Time-Triggered-Architecture Diagnostics

El Salloum *et. al.* studied the integration of diagnostic mechanisms for embedded applications (e.g. automotive, avionics, consumer electronics) and SoCs built around the Time Triggered Architecture (TTA) [185]. The desired property of these systems is to achieve predictable execution for component-based design.

The goals of the diagnostic service is the identification of faulty IP blocks and the discrimination between transient and permanent faults. TTA uses a slotted approach to communication using global time base for the time-triggered Network-on-a-Chip, allowing a diagnostic unit to easily pinpoint the faulty components. The diagnostic unit collects messages with failure indications of other components at the application-level and at the architecture-level. Failure detection messages are sent on the same TT NoC. Each message is a tuple $<$ type, timestamp, location $>$, which provides information concerning the type of the occurred failure (e.g. crash failure of a micro component, illegal resource allocation requests), the time of detection w.r.t. to the global time base, and the location within the SoC.

To provide full coverage, failures within the diagnostic unit itself must be detected and all the failure notifications are analyzed by correlating failure indications along time and space, the diagnostic unit can distinguish permanent and transient failures and determine the severity of the action: whether to restart a component or to take the component off-line and call for maintenance action. The authors conclude that the determinism inherent in the TTA facilitates the detection of out-of-norm behavior and also find that their encapsulation mechanisms where successful in preventing error propagation.

3.6 The Future

Future research in the field is needed so as to offer more monitoring flexibility at a smaller cost. The trend is that processing logic is becoming increasingly cheaper than communication, so intelligent information pre-processing and compression can reduce even further the amount of data transferred.

Also the mechanisms used in profiling at the processing nodes and the monitoring of network resources offer a potential for convergence into a common mechanism. In particular, regarding the future work on the monitoring systems for NoCs there are numerous specific challenges that have not been addressed by the existing systems.

First of all the programmability aspect of the monitoring system has not been covered by the existing approaches. In order to efficiently and widely use the proposed monitoring systems, the operating and run time systems should be able to seamlessly support them; that requires the development of efficient and if possible standardized high-level interfaces and special modules supporting certain operating systems attributes. The complexity of this task is augmented due to the fact that numerous NoC monitoring systems are highly distributed.

NoC monitoring systems that will not only measure the performance of the interconnection infrastructure but also the power consumption and even some thermal issues may also proved to be highly useful. Such systems will utilize certain heuristics for evaluating the power consumption and the operating temperature and the thermal gradient of the monitored hardware structure.

Another interesting issue that has not been addressed yet is how the monitoring system can be utilized in conjunction with the partial real-time reconfigurable features of the state-of-the-art Field Programmable Gate Arrays (FPGAs). In such a future system the monitoring modules will decide when and how the NoC infrastructure will be reconfigured based on a number of different criteria such as the ones presented in the last paragraph. Since the real-time reconfiguration can take a significant amount of time the relevant issues that should be covered is how the traffic will be routed during the reconfiguration as well as how the different SoC interfaces connected to the NoC will be updated after the reconfiguration is completed. This feature will not only be employed in FPGAs; it can also be used in standard ASIC SoCs since there are available numerous field-programmable embedded cores that can be utilized within a SoC and offer the ability to be real-time reconfigured in a partial manner.

The monitoring systems can also be utilized, in the future, so as to change the encoding schemes

employed by the NoC. For example when a certain power consumption level is reached, the monitoring system may close down some of the NoC individual links and adapt the encoding scheme so as to reduce the power consumption at the cost of reduced performance. In order to have such an efficient system, the monitoring module should be able to communicate and alter all the NoC interfaces so as to be aware of the updated data encoding system.

It would also be beneficial if the future monitoring systems are very modular and are combined with a relevant efficient design flow so as to offer the flexibility to the designer to instantiate only the modules needed for her/his specific device. For example in a low-cost, low-power multi-processor system only the basic modules will be utilized that will allow the processors to have full access directly to the monitoring statistics that will have been collected in the most power-efficient manner. On the other hand in a heterogeneous system consisting of numerous high-end cores the monitoring system will include the majority of the provided modules as well as one or more processors that will collect numerous different detailed statistics that will be further analyzed and processed by the monitoring CPU(s).

3.7 Conclusions

Network monitoring is a very useful service that can be integrated in future NoCs. Its benefits are expected to increase as the demand for short time to market forces designers to create their SoCs with an incomplete list of features, and rely on programmability to complete the feature list during the product lifetime instead of before the product creation. SoC re-use for multiple applications, or even a simple application's extensions may lead to product behavior vastly different than the one originally imagined during the design phase.

Monitoring the system behavior is a vehicle to capture the changes in the behavior of the system and enable mechanisms to adapt to these changes. Network monitoring is a systematic and flexible approach and can be integrated into the NoC design flow and process. When the monitored information is abstracted to higher-level constructs such as complex events, and when the monitoring process shares resources with the regular SoC communication, the cost of supporting

monitoring can be incremental. However, given the potential benefits of monitoring during the SoC lifetime, supporting a more detailed (lower level) monitoring abstraction can be acceptable, when the monitoring resources are re-used for traditional testing and verification purposes.

Chapter 4

Design of Monitor Primitives

THE NUMBER of available hardware resources, the level of integration and the speed of components have dramatically increased over the years. Multicore processors common today are well-suited not only for massively parallel workloads running in servers but also for efficient processing of embedded workloads. First, they improve throughput per chip over single-core designs. Second, they amortize on-chip and board-level resources among multiple hardware threads, thereby lowering both cost and power consumption per unit of work (that is, per thread). Exploiting heterogeneity is becoming critical to achieving the energy efficiency levels necessary to enable embedded computing. In parallel with multicore chips, programmable domain-specific processors have evolved to exploit particular forms of parallelism common to certain classes of embedded applications. Examples include digital signal processors (DSPs), media processors, network processors, and field-programmable gate arrays (FPGAs). However, full systems often require a heterogeneous mix of these cores to be competitive on complex workloads with a variety of processing tasks. Embedded systems increasingly utilize heterogeneous compute resources such as embedded processing units (CPUs and GPUs) and multiple custom function cores for performance and efficiency. These systems exhibit a highly dynamic behavior due to the multitude of the system components and their interactions. This trend leads also to an increasing number of applications to be supported, which may have different features in terms of functions and traffic patterns.

In current heterogeneous systems, the heterogeneity is largely invisible to the operating system. Computing devices with different programming and execution models such as GPUs and custom co-processor accelerators are often accessed as I/O devices via library call interfaces. These resources must be manually managed by the application programmer, including not just execution of code but also in many cases tasks that are traditionally performed by an OS such as allocation, load balancing, and context switching. However, both the programmer and the OS lack of full hardware assistance and methods to investigate and identify sources of inefficiency at run-time. In particular, it is notable that increasing concerns are placed on ensuring protection of the code integrity of software running on commodity hardware [186] and mobile devices [88] utilizing custom hardware-based solutions.

Todays multicore chips develop significant diversity at the architecture and the programming level which strains existing programming models, especially when struggling to get the maximum performance within strict energy constraints. Thus, system monitoring and run-time control of the state and dynamics of the device is becoming increasingly important. The goal is to enhance system flexibility to adaptively match system resources on-line to dynamic workloads and second, achieving energy efficiency and desired performance under power and thermal constraints. These objectives come on top to the traditional use of software and hardware techniques, such as watchdog processors [1, 2] that exploit special-purpose hardware modules to monitor the control-flow of programs, as well as memory accesses for potential faults. The primary objectives include non-intrusive hardware support for SoC monitoring that imposes minimum concerns for probe-effects[1] for distributed/parallel real-time systems that are inherently sensitive to timing disturbances [187, 188]. In addition to dependability of on-chip systems that becomes an increasing problem as feature sizes continue to shrink, intelligent system monitoring is required to handle systems' adaptivity and enhance their efficiency. IBM has introduced the Autonomic Computing initiative in 2001, with the

[1] In concurrent programs the delay that is introduced by the insertion of additional instructions for instrumentation may alter the behavior of the program so that a functioning program may stop working when the inserted delays are removed, or a non-functioning concurrent program works with inserted delays.

aim of developing self-managing systems [189, 190] With the growth of the computer industry, notable examples being highly efficient networking hardware and powerful CPUs, autonomic computing constitutes an evolution to cope with the rapidly growing complexity of integrating, managing, and operating computing system.

In the scope of embedded systems, designing that will always meet deadlines can in general be achieved by over-engineering them. However, due to the requirements for adaptation, new algorithms have to be installed and used in existing embedded systems, and the static methods may not be able to guarantee performance after the system has been upgraded or modified. Additionally, over-engineering, while acceptable for mission-critical applications, is not desired in soft-real time systems where the cost of missing deadlines is unpleasant but not dramatic. In such systems, a graceful degradation approach to changing operating conditions is desired. In this work we consider dynamic multicore embedded systems that have performance, throughput, and power restrictions and possibly best-effort, soft deadlines, but not hard deadlines. Hardware-assisted monitoring is proposed for run-time optimization of these systems considering low overhead on-chip interconnect techniques and topologies for monitoring information, with the capacity to integrate a variety of different monitor types and processing components for the run-time monitoring data. This approach intends to overcome application-specific system and monitoring development as in traditional techniques that utilize general-purpose blocks of counters, or monitors deeply integrated in the processor pipeline.

The chapter presents a brief background on monitoring approaches for multi-core SoCs, while section 4.2 introduces the proposed designed monitoring support elements. In section 4.3 the monitoring infrastructure is analyzed at the Network-on-Chip level.

4.1 Background and Related Work

Embedded processors are highly application-specific with the requirements for a processor, in terms of its characteristics to differ for each application to be implemented. As the number of available

logic blocks on today's Field Programmable Gate Arrays (FPGAs) rapidly increase, complete multiprocessor systems can be realized on FPGA. Thus, a few proposals involve flexible multicore systems and even FPGA-based multiprocessors supporting reconfigurable accelerators. Various architectures have been presented in the literature with different reconfigurable granularity, such as Morpheus [191], XiRisc [192], or ADRES [193] processors and infrastructures supporting the design-time and run-time adaptation of the communication infrastructure, the number and types of processors and the accelerators, such as RAMPSoC [194]. It is recognized that the required embedded SoC characteristic can be different at run-time, since applications often need to react to the demands of the environment. The dynamic nature of many applications requires efficient management of the SoC resources, which sometimes is provided by the operating system to abstract the reconfigurable computing resources and provide an efficient layer [195].

Researchers have investigated monitoring infrastructures to provide efficient on-chip solutions in terms of consuming minimum silicon area and power resources. Guang et al., present monitoring services for NoC-based systems that functionally include performance optimization, fault tolerance, power and thermal management [196]. The monitoring operations are assigned to hierarchically organized communication in the system. Authors propose physically separate networks in order to provide flexible and energy-efficient transmission of monitoring information, with guaranteed latency independent of data traffic conditions. Ciordas et al. propose in [140] to take into account monitoring services at design time. Designers can integrate the monitoring functionality and placement in a network-on-chip through the system design flow. The cost of the complete monitoring solution is shown to be affordable for NoC designs. The overhead includes the monitors, the extra network interfaces (NIs), NI ports or enlarged topology needed to support monitoring in addition to the original communication infrastructure. Different configurations of dedicated monitoring infrastructure are presented in [197]. Zhao et al., describe a lightweight scheme to achieve monitor data transfers on NoC-based systems and show system-level benefits by using these data to scale processor frequency and operating voltage. The authors designed a low-overhead interconnect with interfaces to a variety of different monitor types and monitor data processing components including

thermal monitors and even a microcontroller. Al Faruque et al., also propose an event-based monitoring system, which offers adaptivity at both system and architecture levels [56]. Adaptation at system level is applied by mapping applications while adaptation at architecture level occurs by the execution of a routing algorithm and virtual buffer channels according to their demand.

State of the art proposals to deal with traffic congestion and Quality of Service (QoS) in NoCs also involve monitoring. Formally, QoS in system-on-chip communication can be broadly classified into two categories: (i) time-related performance guarantees (e.g. bandwidth, latency and jitter) and (ii) packet delivery-related reliability guarantees (e.g. in-order data transmission, loss-less data transmission). In order to control the injection rate and track any abnormalities in the network traffic, various approaches are based on specifically developed monitoring units [198, 199]. Tedesco et al., presents a distributed monitoring scheme to apply an adaptive routing algorithm at run-time [200]. NoC routers can be configured to change their routing decisions and re-route to non-congested regions. However, one of the main problems in NoC monitoring is that processing the entire contents of every packet imposes high demands on packet monitors and their hardware resources. One way for reducing the volume of the data is by utilizing certain techniques for filtering, aggregation, and sampling just as it is done in the case of telecommunication network monitoring [201].

Some approaches require extensive profiling to extract reliable workload statistics. However, it is questionable how effectively they can handle time-varying workloads. A feedback control theory-based approach was used to tune the system performance at a finer granularity [61]. Moreover, a DVS approach is proposed [202] to save energy consumption by exploiting the Kalman filter that can estimate the future processing time based upon the previous processing time observation and adaptively calibrate estimation error by feedback. Coskun et al. [73] use auto-regressive moving average modeling and lookup tables for forecasting temperature and workload dynamics, and proposes proactive thermal management techniques for multiprocessor system-on-chips.

History-based architectural approaches can be divided to one branch that performs adaptive responses reactively, based on most recent behavior and to a second branch which uses past behavior to extract predictable characteristics so as to predict future behavior with relative accuracy.

Researchers following the first direction use various metrics, such as application working set information ([203]), memory access patterns or to perform on-the-fly energy-efficient system management. On the other hand, run-time prediction includes performance counters for example to estimate certain metric behavior such as IPC and cache misses. Prediction can be guided by table based predictors or statistical approaches to predict variable application behavior. For instance, Isci et al. [65] identify recurrent execution and predict phases seamlessly during native application execution without prior instrumentation or profiling. Researchers have also proposed dynamically-constructed regression models, that allow the CPU to calculate the expected workload by estimating and exploiting the ratio of the total off-chip access time to the total on-chip computation time [63]. Then, the CPU adjust its voltage and frequency, in order to save energy, which shows greater benefits for memory-bound applications.

Donald et al. [204] analyzed power and performance characteristics of multicore architectures when running multiprogrammed workloads and parallel applications in the context of process variation. Additionally, Meng et al. [205] presented a strategy for thread assignment that reduces the overall power consumption for chip multiprocessors fabricated under extreme parameter variations. Isci and Martonosi [65] show that performance monitoring counters are suitable for determining functional unit utilization by which tasks are characterized. Though they propose an intrusive design approach it is concluded that global power management is required to be effective. The performance counters that exist in most processors can be used to monitor activity data (access count) of most on-chip functional units and therefore allow interpretation of these counter values also for localized temperature sensing across a microprocessor. Off-line regression analysis is presented by Chung [206], while on-line computation has been shown [75] as a way to infer temperature (even though to track a high resolution temperature one may argue it is expensive and the latency should be optimized).

The major contribution of this section compared to related works involves the combination of flexible components for monitoring of system processing together with intelligent monitoring interfaces for bringing an easily accessible abstraction layer (from the programmer's perspective)

for run-time adaptive SoCs. The monitoring primitives developed hereafter are not application-specific, instead are modular and flexible enough to be adaptable with minimum effort.

In general, application-specific monitoring includes techniques that are built to address a particular issue, such as cache accesses in order to optimize replacement algorithms and victim caches. IBM Power7 also utilizes 31 activity sensors per core to perform DVFS, voltage droop compensation and error rollback[18]. Alternatively, hardware monitoring units and sensors can be organized both in terms of their functionality and in terms of complexity and flexibility. The desirable traits of hardware monitoring, driving our objectives, can be summarized as follows.

- A set of well-defined functions that provide low-level facilities and services for easy development of a software management stack for monitor infrastructure
- Modularity of the hardware components that can be scalable to larger structures dealing with a large number of events or many tasks requiring independent event monitoring
- Programmability and configurability (even giving multiple personalities) to fit the run-time changing needs with low overheads
- Reaction capabilities embedding autonomous behavior in order to reduce management latencies or security risks

4.2 Hardware Hooks for System-on-Chip Monitoring

This section presents a monitoring infrastructure that provides efficient hardware primitives operating in a distributed fashion while facilitating measuring of metrics of interest and dynamic management of system resources. The monitor agents include mainly counter-based units that capture events with optional pre-configured filtering. Their contents may be set and read by software and used to analyze and optimize the performance and power consumption of the entire system. System-level metrics such as read or write data throughput and interconnect read latency can be computed by obtaining all metrics, after selecting the agents of interest in the system.

The design of the monitoring subsystem involves a number of trade-offs from architectural point of view, including communication protocols and software interfacing, as well interaction and interoperability; as such, the design must be performed in concert with the design of main multicore SoC resources. Collected real-time monitor information can involve substantial amounts of non-critical data (i.e., performance statistics, throughput, and jitter) that may require separate system resources to transfer and process them. On the other hand, critical monitor information, such as power and temperature or soft-error failures require instantaneous attention at system level. At the same time system dependability is increasingly important in the face of numerous environmental and process-related variability that can affect operation and performance of modern complex SoCs; for instance, these cover unexpected voltage drops in the power supply network, temperature fluctuations, process variations (gate length and doping concentration), and cross-coupling noise.

Next, novel hardware monitoring primitives are presented along with particular enhancements to support system-level run-time monitoring.

4.2.1 Monitoring Primitives in Hardware

Counter-based monitoring units comprise the primary components for collecting system events and maintaining statistics for performance optimization. By utilizing these primitives two main methods can be employed to retrieve monitor information: i) event-driven sampling that relies on interrupt notification when counter overflow happens, and ii) time-driven sampling where periodically the monitor manager collects monitor statistics. The following components are developed to facilitate accounting of various events.

Event counters: currently single 32 or 64-bit counters are designed to accumulate the captured events and are programmable in order to be controlled by the software running on the manager of the monitor components or on the processors itself. Each such counter is associated with a control register to enable programming of the behavior of the counter at run-time. The *mult-event mask* field of this register allows masking of a number of events occurring at the same time. This mask can be used to count events only if multiple occurrences happen in a clock cycle. The *interrupt* field

generates an interrupt either when the counter wraps around or when a threshold is reached. Finally, a *SU* flag, or supervisor field, provides a protection level.

Free-running timers: These timers are run-time configurable to increment at processor or sub-system clock rate. Timestamps can be enabled to mark the timing of captured events related to a free-running timer that operates as a wall-clock reference. Additionally, it is useful to obtain events from different clock domains associated with a local timestamp when insight is needed at a block level. However, currently we opt for a centralized counter/clock to avoid distributed time consensus issues. A tuple timestamp, event forms an event-time structure.

Event filter: A monitor filter commonly is build on the basis of a masking operation that applies on the captured sample in order to isolate the field of interest [140]. The developed multi-filter unit mainly consists of three masks which are user programmable. The following figure 4.1 shows the signal vectors/inputs Vdata and Vtrigger that comprise the data bus to monitor and the signal to trigger the sampling of this data bus. The masks can optionally be programmed by the user software and thus specify which part of the Vdata and Vtrigger is desired to be qualified. Furthermore, by programming the filter both greater-than-or-equal-to and less-than comparisons can be done at run-time.

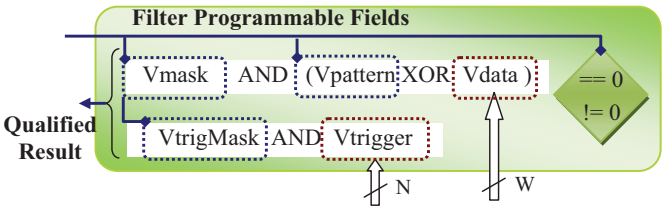

Figure 4.1: Structure of a single programmable monitor *event filter*; vector inputs Vdata and Vtrigger are specified by the designer at configuration time, while Vmask and Vpattern are programmable at run-time

In the current implementation the width of Vdata and Vtrigger vectors is design-time parameterizable, while the monitoring interface is set to 32-bit words for compatibility reasons.

Statistic counter (moving average): Consists of a counter with associated logic in order to manage an input stream of events (x) indexed by time (e.g. x_t is the value of x at time t), or counter values, while a new piece of data is received in a sliding window time interval measured in clock cycles. Hence, for a sliding window of eight entries the computations needed are one addition and one subtraction:

$Sum_{t+1} = Sum_t - x_{t-8} + x_t$

$Avg_t = sum_t/8$

To implement this hardware structure a circular buffer is configured with eight entries (the size is configurable at design-time).

Switching activity counter: Dynamic power consumption is directly proportional to the switching activity the fraction of input transitions in which a 0 to 1 transition occurs at some output node in a circuit. Besides the counter, this switching activity counter unit integrates a circuit to compute the number of bit transitions as a proxy in order to provide support for energy monitoring. Of course coarse-grain activity measurement can be based on accounting of traffic through counting number of packets. This particular switching activity circuit though offers very fine-grain metrics as each bit-flip is captured. The probe circuit monitors a fixed set of bits and captures a count of all of the bits in the set of bits that have changed state from time t to time t+1, and sums the count of all of the bits in the set of bits that have changed state.

In order to measure the transition activity the same cost model is utilized as in [207]. The *Hamming Distance* is used to estimate switching activities in the monitored bus. Given two binary strings, hamming distance is the number of bit difference between them. Let x_i and y_i denote two consecutive words, then h(x_i, y_i) is the hamming distance between them.

A system-level unit featuring multi-counter-based measurement components is architected in slices, providing a scalable solution in order to accommodate for capturing an extended number of different event types. Even though multiple sets of counters are packed in memory blocks, reducing

the utilized slices can amortize the cost in terms of area and energy consumption through partial activation of the slices in use. The developed infrastructure follows a middle ground approach by employing a shared control and interface glue logic that does not provide the ultimate performance but caters for the needs of medium multicore SoCs.

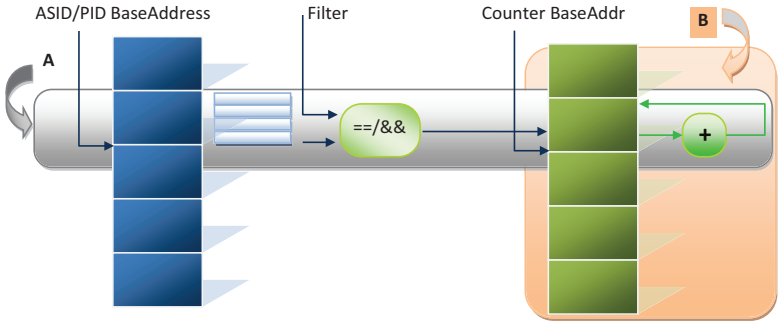

Figure 4.2: Organization of the multi-counter monitoring block allowing separation per task identifier in addition to filtering functions

The monitoring structure depicted in figure 4.2 is developed to operate in dual mode. The horizontal shadowed slice depicts the basic circuit unit, which includes a first level filtering, an equality, full or partial comparison stage and final recording in the counter unit; the process is activated by event generation as indicated by event trigger case A. Alternatively, the left part of the slice can be utilized as a filter stage to access the counters, while the counters can work independently and log preconfigured events; the vertical shadowed rectangle indicates the option to activate counters by event generation (case B). The partitioning and modularity of this scheme allows for protection against illegal accesses.

A high-speed system-level unit featuring multi-counters, extending the previous architecture and inspired by Salapura et al., is designed to support 256 counters with the capacity to operate and capture 256 events, one per clock cycle [49]. As shown in figure 4.3, each 8-bit counter operates

independently, triggered by an external event source; when an overflow occurs, the corresponding counter that is stored in memory is extracted and updated. At the same time, the pre-configured threshold is accessed to identify if a particular action needs to be activated.

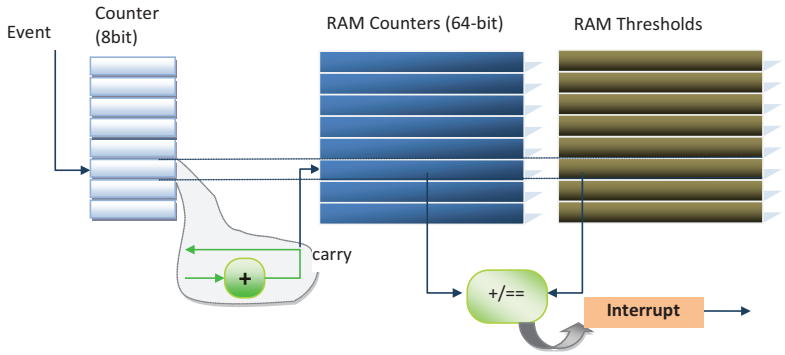

Figure 4.3: Organization of the multi-counter monitoring block using compact counters in block memory

The eight bit width of the counter circuitry allows for one overflow (wrap-around) every 256 clock cycles, which subsequently allows for accessing 256 times the memory block to update the corresponding value. Notice that the memory block is true dual-ported, thus permitting simultaneous read and write accesses. Enhancing the scheme proposed by Salapura et al., permission attributes are embedded together with the threshold values in the second memory block so as to further control the generation of interrupt signals.

Monitors with self-adaptive properties, as environmental constraints change in time along with the fact of workload variations, it is essential to adjust the parameters of the monitoring units in order to adjust their sensitivity and economize in terms of memory storage and power consumption. However, changes can be too short in time and as a result, observing and reacting in software would be ineffective. Figure 4.4 shows a new approach to build self-autonomous monitoring components by integrating a low complexity, yet very powerful module. In principle the objective is to perform

autonomous sensitivity adjustments fitting to the variations of the quantity that is sensed (e.g., power or temperature) in a time-based sampling approach, or adjustments of the time window that event capturing is enabled fitting to the current traffic profile.

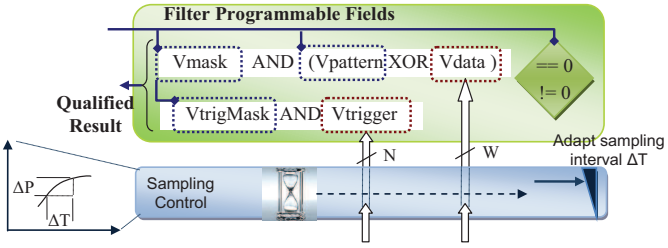

Figure 4.4: Embedding intelligence to the hardware monitoring unit with the filter through adding adaptive properties in order to adjust sampling rate in a dynamic fashion. The sampling controller tracks the qualified result values, samples S_i, (e.g., the delta power) to automatically adjust the sampling rate.

Prior to enabling this autonomous monitoring component, the sampling thresholds (lower and upper) must be configured. The qualified result values, samples S_i, are stored to create a short history (up to four values for instance), which is used to decide whether to change the sampling rate. If a significant change is detected, then the monitoring unit sensitivity is increased by reducing the time interval for sampling. At the end of each sampling interval the controller compares the current difference with the old ones and decides to scale up if the result is larger than the upper threshold and to scale down if the result is lower than the low threshold. Actually, only the last difference of the sampled values is sufficient, $\Delta S_{i-2} - \Delta S_{i-1}$; however, we maintain one additional difference value to avoid false positives. Controlling the establishment of lower and upper thresholds is still a responsibility of the monitoring manager. Run-time adaptation of the sampling rate though is off-loaded to this intelligent monitoring control module.

4.3 Monitoring Networks-on-Chip

Processor operations and transactions over on-chip communication interconnects are the dominant types of elements that developers most often desire to monitor. Since the observation of internal processor events is usually tightly related to the processor architecture itself, this section examines mechanism to incorporate monitoring over networks on chip, a crucial component of emerging multicore SoCs.

One of the primary objectives in embracing monitoring services is to develop advanced monitoring components in close synergy with network interfaces bringing a dynamic nature to these. Run-time collected statistics can assist, first, in optimizing provided QoS levels in terms of latency, throughput and jitter. Operating system and host's applications can provide improved resource management with the aid of run-time metrics such as throughput or packets slack. A packet's slack is the number of cycles the packet can be delayed in the network without affecting the application's execution time [200]. Second, based on run-time estimated metrics the system can adapt and improve the utilization of network resources and corresponding energy consumption through employing various reaction policies. By means of discovering potential congestion, adaptive routing mechanisms, such as those demonstrated by Moraes et al. [208], can be applied to support QoS traffic. Dynamic allocation of virtual channels and or queuing buffers, DVFS mechanisms, throttling and policing of packet injection rates are yet different alternatives to control NoC resources. The developed monitoring probes provide an easy, fast and efficient infrastructure to jointly monitor NoC-based system resources and software applications running on top, and a seamless integration of the hardware monitor agents with the underlying NoC infrastructure in a non-invasive way.

Monitoring a NoC infrastructure involves mainly two important aspects. In the viewpoint of supporting high-performance communication services, and particularly of supporting guaranteed quality of service, the monitoring policies should be tailored to impose negligible interference to the system and its performance/latency characteristics. The second aspect of monitoring involves the location and amount of monitor information that needs to be maintained.

Monitoring mechanisms usually are required to provide throughput and latency statistics. In

order to identify end-to-end events, such as request-replies in a NoC and corresponding latency or turnaround time mainly two techniques can be utilized. Packets are either tagged with timestamps and potentially with additional information, such as a network interface (NI) identifier and a sequence number, or new separate monitor packets are generated to provide similar information to the monitor at the destination NI. Both techniques are intrusive. It is essential that the monitor resources should be kept at a minimum to achieve low overhead.

The developed monitoring solutions are designed to differentiate various parameters of their own design in order to be adapted on the basis of systems' constraints and requirements. Thus monitoring can be developed as follows:

- maintain monitor information locally (applied for statistic counters, and only if the sharing degree is low, i.e. if many processor threads desire to access these counters could cause potential traffic overloads)
- maintain monitor information in shared memory (appropriate for large amounts of traces through the monitor probes and if the sharing degree is high)

Actually, the core monitoring components are not aware of where the collected information is stored. Their interface together with the monitors' manager is responsible to implement this task. In particular, the interface of the monitoring primitives is configurable to maintain a cyclic buffer of N entries (in the implementation presented in section **??**, N is set to 64), or interface directly the system memory. The physical address range of the buffer in memory is programmable at the interface in order to avoid spilling and pollution effects of the system memory.

4.3.1 Monitoring for throughput accounting

One important metric that evaluates the quality of a network-on-chip is throughput. Bandwidth indicates the amount of data that can be put on the network in a given amount of time. The monitors that we designed can be spread over a NoC infrastructure to measure various variables that can be exploited to characterize the traffic over particular physical or virtual partitions of the NoC. In

particular, the developed counters are designed as performance counters to capture the number of packets departing a router in a predefined and programmable time interval and additionally can be employed to measure the buffer occupancy inside each router, so as to determine its level of congestion. Furthermore, multiple performance counters can be integrated for a single router, or a network interface to address differentiation of the various traffic flows. Trading area overheads for supported counters per router challenges the use of block counters (with a lower granularity or sampling rate thereof).

4.3.2 Monitoring for latency accounting

Latency is a difficult comparison criterion, because it depends on many application-specific factors. Depending on the application or the criticallity of a guest, minimum latency on a few critical paths can be more important to measure and ensure via particular policies than statistical latency over the entire traffic flows. The overall system-level SoC performance usually depends only on a few latency-sensitive data flows such as processor cache refills, while for most other flows only achievable bandwidth will matter. But even for the latter data-flows, latency does matter in the sense that high average latencies require intermediate storage buffers to maintain throughput, potentially leading to area overhead.

Interesting latency measures such as the round-trip delay of a read or write request performed by a master core can be very valuable to optimize memory access patterns. This can be done in software by the core itself, or by a monitor component in hardware. However, software monitoring entails the maintenance of a possibly large number of entries in a queue since a master can initiate multiple requests and obvious software inaccuracies. We opted for distributed monitors that capture the latency of a packet as determined by the clock cycles measured from the time that the packet enters a router until the time that its first word departs from the same router. This difference is marked in the packet itself and is updated at each hop inside the network-on-chip. Thus, when it finally exits the NoC a local monitor extracts the accumulated latency. Practically, there is no overhead in terms of consumed bandwidth in our NoC infrastructure which is described in the following section, since

we utilized a 16-bit field of the header flit that was unused.

4.4 Monitor Infrastructure for NoC-based Systems-on-Chip

Monitoring and management of MPSoC resources is becoming an increasingly important challenge. Unlike traditional, small scale SoCs, lack of integrated, system-wide, parallel-hierarchical models presents a significant design challenge for creating scalable system-level monitoring solutions. This section explores the architectural impact of different organizations of monitors, communication protocols and implementation issues in platforms with dynamic characteristics.

A two-level virtual hierarchy for monitoring management

A software monitor agent is ideally an application-independent system thread which is responsible to collect and potentially pre-process useful information extracted from a core while various tasks are running. In a multicore environment interfacing such software agents or even migrating them is a viable (and desirable) option when building system-level policies for resource management. Underpined by collected information from monitors the system should facilitate dynamic reassignment or partitioning of processor and memory resources to multi-task applications (parallel or streaming) while respecting system constraints. However, hardware monitoring components which are often physically attached to critical points or paths in a complex multicore system should provide the option to support remote management services or differentiation for multiple simultaneous roles, or sharing of the exposed monitoring services. Such needs can be raised when a hardware monitor is utilized by various processors, possibly not physically attached to this monitor. To reduce system cost, monitor components should add minimal overhead either by moving functions to software or by instantiating one monitor per cluster of nodes. The latter option perfectly fits in island-based NoC organizations or GALS architectures. In this case configuration and overall management of the monitor block inside the cluster can be assigned to any cluster node which is named hereafter “parent” node.

In a multicore SoC or tiled CMP architecture the OS can dynamically assign resources to tasks.

Moreover, multiple applications can be deployed onto *Virtual Machines (VMs)*, with variable number of cores at different physical locations to be assigned to different VMs. To address such dynamic needs hardware monitors which are physically attached to particular nodes or tiles should facilitate dynamic re-assignment and sharing by multiple tasks or virtualization services for multiple VMs.

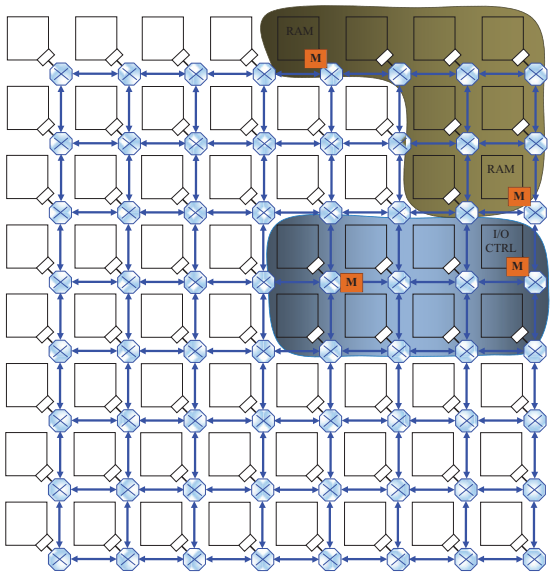

Figure 4.5: Monitor components in a 64-node NoC attached to network interfaces of memory and device controllers and to critical on-chip routers.

Figure 4.5 shows an 8-by-8 NoC with two virtual groups of cores, mainly comprised of processors, memories, controllers and coprocessing units. Unfortunatelly, developing a second NoC to support monitors sitting at each network interface and system unit costs significantly in terms of silicon area. Thus, as figure 4.5 depicts, the system developer can select critical locations to attach monitoring functionality, like interfaces or units which are susceptible to congestion, to performance bottlenecks or to power consumption hotspots.

Flexible resource and monitor management reduces wasteful overprovisioning. To this end a virtual network of monitors built at runtime comprises monitor agents which are not physically tied to particular hardware component. A simple table translation directory-like protocol resolves the mapping during creation and operation of this virtual network. This N-entry mapping table contains the physical location of each monitor, where N is the number of nodes in the system. A status field is included per entry to indicate whether this monitor is shared, enabled or coherent enabled. Monitor components have two options to ensure secure and protected information management. One node can exclusively use a monitor through locking when the "shared" access bit is reset. Alternatively, atomic transactions can be requested by special type atomic commands. The coherent function of a monitor enables the system to keep the extracted statistics, metrics and thresholds coherent across a virtual domain.

Sharing of monitor components and scheduling of the monitor services can be enabled through software together with architectural and circuit support. Spin-waiting and explicitly marked synchronization points, which are common techniques for multiple processors, can be utilized to achieve management of shared monitors in software. However, to filter out unnecessary communication such schemes rely on the programmer or the compiler. Hardware assistance helps to reduce time-consuming software-based mechanisms while at the same time ensure correctness. Figure 4.6 shows the mapping and management functions embedded in the network interface of a tile-based MPSoC. This configuration allows decoupling of the processor from the management of monitor functions. Synchronization, mapping, freeing of monitors is off-loaded to the network interface. This can be accessed through the network without interrupting the processor.

Figure 4.7 depicts different configurations for monitor sharing. One option involves multiple physical monitors attached to a node, creating a group of homogeneous or heterogeneous monitor components in terms of functionality. Each monitor unit can be assigned to a different task (local or remote) ensuring complete independence and security. The drawback is the increased cost in silicon area. Alternatively, a single monitor component can integrate multiple filters and include multiple sets of configuration registers to support access from different masters. In this case, arbitration and

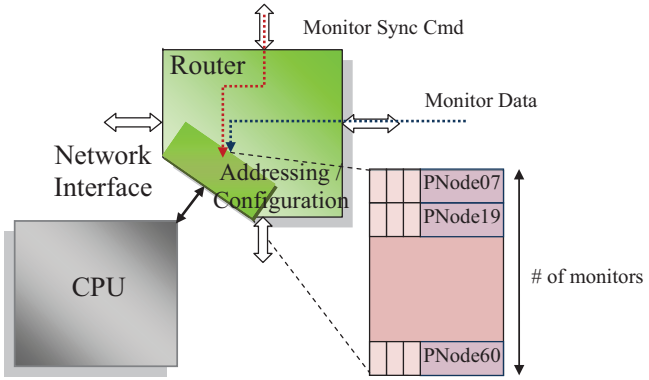

Figure 4.6: Setup of mapping functions and control inside the network interface of a node.

delivery of the appropriate monitor data is performed internally in the monitor component. It must be noted that the physical location of a monitor is a system-level decision, mostly related to the source to be monitored and also, to the distance that the monitor data need to be transfered across the NoC routers.

Figure 4.8 shows the translation operation when configuring monitoring for a task to access particular monitor components which are tied to its parent node. The set of monitors that the system needs to enable for a group of nodes must be appropriately initialized in the translation table and configured thereafter. *Directory-based protocols* can be activated to track critical monitor updates, identify stale information and maintain system consistency. Monitor related data may involve critical configuration thresholds, running counters where accuracy is of paramount importance, e.g., for real-time systems. These can be maintained locally inside monitor buffers, in the local memory, in the cache of the parent node, or in the system memory. When an access request for these monitor data cannot be satisfied, then, a directory provides the pointer of the current updated location of the data. Different structural organization of the directory and its physical location create a wide space of solutions for efficient directory protocols.

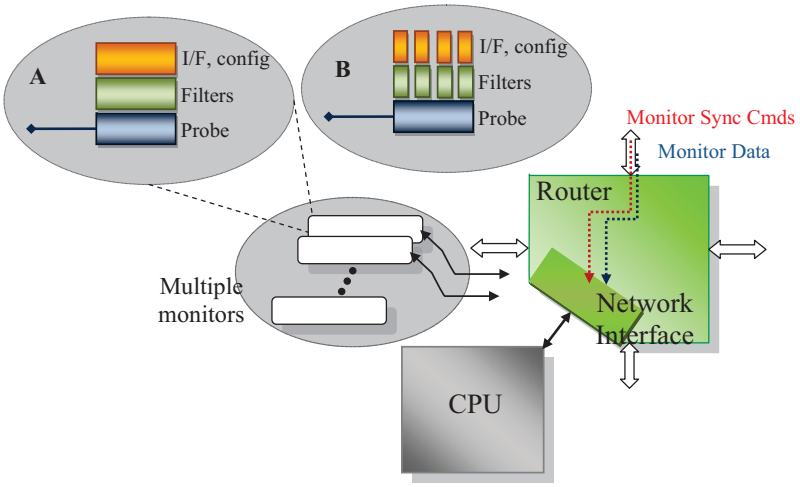

Figure 4.7: Monitor configurations to support sharing among multiple masters

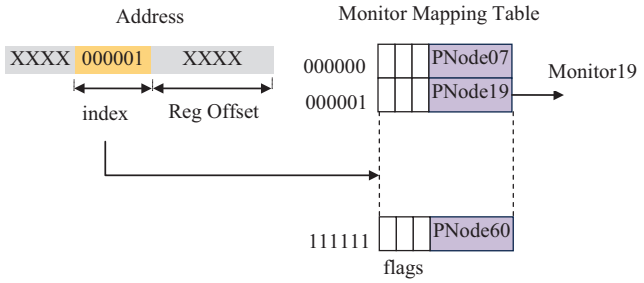

Figure 4.8: Monitor mapping table for each node in a MPSoC

In a multiprocessor system, tasks can be associated to a logical processor that has allocated a subset of the available resources. Therefore, a table is employed to store pointers to the subsets of nodes that comprise a logical group of active computing resources. Each such subset can be

straightforwardly represented as a bit vector, where, for each participant the corresponding bit is set. This first level virtual hierarchy can also be dynamically maintained with nodes joining, or subsets destroyed at runtime.

If a monitor provides read-only statistic information, then, this directory table which contains all the processors enrolled in this virtual group guides the monitor agent or its parent to send point-to-point messages to these processors. Unfortunatelly, more complex support is needed for configuration data, like thresholds, filter masks, and timer values that one processor needs to set to the monitor. For instance, a monitor located at a memory controller will potentially keep accounting for memory transactions to a particular bank, or from a particular source, or identifying DMA transactions from a device, or identifying appication-specific accesses. If two tasks need to set different time intervals for the accounting process then, one solution is for the monitoring component to support a limited set of parallel counters and corresponding finite state machines. Alternatively, a directory scheme can provide information about the current owner of each function of the monitor. The rest candidates can potentially read the counter values of an engaged function, but can alter its parameters only after the owner releases it. The directory keeps bit masks so as to send the appropriate messages.

Building multi-probe monitors with parallelly accessible counters exposes tradeoffs between optimal utilization or performance and protocol and overhead in terms of resources. Energy and area budget can drive design decisions on the number of different sets of counters, timers and corresponding logic used in parallel. Ideally, each monitor component should flexibly assign the needed resources to requesting cores or tasks, reaping the benefits of statistical multiplexing of interfaces and communication links. Despite these sharing opportunites sophisticated monitoring components are rather uncommon choices from complexity and power consumption perspective.

The arbitration and allocation strategies of monitoring resources defines one aspect of developing monitor-aware SoCs. For instance, if two or more concurrent threads act on a shared variable which is monitored by different agents, each one operating in its own clock domain, then the notion of *logical* timestamps or clocks needs to be used. The logical clock corresponding to a process in

the system progresses when it is active and this logical time tracks the logical sequence of events. To monitor events on such shared variables the number of acting procesors is needed to allocate a class of monotonically increasing timestamps.

The next sections discuss the inherent difficulties and challenges of getting the distributed monitors to be matched to the disparate tasks being performed by the system computing components.

Challenges for distributed co-operating monitors

Maintaing a global time in distributed systems has been extensively researched. Logical clocks are the most prominent solution to deal with event book-keeping and distributed synchronization. As the global timing and state of distributed (and asynchronous) systems are characterized by nondeterminism and low predictability, capturing the correct sequence of events that depict the systems run-time behavior is difficult to observe.

Advances in sensor technology (integration, accuracy, sensitivity, linearity, etc.) motivate infrastructures for run-time management based on sensor processing components that share and distribute run-time signatures, voltage, thermal, and error information. Unlike traditional, simple monitoring techniques, recent architectures can include processing components that are dedicated to sensor data analysis and physically separate communication infrastructure for sensor data [209]. Ideas in the scope of improving energy efficiency of the interconnection network of a Chip Multiprocessor (CMP) can apply also apply for the monitoring infrastructure by using heterogeneous interconnection network composed of low-latency wires for critical messages and low-energy wires for noncritical ones [210].

The potential of sharing monitoring infrastructure (sensors, probes, communication links, sensor data processing units) in order to reduce overheads stimulates the creation of cooperation policies. To be effective, these strategies must carefully consider specific attributes of the available infrastructure, such as:

- *Sharing/reuse degree of monitor.* The number of cores/applications that engage a monitoring configuration in its particular location in a time-window.

- *Affinity degree of monitor.* The ratio of usage of one core over the total usage of all cores (in a logical group) for a particular monitor.

Tagging monitors and protocols for consistency

Hardware monitoring units need to be of low overhead to save silicon area and energy. Filtering the traffic of on-chip communication links can extract the useful pieces of information and reduce post-processing effort and memory requirements. However, different processors and threads all interacting with the same underlying physical monitor unit may cause excessive context switches or consistency problems of the information requested by different threads. When the captured information is not directly and immediately sent to the requester and is maintained locally in the memory of the monitor component, then it can become inconsistent. Spin-locks and atomic operations can protect and limit access to the monitor component to prevent inconsistencies, but can potentially cause other problems, i.e. deadlocks, starvation, etc.

To rectify this potential hazard the notion of tagged monitor is introduced. Instead of generating arrays of monitor components, a solution is to tag the captured traces with the core/process identifier (CPID). In this regard the monitor local memory can be organized into data and tag fields and act as a content addressable memory (CAM). Alternatively, a partitioned memory per CPID offers the advantage of reduced cost but decreases the utilization.

Moreover, such a monitor component which offers shared services should support multiple filters organized into a parallel or hierarchical scheme. Each filter has a configuration port in order to be programmed on the fly, and the data port which processes the captured data. Multiple filters can be structured in a parallel configuration, all operating on the same data sample at the same time. To organize the filters in a hierarchical way the monitor component must infer the common part of all filters and apply this on each data sample. After, the remaining part of each filter must come into action on the result of the previous operation. This filtering style can be applied on transactions with a particular structure, e.g. AXI or PCI Express. Matching or excluding a captured sample can proceed through processing the transaction fields in a hierarchical manner.

Filtered data either must be logged for further processing by the requester CPID, or signify an event and must update the corresponding counter owned by the CPID. Both cases require a high throughput memory store, and/or FIFOs to regulate different rates in generation of events and storage of the results.

4.4.1 Monitor data processing for NoC-based MPSoCs

Networks-on-chip have alleviated bandwidth bottlenecks and long wire delays. However, they suffer from large communication latencies. Although many techniques have been proposed to reduce on-chip communication latency between processors and memories or custom processing units, there is an inherent limitation in reducing it. This is because latency is dependent on the communication distance, which is an unavoidable physical constraint. Therefore, modern SoC designs rely on various heuristic techniques which try to reduce or hide communication latency, such as caching, prefetching, smart communication scheduling or intelligent network design methods such as express links [211]

Architectures sensitive to monitor critical information involve decision making for synchronization, deadlock detection, power management. In addition, fine-tuning of system parameters such as cache controller's policies depend on efficient communication of monitoring data. Hence, it is critical to reduce the overhead of conveying monitoring information between the processor and the monitor agents. To this direction one solution, usually combined with hierarchical organizations, involves moving the pre-processing computation to the monitor side, that is the pre-filtering and selection processing of monitor data. This is critical especially for monitor probes attached to buses or other communication links, which often generate frequent (and possibly irregular) monitor transactions.

For instance, during development and in normal operation it is desirable to perform behavior characterization of a hardware or software component, which involves the indentification of resource usage, amount of activity in processing data, consumed or produced data, etc. Such metrics characterize the behavior of a component in time and subsequently entail performance, power or

thermal impact to the system. In general, the metric of interest is the derivative of each attribute of the component in time, as this gives the rate of change with respect to this particular attribute. For example, usually it is important to track the rate of memory accesses of a thread or of a processor to identify its throughput requirements as these change in time. Thus, a monitor used for behavior characterization can feature computation of the derivative of a components' attribute. Temporal derivative refers to rate changes in time, while the spatial derivative can identify the change of an attribute in space, affecting or affected by the nearby environment. Through more accurate understanding of the behavior of a unit the cost-intensive integration of large number of physical sensors can be reduced. Additionally, slow reactive countermeasures can be replaced by proactive measurements in advance.

In the following sections the developed monitoring primitives are utilized in various system scenarios to exploit dynamic behavior identification for different objectives, mainly for performance and power optimization.

Chapter 5

Monitoring for Power and Thermal Management

WILST the complexity of emerging multi-core Systems-on-Chip makes the use of extensive simulation-based power estimation prohibitive, online monitoring assisted by efficient hardware units is prevailing as an efficient method to capture dynamic system behavior.

Power and thermal management are becoming increasingly crucial with modern increasingly complex Systems-on-Chips and at the same time the obstacles in accurately predicting the exact behavior of demanding parallel applications are also escalating. Thus, dynamic mechanisms are becoming significant in identifying efficient multiprocessing task partitioning and mapping while considering performance and power; these mechanisms are the only approach that achieves increased level of confidence and accuracy in dynamic environments. Hot spots in single processor systems, namely the register files, load-store queue, etc, are well understood and often managed successfully, but in heterogeneous multi-core SoCs power density and temperature management is not an easy task. Even with advances in profiling, assigning too much work to slow cores and too little to fast cores may be impossible to predict. The dynamic, semi-random activation and deactivation of certain tasks during run-time is difficult to capture and utilize e.g., to predict the distribution of branch probabilities. Run-time mechanisms are required to balance the performance against power

in applications with dynamic behavior or data-dependent processing that cause fluctuations to power consumption which may exceed the SoC's temperature limit threshold.

The contributions of the proposed methodology include: (i) dynamic task characterization using user configurable hardware monitors easily deployed using reconfigurable technology, (ii) accurate, system-level task throttling to chip's processing units to achieve the desired power envelope via on-line workload monitoring and, (iii) use of multi-disciplinary efficiency metrics to monitor and predict the maximum performance under safe power thresholds, while utilizing coprocessors that accelerate critical code segments.

Instead of viewing the related works as competitive alternatives, this chapter describes a methodology to estimate the power and thermal behavior of applications at run-time utiling hardware monitors. The running task is free from any instrumentation at the software level, while at the same time no custom processor re-engineering is required. This is in line with the common practice for most modern multi-core SoCs which are IP-based, providing little or even no internal information to the system developer.

5.1 System-on-Chip Power Management using Monitors

To dynamically manage the performance-power of a multi-core system-on-chip this section presents an architectural technique which exploits instruction-level parallelism (ILP). Dynamic voltage scaling (DVS) can achieve approximately cubic reductions in power density relative to the reduction of frequency, however it carries inherent overhead due to step size and switching time. DVFS techniques are not easily applicable at the application level in multi-core systems because they assume that critical information about all tasks (i.e. task arrival time, deadline, and workload) are known in advance. Moreover, the workload of a task is often represented by the number of CPU clock cycles required to complete the task regardless of whether the workload consists of mainly CPU-bound or memory-bound instructions, or even worse whether the workload is data-dependent (e.g., quicksort). Throttling a processor can prevent thermal emergencies when the temperature of the hottest part of the chip surpasses the critical temperature limit, no matter if there are other units

whose temperature is far below the limit. However, generally it is not desirable to apply throttling since it reduces not only the processors' power consumption, but also its performance. Balancing temperature within a chip mitigates hotspots and thus also reduces the need for throttling.

5.1.1 System Model

An energy efficient multi-core SoC should manage the available system resources to obtain the required system performance while keeping the system power budget within acceptable limits. In modern SoCs, processors integrate specialized accelerators, memory instances or other heterogeneous components operating at different data rates and executing simultaneously various tasks.

Our approach focuses on integrating a low-frequency processor for standard tasks in a power efficient way, while for increased performance a second high performance CPU is tightly coupled to accelerate processing. The master processor is completely responsible to assign specific workload to the accelerator if more processing throughput is needed. Modern complex SoCs utilize various customized hardware components to improve baseline processing, and in some cases utilize automated customization tools. However, in our system model by choosing a programmable processor-based accelerator, more and more functionality can be added in software without having a lengthy circuit design re-spin. Of course, this increases the needed area budget and is not efficiently targeted to accelerate a specific computation; nevertheless in the scope of the current work we sacrifice performance for the flexibility of a programmable high-speed co-processor.

The master processor is augmented with two types of hardware monitors to observe the activity of the accelerator both in terms of number of instructions executed and in terms of amount of data transferred for processing in a specific time-frame. Thus, useful metrics can be calculated online for the behavior of the task performing data processing and for predicting the amount of workload and corresponding energy needed to process this workload in subsequent time-frames. The characteristics of temporal varying data-dependent tasks can thus be efficiently captured. The hardware monitors are attached on critical observation points, on buses transferring instruction and data. While we do not use caches in our prototype and evaluation, our approach using monitors is

actually independent of cache existence, size, and cache implementation, and can be combined with caches if they exist in a SoC. In addition the uncertainty of fluctuations due to overloading of shared buses and to arbitration mechanism has no impact to our system model as the monitors are attached to the private bus of the accelerator processor. This allows a more accurate characterization of the workload; alternatively, under congestion conditions of shared buses used by many processors, the monitors should operate on a more frequent or even continuous basis.

To facilitate dynamic monitoring of multi-core designs a hybrid infrastructure is developed that utilizes both *hardware monitors* and *software monitor management at run-time* for increased flexibility, which is straightforward and easy to employ. The *non-invasive monitoring* of embedded SoC applications can be considered as the clear value addition by using the methodology described next. To determine the run-time characteristics of a task, a power/temperature lookup table (PLT/TLT) is built with the aid of the hardware monitors. Software agents are responsible to consider the PLT and regulate an evenly balanced energy distribution among all processing cores of the SoC.

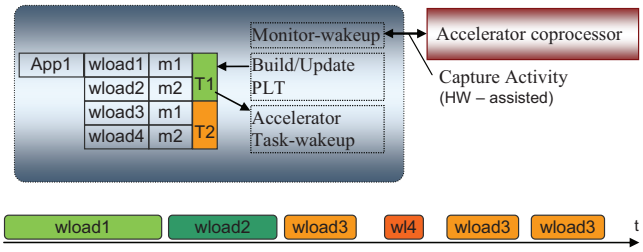

Figure 5.1: System power/thermal-aware operation with the assistance of a software lookup table

Figure 5.1 shows an example organization of the software tasks of the main processor which is responsible to build and update the power/temperature table. Even if the metrics m1, m2, ..., mN, which denote the activity for each amount of workload, may be the same for a specific application, different temperature thresholds dictate providing different amount of workloads to the accelerator.

The table entries wload1, wload2, ..., wloadN stand for the basic chunks of workloads, measured in number of words, that contribute to the power consumption as characterized by m1, m2, ..., mN respectively. A new workload will be segmented to chunks and supplied to the accelerator. When the first chunk is processed the monitor is activated so as to acquire the corresponding metric. The next chunks of data, assuming they follow the same power profile, will be provided to the accelerator based on the selected system policy. This flow can be utilized both for offline characterization and for dynamic identification of workload intensity.

5.1.2 Framework for Real-Time Monitors

Similar to infrastructures targeted to identify manufacturing faults, we modified the design flow to allow the designer to specify the *trace* and the *trigger* signals of each monitor component before running the implementation tools. Then, monitors are generated and seamlessly integrated in an automatic way with the monitor processor. The hardware monitors need to be attached to the critical signals to observe, while the monitors' software interface offers a simple API to the software developer.

The main features of the developed methodology are briefly summarized in: (i) providing an easy, fast and efficient tool to jointly monitor hardware designs and software code, (ii) integrating monitors with the SoC design in a unintrusive way and, (iii) freeing the software application from any instrumentation code run as it would on an actual embedded system. The monitoring framework employs configurable plug-ins which can be specified optionally, depending on the designer's requirements and resource constraints. After the inputs are scanned and processed the configuration plug-ins are considered to produce the final top level synthesizable SoC. The developed plug-ins include a *free-running timestamp timer*, *statistic counters* and a *compression unit*.

The functionality of a Trace Monitor Unit (TMU) encompasses a programmable filter to reduce the amount of captured data and the required time to process these data in software. In addition, the TMU features on the fly update of the filter of the monitor, giving a clear advantage over all other on-chip verification tools, such as Chipscope, or ARM's CoreSight. In the current implementation

the monitoring processor communicates with the TMU using a register based interface. Thus, the filter can be programmed at run-time to mask the traced vector using conditioning circuitry to collect only the samples of interest, and/or use a maskable trigger event to enable breakpointing, or versatile run-time control of the trigger function. To capture desired samples the data must satisfy a group of conditions set by the trigger fields and the masks. However, default masks allow an easy and straightforward usage by the designer. Figure 5.15 shows the internals of a single TMU which is attached to a processor; for our evaluation testbed a MicroBlaze soft-CPU from Xilinx is used as a vehicle. A few words can be transferred via point-to-point links (Fast-Simplex-Links named FSL) utilizing FIFOs for synchronization decoupling.

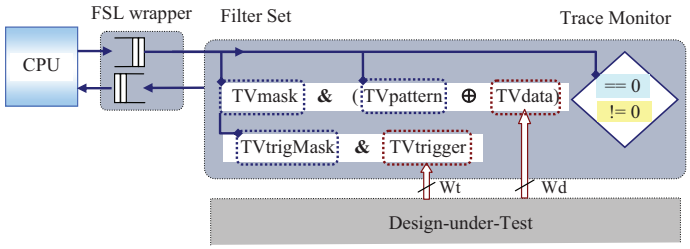

Figure 5.2: Architecture of a single programmable trace monitor unit. The designer specifies the trace vector (TV) signals: TVdata and TVtrigger at configuration time, while TVmask and TVpattern can be programmed and modified at run-time.

TMU is a modular design allowing decoupling of its internal architecture with the type of memory used to store the traces. TMU is created based on the user's configuration of the *TVdata* and *TVtrigger* signals, while the user can optionally program the mask with software at run-time. The TMU component is modular and independent from the FSL wrapper. The FSL wrapper is used to exchange information with the MicroBlaze CPU, receiving the filtered samples and setting the appropriate masks according to the designer's code. However, the TMU module is quite generic and can be used easily in other platforms, besides Xilinx's.

Table 5.2 summarizes the implementation cost of indicative monitor configurations using a Virtex-4 VFX20ff672-10 device; the area of a MicroBlaze baseline core without local memory controllers or instruction and data caches is also depicted for comparison. Unless we integrate complex functions in hardware, such as compression or classification of events, the cost of integrating even multiple TMUs is affordable.

Table 5.1: Implementation cost of hardware monitor units (each block connects to MicroBlaze core with a single FSL interface)

Block	Slices	RAMB16s	Freq
MicroBlaze core v.7.30	1240		134
TMU 32bit monitor	332	3	297
Switching activity monitor	148		387

5.1.3 System Organization and Operation

Employing our automated methodology for integrating hardware monitors we prototyped an embedded System-on-FPGA as shown in Fig. 5.3. The developed prototype limits the achieved frequencies, but the system behavior can give useful insight and expected trends when scaled to today's high frequency SoCs. A low-frequency (75Mhz) MicroBlaze soft-processor on a Xilinx's Virtex4-FX20 device typically executes kernel processing on data stored in either external or internal memory. On performance need, the higher-throughput MicroBlaze is activated to accelerate processing of the workload transferred in the shared memory. The offset of the source data is send via the mailbox hardware block along with the size of the chunk of data. Note that the provided workload can be processed while other data are transferred from off-chip memory since the shared memory is two-ported. The behavior of a task can be traced by counting the switching activity of the processor instruction bus and the switching activity of the data bus of the shared memory in a time-frame. A task running different algorithms successively can thus be characterized in real-time based on these metrics.

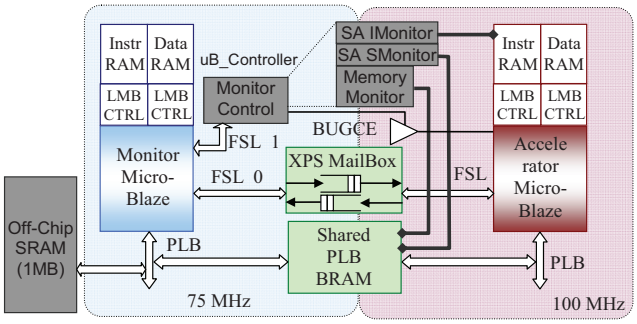

Figure 5.3: System organization based on extensible MicroBlaze soft-CPUs integrated with multiple hardware monitors.

As shown in Fig. 5.3 the master MicroBlaze manages three monitors which are directly connected via FSL links. The monitor attached to the shared memory is responsible to start the integrated 64-bit time counter when the accelerator task begins and to observe the progress of processing. When a data intensive application is executed the monitor is set to observe the address of the shared data buffer, so as for the master MicroBlaze to initiate the transfer of processed data back to the main memory (off-chip SRAM) since the size of shared memory cannot account for large data sets.

The monitor also detects task completion by observing when a predefined memory location is accessed; in all scenarios presented next this memory location is set to the high address of the shared RAM. When this event occurs, the *uB_controller* accesses the switching activity monitor to provide the current value; an internal 32-bit divider (8-stage operation) is then activated. The internal monitor timer value is divided by the number of switchings and the quotient and fractional part are maintained in registers of *uB_controller*. After the result is ready the *uB_controller* suspends the accelerator processor. In the meantime the accelerator processor has already sent the ”done” message via the FSL connection before put to sleep. Thus, the main MicroBlaze is notified that the task is completed and collects the calculated ratio of time/transitions.

5.2 System-Level Power Management

To demonstrate the operation and benefits of our proposed technique, we select a set of three application kernels, namely IDCT, ADPCM and Sobel, each with different behavior; these are assigned statically to a random predefined number of neighbor processors, each of which consists of the tightly coupled dual-core processing node described above. The objective is to integrate the monitoring information as extracted by the monitors prototyped on FPGA with a system-level power management layer in software. Inverse Discrete Cosine Transform (IDCT) is a well known processing step in image decoding. The numerous computations involved in IDCT make it a candidate for performance improvement with the aid of an accelerator co-processor. The kernel processes blocks of 8×8 DCT matrices. We also use a simple 16- bit PCM to 4- bit ADPCM coder kernel, which is also a CPU intensive algorithm. The co-processor works on blocks of sixteen integer raw samples at a time. The third application is the Sobel edge detection algorithm, which performs partition of the pixels into edge and non-edge ones.

The measurements from the prototype are compared with (a) power estimation results from XPower from Xilinx after extracting full system switchings in VCD format and importing them to XPower matching 80% of the nets, and (b) using actual current measurements with a power/current sensor, chip INA219 from TI. Figure 5.4 shows this comparison plot for various applications including MD5 hash and least squares calculation.

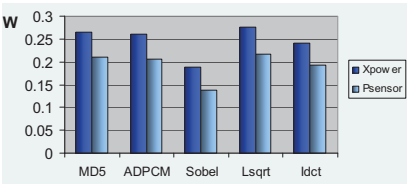

Figure 5.4: Power measurements using simulation and actual power sensor device

Using the real-time monitors the measurements of instruction activity ratio for a time-frame of 1M clock cycles give an average of: (i) 628 for Sobel, (ii) 70 for IDCT and (iii) 54 for ADPCM. The hardware monitor is set to compute and provide the result of:

$$\sum_{t\_0}^{t\_timeframe} Time_{(clock\_cycles)}/Activity_{(Instruction\_bus)}$$

for the time interval that the accelerator is activated and processing the provided data. The lowest value ratio indicates the most computation intensive task; our comparison graph (fig. 5.4) argues in favor of this conclusion. Now, a simulation model for a 4×4 multi-core system is constructed using these metrics; each core follows our dual-processor model. A potential mapping of these three tasks is shown in figure 5.5.

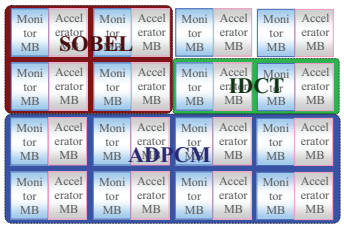

Figure 5.5: Simulation testbed organization with three applications: Sobel, IDCT, and ADPCM statically mapped on a 4×4 multi-core system

One processor in a cluster, for each application, or for the entire SoC, is assumed to be in charge of managing each processor to apply a power policy; alternatively an individual processor may periodically consider the system power and trigger the rest ones. Inter-core communication cost via FSL direct links is less than ten clock cycles. Each processor starts executing a task with a fraction of the whole workload, while feeding the accelerator with parts of the workload depending on the power policy. When the accelerator completes its task, it is put to sleep by freezing its clock.

In scenario shown in fig. 5.6 each processor wakes up its accelerator and provides increasing

amounts of workload to process, while the system power is monitored. Note that for the moment we consider a symmetric case in the sense that each core is given equal priority to increase its throughput proportionally.

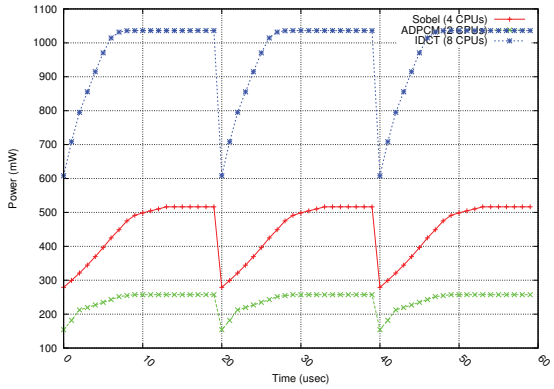

Figure 5.6: System-level symmetric power management per application

In our experiments on the FPGA platform the MicroBlaze accelerator consumes 130.144 mW in average in typical operation. This value is used for the simulation runs, due to unavailability of a larger FPGA device for prototyping the entire system under test.

Since modern SoCs have multiple thermal sensors in various places on chip, the motivation is to evenly distribute the power variation in the neighbor cores and consequently to a wider area in an even way (assuming that processors which are logically zero-hop away are also physically located nearby).

In our simulations we use a simple adaptation of the Floyd-Steinberg dithering algorithm [212] for images; we modified it to apply to all cores so as to diffuse the steep changes of power hotspots. This results in exponential changes of power as shown in figure 5.6.

Thermal emergencies require immediate and drastic treatment, thus leading us to employ an algorithm with exponential effect. On the other hand, real-time tasks may require instantaneous

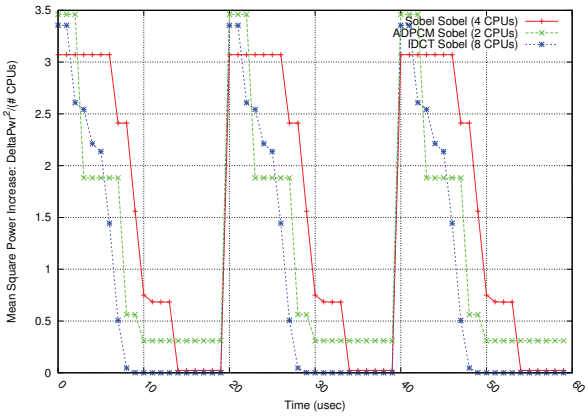

Figure 5.7: Normalized changes of power increase under system-level power management

increase of performance and hence full performance of the accelerators on the spot. The figure agrees with the latest scenario as rapid activation of all cores (in exponential fashion) is shown to occur three times. Figure 5.7 shows the difference at each change of the power normalized over the number of processors for each application kernel. Note that due to very few number of processors for Sobel and ADPCM jagged steps are shown in the plot.

The modified Floyd-Steinberg dithering algorithm used in our system typically proceeds from the top row of processors to the bottom row dispersing the desired change to the neighbor processors. We modified this algorithm (the directional wave of change) in order to consider the fact that a specific application is mapped to a cluster of processors which not necessarily occupies all resources. When a group of processors is in full performance and draws the maximum power, the SoC power management master can throttle the workload of this cluster and additionally the workload of adjacent processors by the weights similar to Floyd-Steinberg algorithm. Fig. 5.8 displays the even reduction of power for all surrounding processors of the selected application kernel to be throttled. First, Sobel is throttled affecting the adjacent cores that execute a different application kernel, and then, IDCT is throttled after time 20. Since IDCT is assigned to two cores in row two, all kernels

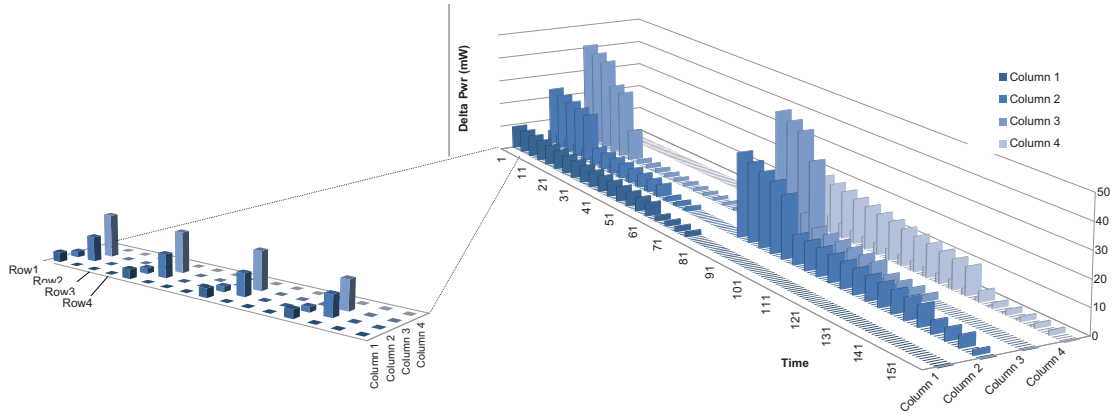

Figure 5.8: Independent workload throttling for power management of Sobel kernel ([Row_1, Column_1] – [Row_2, Column_2]), followed by throttling of IDCT kernel ([Row_2, Column_3] – [Row_2, Column_4]). At each time slot one instance of the 4×4 SoC is plotted adapting power with regard to its adjacent cores.

are now affected as shown in Fig. 5.8; note that only the surrounding cores modify their power consumption. In all these cases the center of the multi-core SoC is mostly affected, which is usually the most important hotspot on a chip that requires emergent attention. Thus, even if the tasks possess different characteristics regarding the intensity of utilization of chip resources a system-level approach is shown to manage uniformly the power requirements under safe thresholds.

A novel dynamic management scheme supported by hardware monitors is introduced to address real-time monitoring of multi-core SoCs for power and thermal management in an non-invasive way. While the area cost of such monitor units is affordable, the flexibility and more importantly their effectiveness to provide useful metrics in almost real-time are invaluable. We modeled a multi-core system-on-chip with the capacity to detect and predict at run-time the power requirements for different kernel applications. Moreover, we presented a distributed algorithm to diffuse power changes smoothly to neighboring cores. The evaluation of the presented infrastructure shows that

inaccuracies of off-line data profiling or process variation effects of multi-core chips, can be easily overcome with the aid of hardware monitors which provide metrics representative of the intensity of a running task in less than 10 clock cycles. We believe that particular hardware metrics collected at run-time and corresponding empirical application results can provide a basis for future power-aware research.

5.3 System-level Run-time Monitoring

To face the exposed needs of power- and thermal-aware computing at block and system level, in this work we demonstrate a heterogeneous architecture featuring multicore co-processor accelerators and real-time power and temperature sensors per node, which is realized using a reconfigurable prototype platform. Using the proposed solutions the developer can perform co-evaluation of adaptive optimization techniques with varying application behavior which is essential to understand the interplay of hardware and software schemes on heterogeneous NoC-based designs.

The developed platform provides centralized and distributed mechanisms, along with prediction algorithms, DFS and workload throttling, to support dynamic runtime control and adaptation of power consumption both at local and at system level. These are exploited to enhance designing of FPGA-based energy- and thermal-aware embedded systems, or in the scope of emulating power-sensitive SoC components, policies and applications. Actual current and temperature sensors per node are sampled to ensure that the NoC-based MPSoC operates within a safe operating range and at the optimal performance level. Thus, one can explore workload distribution to processors based on their heat distribution capabilities while ensuring that each processor has at any given time a good mix of high power, *hot*, and low power, *cold*, tasks. A number of thermal prediction and management techniques have been proposed in the literature [73, 213, 214] which generally fit a mathematical model to a local window of observed chip temperatures. Others use performance counters and core utilization metrics to estimate the power consumption and core temperature. We preferred the direct approach through current and temperature sensors, which are available in most modern processors. Our work shares similar ideas with Cochran's and Reda's [215] methodology on

temperature prediction and proactive control to handle thermal coupling between cores, but extends it to heterogeneous NoC-based systems using reconfigurable technology.

Another hardware related approach to avoiding hotspots, which is in the same spirit as our proposal, lies in replicating processor resources such as register files, issue queues, or ALUs on-chip and migrating computation to one of those spare resources when the original resource reaches a critical temperature [216]. The main overheads are silicon area increase and longer wires, while lost cycles to support activity migration are negligible. Our scheme also spreads out computations to hardware coprocessor units and adjusts their clock rate while controlling system's power dissipation and temperature at real-time.

Ou and Prasanna demonstrate dedicated power management hardware peripherals tightly coupled to a single soft processor which runs a real-time operating system [217]. They examine the power efficiency of the proposed scheme when selectively waking up the soft processor and its hardware components, and put them into proper activation states based on the hardware resource requirements of the tasks under execution. We have extended their scheme in many ways, offering multiple coprocessor cores operating at four different frequencies, tight cooperation of hard with soft processors, and a high-speed interconnection scheme to connect multiple islands together. Additionally, hardware support is provided for message passing and shared memory programming models.

As presented, the design space has numerous aspects. Our reconfigurable system is designed and implemented to utilize many different power- and temperature- aware methods found in the literature, which in conjunction with our own can be tailored to increase the efficiency of modern MPSoCs.

5.4 NoC Node Architecture: a Multi-core Island

Figure 5.9 shows the microarchitecture of a NoC-based system-on-chip where each node consists of a single master CPU and a number of co-processing elements to accelerate application tasks. This design structure complies with the concept of voltage-frequency *islands* (VFIs), which has been

recently introduced [218] for allowing system-level power-efficient management. Inside a single node all processors are attached to a bus and additional point-to-point links connect the master CPU with the accelerators. At system-level the network-on-chip that connects the nodes is a two-dimensional torus as depicted in the figure. Given a particular application domain in the era of embedded SoCs, computation-intensive functions are usually selected as candidates for off-loading the general purpose CPU to customized hardware components [219].

Typically, in power-conscious computing the control CPU executes a task from its local memory and, if the power budget of the processor cluster allows then, a number of helper cores are activated to accelerate compute intensive tasks. A management thread is responsible to estimate the appropriate number of cores along with their operating frequency in order to keep the power consumption under valid limits. If temperature reaches hazardously close to threshold temperature then the manager thread can shut down the co-processors after the current workload is processed. To alleviate such critical situations various emergency policies can be investigated on our platform in terms of response time and efficiency. An *efficient* policy should feature low complexity and incurred hardware (silicon area and power consumption) and software (CPU processing time) overhead.

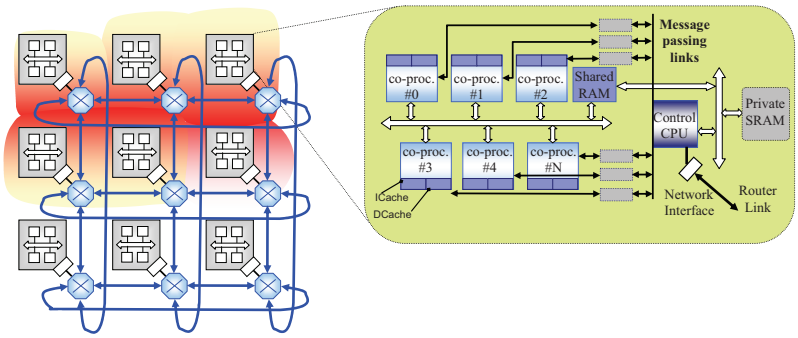

Figure 5.9: Organization of a heterogeneous NoC-based power-aware MPSoC.

In a NoC-based multicore environment, power consumption and thermal effects of adjacent

cores should be considered. During runtime, brute force thread scheduling to idle cores can result in uneven amounts of activity in different cores making the system susceptible to hotspots. Usually, the task distribution on NoC-based systems is performed primarily for performance and communication constraints (since interconnect delay is a major component of performance). Therefore, thread scheduling is unlikely to present a risk-free solution in terms of power and temperature distribution and localized activity may prevail leading to thermal emergencies. Nevertheless, thermal-aware task distribution will certainly be an important component of the thermal-aware system design paradigm and our techniques would co-exist with them in a comprehensive solution.

The runtime management system employs various power and thermal techniques in a synergistic way. Each node integrates monitoring threads which utilize hardware resources and advanced software techniques to explore different dynamic management policies. These resources include: (i) a current sensor to provide power measurements to assist in real-time estimation and characterization accommodating for application or process variability, (ii) a temperature sensor to provide the node's thermal behavior at real-time, (iii) a lookup table with pre-calculated power requirements per application type and per core, and (iv) a fast prediction algorithm utilizes history to predict future workload acting in a pro-active way.

Modeling and estimating of the power of the system interconnect is difficult to be addressed accurately and at a reasonable speed through multiple abstractions. The employed method that we utilize with current sensors per node can effectively provide reliable results for the power behavior of complex communication schemes and protocols. This is important to note especially when these communication schemes are combined with advanced processor microarchitectures and workloads variability.

5.4.1 Characterization for Power

Besides on-line monitoring and system-level regulation in presence of power transients, the integrated sensors are also used to acquire actual measurements during the characterization process. Empirical modeling provides distinct advantages over analytical and simulation modeling because

the evaluation is performed on a real prototyped system. The speed of the emulated system, which can be four orders of magnitude higher compared to simulation, is an attractive alternative. Limitations of experimental methodologies may involve inaccuracies due to mis-calibration or weaknesses to capture very fine events. The latter may appear during the evaluation of the sensors' readings which can be a cumbersome process due to slow transfer or interpretation of their values. For instance, the time interval between two values extracted from a current sensor and transferred over a slow I^2C bus can be large enough to hide current spikes due to hardware events. Simulation results thus can help in discovering such events and in validating the experimental measurements.

In addition to empirical models, we developed an activity-based coarse-grain power model for a core in order to generate a lookup table with core power states. Each core can operate at four different frequencies f_0, f_1, f_2, f_3, where f_0 stands for idle. Disabling the clock (frequency f_0) is preferred than engaging the core to an idle thread, since power would still be wasted. Thus, only leakage power is consumed by the core when it is deactivated. These four power states for each processor core allow for a very fine determination of the power levels that each node can reach. Actually, six accelerators operating at four different frequencies give $\binom{4+6-1}{6} = 84$ valid states. This implies fine-grain distribution of power transients as well.

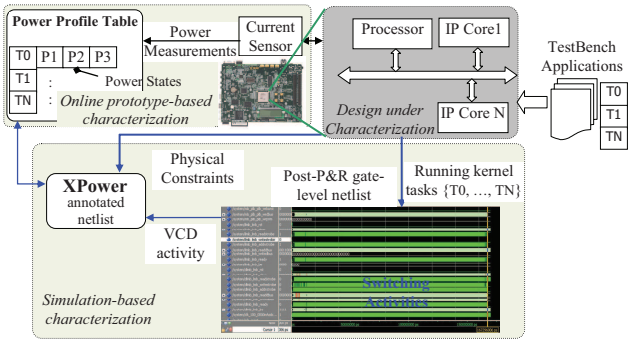

Figure 5.10: Methodology for power characterization of each processing core

Each application benchmark gives a set of power profiles for the adopted core frequencies. The system comprised of the gate-level netlist and the executable code, are loaded to the simulator to obtain the signal toggle information. By dis-assembling the code we identify the critical instructions that define the right time window to capture this toggle information. For the kernel benchmarks we utilized it is an easy task to locate the first and last assembly instructions in hexadecimal which are used to set the start and stop breakpoints in the gate-level simulation. Behavioral simulations also verified the desired time window. The netlist, the physical constraints of the system-on-chip, the power libraries and the switching activities captured during the most critical time windows (captured with timing simulations) are loaded to a commercial tool, XPower from Xilinx, to calculate the required power per system component. This flow is automated with scripts that handle loading of different kernel benchmarks to the simulator.

5.4.2 NoC Communication Protocol

To implement a power-aware emulation platform, we need to communicate the power and temperature status of each node to the neighboring ones. Assuming independent workloads and communication patterns across a NoC, it is important for each node to broadcast its power budget status regularly. Power transients may thus cause increased number of messages, while temperature effects are more mild, appearing in the order of hundreds of microseconds, or even milliseconds.

Considering these requirements the system-level NoC which supports data and control communication is organized as follows. The transported data words, called flits, are 4-bytes: a 2-bit type field is used to provide control information alongside the 32-bit data field. Three types of flits are used: header (H), payload (P) and tail (T). The H flits utilize the *number of flits* field to indicate remaining flits to follow the current flit; thus, for control flits we use zero in the number of flits field to save the last T flit. In the developed platform we assume in-order fault-free delivery of flits and thus this field, acting also as sequence number, is not used. The payload flit uses “00” in the 2-bit type field.

The system ensures that communication links and network interfaces will not be burdened with

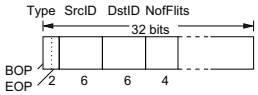

Figure 5.11: Format of fixed-size 32-bit flits

island-to-island task traffic and degrade system response to crucial information, such as flits containing notifications of critical power transients. Each island performs power-aware computations in a distributed manner. However, when adopting centralized policies, the transmission of both flits with monitor information and of flits with reaction commands must be guaranteed through the use of priority communication, or through guaranteed throughput services [220]. Hence, emergency flits are prioritized and use "11" in the 2-bit type field. Moreover, the communication system is over-designed in order to support monitoring traffic, potential unbalanced or asymmetric use of the platform nodes and virtual channels providing protocol-level deadlock freedom. Additionally, to reduce traffic, monitor filtering can be employed, as we demonstrated using hardware programmable filters (citation blinded). The platform is scalable to N$\times$N mesh with well-understood dimension-order routing algorithms, like deterministic XY routing, where all packets are first routed in the X dimension followed by the Y dimension. To avoid routing-induced deadlocks two virtual channels are used, with a buffer size of 16 flits per virtual channel and and ACK/NACK flow control. Splitting physical channels on cycles into multiple virtual channels and restricting the routing, the dependence between the virtual channels is acyclic. Note though, that we focus on power and thermal- aware computing in a multi-island microarchitecture. Thus, we opted for mapping accelerators to available device resources rather than mapping complex router components, as discussed next in section 5.6.5.

5.4.3 Task Scheduling

It is crucial to guarantee application performance, such as throughput and latency, to avoid severe quality degradations, or deadline misses. Thus, each multicore embedded system in general requires end-to-end application guarantees beyond per-task guarantees. On top an energy-aware scheduler should respect performance needs and power budget and dimension these requirements on the fly since, often, the behavior of data dependent tasks can hardly be predicted accurately.

Utilizing hardware support, mechanisms are employed so that:

- each task provides notification for its progress (to detect on-time termination), and
- to determine memory space for input data and output buffer availability (to determine task readiness).

The power dissipated by our MPSoC at any time window T_{win} can be described in the form:

$$P_{Twin} = \sum Etask_i/Ti + P_{idle} + P_{leak}$$

The term P_{idle} accounts for the power consumed when a processing unit sits idle, which can be controlled with clock gating, while the term P_{leak} becomes increasingly important for ultra deep sub-micron technologies; this term is mostly technology dependent and is not addressed for the moment.

The target is to achieve the required performance while not exceeding the system power budget for each T_{win} period. Each task can generally be divided in n subtasks, each processing $1/n$th of the workload and consuming equal portions of energy to complete. In a multiprocessing environment other tasks require variable amounts of shared resources, i.e. bandwidth or buffers, thus causing additional energy consumed on arbitration, stalls, etc. Thus, the total energy to complete a task can be expressed as:

$$Etask_i = n * E_n + k * E_{comm}$$

where $k * E_{comm}$ expresses the energy due to communication overheads, and k depends on the

system configuration and arbitration mechanism and depends also on the communication behavior of applications and on the load of the system. The objective of minimizing the communication energy on NoCs has been tackled by researchers [221].

If the time window to execute each task is dimensioned, i.e., by changing the operational frequency, or by applying stop-and-go execution, then the required energy may be distributed evenly in time so as to avoid fluctuations of consumed power, with a negative effect to performance. Execution though in a larger time period may increase E_{comm} due to the larger possibility to collide with other tasks. On the other hand, processing the same amount of workload in shorter period of time requires more bandwidth from the shared resources; the ratio of the energy due to arbitration over the energy due to processing may be larger in this situation. Moreover, different applications with different attributes and varying behavior over time increase complexity. These challenges have triggered researchers to investigate how to achieve optimal scheduling in a dynamic multiprocessing environment [222, 223].

Merkel and Bellosa [224] argue in favor of complementary mixing memory-bound and CPU-bound applications at any given time period on a single multicore platform so as to achieve the best energy-delay product (EDP). While their approach focuses on temporal scheduling of applications Zhang et al. [225] prefer to group applications with similar behavior to run on the same multicore processor chip. They use spatial partitioning of application over multiple chips thus being able to individually control voltage and frequency reducing heat dissipation. The trade-offs between temporal and spatial hot spot mitigation schemes triggered Choi [226] to explore them on a Linux-operated POWER5 SMT system. Either approach is interesting to be explored on a NoC-based emulation framework studying the benefits or adverse effects on co-running applications on sibling cores.

The following section describes the proposed emulation platform which, based on current and temperature sensors, offers the potential for power- and thermal- aware processing using DFS techniques; the use cases described next provide early results exploring trade-offs for energy-aware computing with on-line monitoring.

5.5 Prototype Platform

Each node of the emulation platform is actually a multicore SoC which employs an embedded PowerPC (PPC) hard-processor and six MicroBlaze soft-processors mapped on a single FPGA device. The six soft-processors act as accelerators of the PPC that executes a computation intensive application in order to deliver considerable performance gains. Computation kernels are carefully selected off-line and pre-assigned to the accelerators, which are subsequently activated by the PPC when increased performance is desired. Opting for soft-processors instead of customized special-purpose units does not provide for the maximum performance enhancements but gives ample flexibility to explore and accelerate tasks from different domains. Particular application-specific acceleration using custom hardware to augment the instruction set of a core processor can still adopt our methodology. On the other hand we have witnessed the development of embedded processors that utilize scratchpad memory (SPM) to replace traditional caches in order to achieve lower power consumption. For instance, in the IBM Cell BE [227], a PowerPC engine (PPE) acts as the control plane processor that hosts the main memory and each synergistic processing engine (SPE) uses a 256KB SPM for both code and data. The problem of scheduling stream applications onto SPM enabled embedded processors has been addressed in the literature [223].

In the developed FPGA platform, PPC performs node-level power management by allowing the power of each computing unit to scale with changing conditions and performance requirements. A single thread running on the PPC is responsible to acquire power and thermal measurements from external sensors. The collected data are exploited by co-operative threads which perform power-aware application acceleration and communication with the co-processors. Algorithm 1 depicts the pseudo-code employed to manage the co-processors in a greedy manner based on power pre-characterization of each application, as discussed in section 5.4.1. However, our algorithm follows the same principle as Zhu's [228]. In contrast to the naive, greedy approach to minimize the current chip temperature by executing at each step the coolest available job, our algorithm attempts to push the processing capabilities of the node, which actually increases the chances of exceeding the temperature thresholds. As Zhu proves [228], given two jobs, one hot and one cool, executing the hot

job before the cool one results in a lower final temperature than after the reversed order. The main idea is to run the clock frequencies at a higher level than previously for a short period of time, even if we hit the node thermal limit, thus increasing overall responsiveness. The workload processing is assumed to consist of discrete phases, or several iterations which can be divided to almost equal parts. Thus, *while* loop of line 2 is invoked for each part of the workload. To reveal the capabilities and limitations of the emulation platform we present selected experiments in the following sections. We also examine prediction algorithms to proactively manage power and temperature of each multicore node independently and in-cooperation with neighbor nodes.

Algorithm 1 Pseudo-Code for System Power Management Unit

```
1:  P_MicroBlaze = lookup(Application_j) ;
2:  while WorkLoad = TRUE do
3:     for all MicroBlaze_i do
4:        activate MicroBlaze_i; sum(P_MicroBlaze) ;
5:        break if ∑P_MicroBlaze exceeds power budget ;
6:     end for
7:     calc P_total = k*P_comm+∑P_MicroBlaze;
       /* critical-red limit: P_thresh1 */
8:     if P_total > P_thresh1 then
9:        de-activate all co-processors except PPC thread
          /* alert-yellow limit: P_thresh2 */
10:    else if P_total > P_thresh2 then
11:       slow down all co-processors
12:    end if
13: end while
```

At system level the goal is to balance power consumption across all NoC nodes. In order to take into account activity intensity of neighbors, P_{total} is communicated to all neighbors and each PPC is responsible to update the local thresholds $P_{thresh1}$ and $P_{thresh2}$ and regulate $\sum P_{MicroBlaze}$ accordingly. Depending on performance needs and response time of emergency techniques, an alternative policy can consciously utilize coprocessors considering the operation between $P_{thresh2}$ and $P_{thresh1}$ as acceptable for short time intervals even if the device is allowed to suffer. Section 5.6.5 details implementation results.

5.5.1 Dynamic Clock Management

The Virtex-4 FPGA clock architecture provides a straightforward means of implementing clock gating for the purposes of powering down portions of a design. Thus, as figure 5.12 shows, all logic related to each MicroBlaze processor, the processor and corresponding local instruction and data RAM of eight KB, are driven by an independent block of two cascaded single BUFGCTRL primitives which switch between 50, 100 and 150 MHz clocks, or disable the clock. If no co-processor is active then, the clock gating circuitry dedicated to control one block can disable the clock of the coprocessors' bus (OPB) as well. Overall, figure 5.13 shows the clock generation scheme on the FPGA device. The PLB and OPB buses allow connected devices to operate at certain relationship (1:N) to its own frequency. Stability and reliability reasons led to the selected clocks shown in the figure.

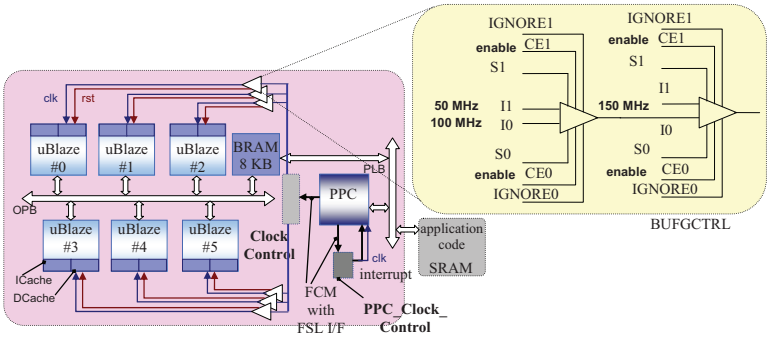

Figure 5.12: Organization of a multi-CPU island node; independent BUFGCTRL primitives are used for per-core power management of MicroBlaze soft-processors. Data/control communication links via FSL between the PowerPC and each soft-processor are omitted for brevity.

The PowerPC (PPC) hard processor manages six independent BUFGCTRL components via the Auxiliary Processor Unit (APU) unit, thus controlling each (co-)processor MicroBlaze individually.

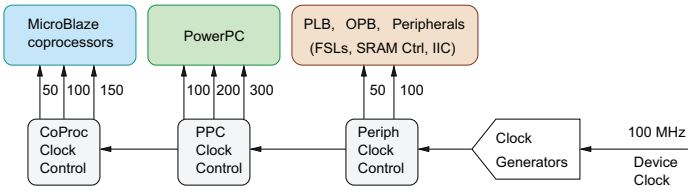

Figure 5.13: Clock domains across a single node

Each co-processor is not allowed to switch its frequency during processing or task preemption or blocking, so as to manage the frequency switching overhead and guarantee task schedulability. When processing of the assigned workload is completed, each co-processor interrupts the PPC to collect the results via messages through Fast Simplex Link (FSL) FIFOs, or via the shared block-RAM (BRAM) depending on the amount of processed data (i.e. the application). Immediately after that, PPC de-activates the co-processor by disabling its clock.

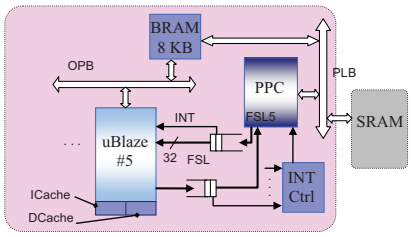

Figure 5.14: Communication among each MicroBlaze and PowerPC. Point-to-point messages are exchanged via interrupt-driven FSL connections. On top, a two-ported 8 KB BRAM connects the private PLB bus and the shared OPB bus of the accelerators.

The FSLs are dual clocked to support asynchronous communication. The write operation is synchronous with the clock of the sender processor while the read operation is synchronous to the

clock of the receiver. Each FSL communication link may store 512 words/messages of 32-bits.

Figure 5.15 shows the layout organization of each multicore node of our emulation platform. The rest device area up to 96% is occupied by two Aurora link controllers, the buses related circuitry (PLB, OPB, FSL channels, interrupt controllers) and the peripherals UART, LCD and external SRAM controller. The LogiCORE^{TM}IP Aurora 8B/10B core is a high-speed serial solution based on the Aurora protocol and the Virtex-4 FPGA RocketIO^{TM}multi-gigabit transceivers (MGT) [229]. Two Aurora components are integrated inside each node utilizing the two SATA interfaces of the board, which operate at 1.5 Gbps. Each Aurora core is configured in streaming mode, single-lane and the MGTs in the core are set to use 4-byte SERDES. A single node is implemented on a

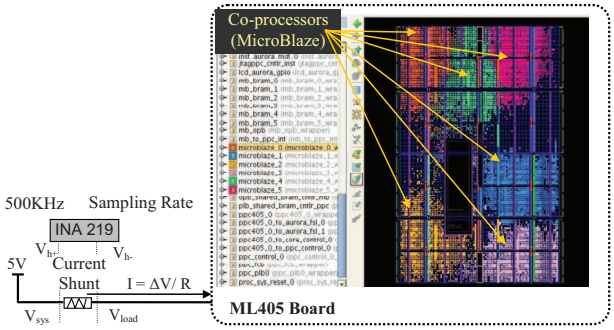

Figure 5.15: Layout (PlanAhead view) of the testbed platform

ML405 platform from Xilinx. The board integrates an on-board temperature sensor MAX6653 from Maxim; we utilize the temperature monitoring pins which are connected to the FPGA TDP/TDN pins that access a temperature diode in the FPGA. In addition, we adjusted a power/current sensor, the device INA219 from Texas Instruments (TI), similar to the setup of the power model as described by Rotem [230] which records the voltage and current of a commercial ASIC CPU.

Table 5.2 summarizes the implementation cost of the major components inside one node.

Configuration	LUTs	Primitives	Freq (MHz)
MicroBlaze (v.7.20.d) (area-optimized: 5-stage)	1330	3273	150
FSL clock controller	86	137	380
Aurora single-lane 4-byte (using SATA i/f @1.5 Gbps)	515	1213	150

Table 5.2: Implementation results

5.5.2 PowerPC Dynamic Clock Management

Dynamic clock management of coprocessors is managed by the island master processor, PPC, at assigned processing boundaries. When the PPC is interrupted from a coprocessor to signal the completion of processing, then the PPC can provide the coprocessor a new load to operate on, along with a new clock rate. However, a few intricacies arise when the PPC decides to change its own frequency dynamically. High probability of instability clearly rules out on the fly change of clock rate; the safe approach is to force the PPC to enter the sleep mode before this change. The clock and power management (CPM) interface of the processor is used to apply *global gating* to the PPC405 core, the timer and the jtag clock zone. In order to prevent unpredictable behavior after sleep mode and change of frequency, a PPC reset request is asserted to "wake up" the processor and enable normal operation. Before transitioning to a different clock rate, PPC needs to complete its own processing, drain its pipeline, serve any pending interrupt or wait for the APU controller if the Fabric Coprocessor Module (FCM) attached to the APU is currently executing an instruction. Next, a *hybrid* restart is initiated at the desired clock rate. This *hybrid* restart includes a custom finite state machine which is attached to the PPC through an independent FSL bus, a special interrupt service routine (ISR) to prepare for the transition to a different frequency, and finally, a modified boot-loader for restoring the processor state. Interrupts to the PPC are context-synchronizing events. All instructions preceding the interrupted instruction are guaranteed to have completed execution when the interrupt occurs, while all instructions following the interrupted instruction are discarded.

Figure 5.16 shows the four phases of the hybrid mechanism implemented to achieve this. After

PPC_DFS_CTRL receives a command with the desired frequency it asserts the interrupt to PPC. In this phase the transition is set up. The current instruction taken from the link register is saved before put to sleep. The address before the last physical one in the SRAM used to identify that the processor executes frequency switching. The last physical address of SRAM stores a pointer the address following the instruction that activated the sleep mode.

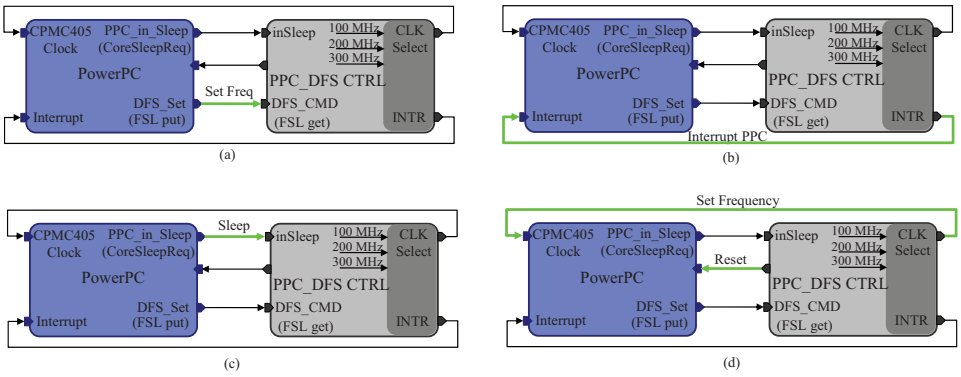

Figure 5.16: Hybrid dynamic frequency scaling mechanism of PowerPC

When the processor switches to sleep mode in phase (c) the FSM controller selects the desired frequency and optionally sits idle monitoring the activity at the incoming Aurora interface. Finally, in the last phase, the processor is reset and at the same time switching happens at the new clock rate. A new developed boot-loader is responsible to resume the ISR after resetting the PPC. Thus, the ISR completes after the reset and clock rate switching happens, since the processor had fall into sleep without restoring the environment.

In order for the CPU to be restored in its previous state from reset, we needed a new “resume_on_reset” patch to the bootloader along with the last value of the link register. These are saved to SRAM from an interrupt just before the core is put to sleep. The SRAM contents never alter before the CPU is restored. During the CPU reset, all crucial data (stack, heap, variables) are stored

in SRAM and if these were altered then resuming the CPU would fail.

5.6 Evaluation Results

In this section we evaluate the emulation platform in standalone mode at first, and then, we implement a multi-platform system comprised of nodes interconnected in 1D torus topology. Initially, in single node configuration, the PowerPC is responsible to monitor and adjust the performance under pre-defined power and temperature constraints considering only the local sensors and pre-computed lookup tables which store characterization values. NoC neighbors are disregarded at the moment.

5.6.1 Power-aware Computing on a Single Multicore Node

Various kernels are selected from different application domains, extracted from the MiBench [231] and ALPBench [232] suites. The message-digest (MD5) hash operates on a set of predefined strings of various lengths. Inverse discrete cosine transform (IDCT) is a well known processing step in image decoding; the kernel processes blocks of 8×8 DCT matrices. We also use a 16- bit PCM to 4- bit coder kernel for adaptive differential pulse code modulation (ADPCM), a digital compression technique used mainly for speech compression in telecommunications; the algorithm works on blocks of sixteen integer raw samples at a time. Sobel edge detection algorithm performs partition of the pixels into edge and non-edge ones. Finally, the least squares calculation kernel, which operates on predefined numerical data as well, is used to compute estimations of parameters and to fit data minimizing the sum of squared residuals.

To compare the reliability of our emulation platform, measurement results are collected using both (a) simulation results and (b) actual sensor measurements. In the first case power estimation results are collected from Xilinx XPower. The back-annotated gate-level netlist is simulated to extract full system switchings in VCD format and import them to XPower. The junction temperature is 57 degrees (C^0) and the maximum ambient is 78 degrees. In (b) the embedded PowerPC acquires actual measurements from the prototyped SoC with a power/current sensor, the chip INA219 from Texas Instruments (TI) which is initially calibrated appropriately. PowerPC communicates with the

external sensor via an I^2C bus which connects to its private PLB bus and does not interfere with the bus of the coprocessors. Each processor executes the specific kernel from its private RAM. Figure 5.4 shows this comparison plot for the various application kernels. Simulation results are more pessimistic giving from 0.048 up to 0.059 difference in W, or an average 21.8% deviation from the actual values.

The simulation process to run the back-annotated gate-level netlist and extract the nets activity required more than 6 hours for simulating only 100μs on an Intel 2.4 GHz E6600 CPU using Modelsim. XPower required only a few minutes to load the netlist and VCD activity files. The speedup compared to our emulation platform varies for different application kernels, however the gains are enormous.

Figure 5.17 depicts the power and thermal behavior of a single node when the PPC manages in real-time the accelerators which perform MD5 calculations. The goal is to determine the accomplished performance and the corresponding peak power/temperature profile. The developed environment presented in 5.6.5 helps in studying this trade-off.

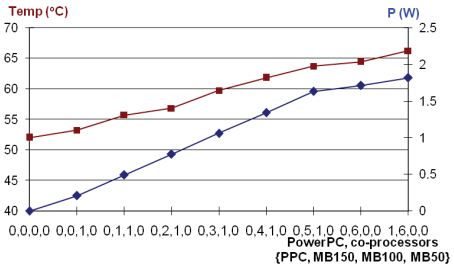

Figure 5.17: Power and temperature measurements using actual platform sensors while different number of cores are activated: MicroBlaze at either 150, 100 or 50 MHz and PowerPC at 300 MHz. All processors execute computation intensive MD5 hash computations.

The following scenario targets the exploitation of co-processors for data-intensive workloads

where bulk data transfers are needed. In this case we use the shared RAM between the PowerPC and co-processors. Figure 5.18 shows measurements of performance and power dissipation when a single node island is employed to execute parallel matrix-matrix integer multiplication that we developed. Typically, PowerPC initializes matrices A and B and afterward computes serially product C according to the formula: $C(i,j) = C(i,j) + \sum_{k=0}^{N-1} A(i,k) * B(k,j)$, where $0 \leq i \leq 17$ is the row index and $0 \leq j \leq 17$ is the column index, and N is equal to 20. All matrices reside in the shared BRAM. In our experiments, PPC assigns a three-row workload to each co-processor to be computed in parallel. Thus, a single complete processing of the allocated load is called one iteration as shown in figure 5.18. The power throttling thread is relaxed by raising the thresholds, to identify the node's maximum performance per watt for this kernel. PPC initially allocates memory space in the shared BRAM, then allocates the load per core, and then PPC controls the processing through commands via the FSL links. First, sets the maximum frequency (150 MHz) for all MicroBlaze co-processors, then, starts the computations, waits for a fixed period of time, and finally collects the number of iterations and puts the co-processors back to sleep.

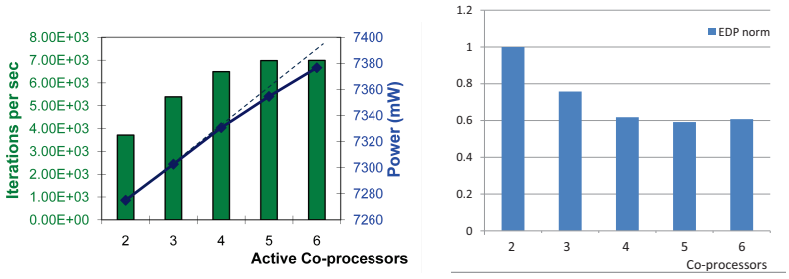

Figure 5.18: Performance and power measurements using shared BRAM to store source and result matrices for parallel matrix multiplication (C=A×B) when scaling the number of active coprocessors.

As figure 5.18 shows, communication bottlenecks due to the high ratio of memory accesses over computations and due to bandwidth limits of the bus, appear when more than three coprocessors are

triggered, with an adverse effect to performance gains. Note also the side-effect of communication collisions is that power consumption does not scale linearly with the number of activated co-processors. Actually, power reduces compared to a linear increase (depicted by the dashed line) as more cores are triggered, since communication cost is less, compared to computation cost in terms of energy. Figure 5.18 also displays the ED product metric, where E is the energy consumed while running the computations and D is the time to complete them. We preferred this metric against ED^2P as suggested by Brooks et al. [233] which cancels out the influence of frequency scaling because E is proportional to the square of frequency, whereas the D^2 is proportional to the inverse of the square of the frequency.

Figure 5.19 shows the instant power consumption of the node when PPC performs matrix multiplication using two co-processors. The abrupt change of power shown in figure is due to simultaneous activation of co-processors, OPB bus and related logic. To minimize di/dt noise caused

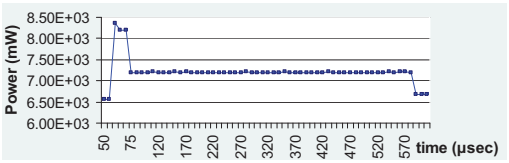

Figure 5.19: Instant power consumption when processing product matrix C using two-coprocessors.

by gating large processing units at the same time, actual ASICs should contain fine-grain gating domains. Additionally, sudden power surges may be undetected when mixed workloads are processed, or when system simulators are utilized to examine applications energy profile in a limited time window.

5.6.2 System Programming Model

This section discusses the programming model which is still evolving to enable efficient hardware utilization. Each accelerator uses its own private memory space (and private local memories) with explicit synchronization messages through the shared memory. Communication can also be interrupt-based. The programmer partitions and allocates private or shared memory space by modifying the linker script. Therefore, data can be pinned to either one of the memories inside a node, or can be transferred across nodes using messages through explicit function calls provided by the Aurora driver. Tasks or critical code segments have to be identified explicitly by the programmer, but the decision which accelerator is active can be done both at compile and at run-time. Currently, scheduling details are not hided from the user, thus allowing to explicitly control allocation, communication and regulation of the accelerators through the scheduling threads of the PowerPC; this is done at the expense of increasing the software development cost. The overall performance and power consumption depends on finding the optimal matching between multiple hardware resources and software.

Figure 5.20 shows an abstract layering of the software model that the developed platform supports. The system power and temperature protocol layer (SPTPL) relies first on the sensor management threads, the intra-node communication messages (using interrupts) to enable DFS, and the system-level control messages through the NoC, and second, on the optional prediction algorithms that are analyzed next in section 5.6.3. This SPTPL layer abstracts the access and control mechanisms to system sensors and to processing units.

The accelerator abstraction layer (AAL) is responsible to exploit the multiple accelerators through scheduling and synchronization control and at the same time interact with the bottom SPTPL layer to accomplish power and thermal- aware computing. The functions forming the SPTPL level are accessible from the user application in order to avoid additional cost and performances overheads induced by a standard layer approach. An application can be scheduled solely as a thread on the PPC, while different tasks run on the accelerators. The interaction between tasks is performed through communication primitives with different semantics allowing blocking and non-blocking calls. It is

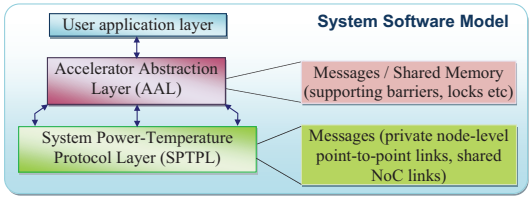

Figure 5.20: Programming model supported by the developed power-aware MPSoC; the AAL includes scheduling of sub-tasks to accelerators and synchronization primitives, while the lower protocol layer involves power- and thermal- aware primitives exploited by the upper layers.

important to note that hardware mutexes are not implemented due to limited area in the device.

5.6.3 Prediction-based Power and Temperature Management

This section explores the integration of efficient prediction techniques to perform power-aware allocation of resources, i.e., activation of the accelerator processors on the emulation platform. We utilize prediction algorithms in two contexts in order to dynamically manage resources in the face of changing workloads. First, we conduct dynamic thermal management by predicting temperature fluctuations and adjusting the performance of co-processors accordingly. Next, we utilize an efficient predictor to control workload spikes which may last in the order from micro- to milliseconds causing node heating up. By predicting one can prevent the overheating when still operating at lower temperatures and distribute the thermal effects more evenly across the cores. Early point predictions also lowers the impact of exponential dependency of leakage power on temperature. Prediction policies can be combined with methodologies proposed by researchers that utilize also performance monitoring units [234] or smart activity capture circuits [235] to detect the nature of workloads. One direction to workload characterization and monitoring is based on the ratio of the on-chip computation time to the off-chip access time, while other algorithms define CPU-boundedness as the fraction of program workload that is CPU-bound.

Alternatively, one can utilize thermal simulators to solve a set of differential equations using numerical methods. However, to determine the optimal frequency of operation through simulating thermal effects dynamically is expensive. When developing complex applications mapped on a NoC-based multicore system it is usually hard to simulate ahead various data distributions and loads at the design phase. Even by using good statistical functional modeling and traffic generators dynamic behavior of particular applications (e.g., video processing) is difficult to capture. To simplify control one can use a fixed time period and attempt to adjust voltage/frequency at the end of the interval. However, this entails the risk to miss opportunities to respond to large activity swings inside the interval.

We employed the developed emulation platform to study effects of applications' dynamic behavior. We used two generators in our experiments, first to provide different random amounts of load to the multicore node and second, to generate uniformly random idle intervals between consecutive loads. The generator threads are performing on the neighbor PPC node and provide the inputs via the Aurora high-speed links.

For comparison we developed two prediction mechanisms which are based on mathematical models. First, we developed in software an auto-regressive moving average (ARMA) model for estimating future temperature; this model uses sequential probability ratio test (SPRT), to detect the drift quickly [73]. The ARMA model which is utilized for temperature forecasting is derived based on temperature traces captured from actual thermal sensors. It represents the thermal characteristics of the current workload when processed by the emulation platform. Since the workload dynamics might change the ARMA model is adapted at runtime when it is detected to deviate more than 5%. SPRT performs statistical hypothesis tests on the mean and variance of the residuals. The user specifies false- and missed-alarm probabilities for the detection process in order to control the likelihood of the missed detection of residual drifts. Alternatively, to reduce computation cost, the user can define a simple threshold value for detection (e.g., setting a threshold for the standard deviation of the prediction error). Second, we employed the control theory approach utilizing recursive least squares which has been proposed [214] for temperature prediction. The least squares regression technique

(LSQRT) for quadratic curve fitting [236] maintains a history in the form of a cyclic buffer of ten entries, which however requires significant computation effort.

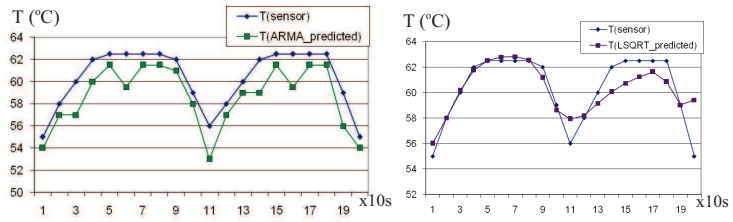

Figure 5.21: Temperature prediction using ARMA modeling and least squares regression (time scaled to plot)

Figure 5.21 shows the accuracy of each technique during a simulation slice in which the temperature exhibits large variations. Even though the temperature differences of the ARMA model are more abrupt the computation cost is less compared to the LSQRT algorithm. Note that due to large computation overhead the time scale of least squares technique is normalized. Each prediction using the least squares method requires almost 15.8 sec when executing at 100 MHz.

In general, false positives or negatives may occur when employing one prediction algorithm, which needs to be fitted to the particular application driven by the user requirements for performance, reliability and resources. The speed of read-out circuitry to acquire temperature values, accuracy, sampling rate, interrupt latency, depth of prediction history, optimizations of the prediction algorithm, thread scheduling policy, priorities, and dynamic workload attributes affect any prediction method. The effect of each decision based on the predictor depends on the amount of speed change, on the amount of workload that will be processed and on the number of cores that will affect. In this use case the cost of employing an instance of computation intensive predictors into each processor of a multicore SoC is increased as large parts of useful processor time are spent on prediction algorithm, while consuming additional energy as well. Thus, lightweight monitors are

essential so that power characteristics of various applications can be extracted quickly and linked with local thermal profiles which are maintained as power/thermal lookup tables in software. Then, based on these lookup tables the allocated workload is balanced at runtime. Building and overall management of lookup tables is left to software, since it depends on the actual application behavior. A task with very irregular activity and non-deterministic behavior obviously requires more frequent activation of the monitors and table adaptations; detailed exploration of such diverse behavior will be performed in future work.

Given an analytic thermal model, prediction schemes are considered in the literature [214], even proposals of a comprehensive thermal model accounting for cooling or leakage current [237]. Similarly, we additionally evaluate a simple temperature model, where we assume T_{ss} is the steady state temperature of an application. Let T(t) represent the temperature at time t and let T_0 be the temperature when an application starts running on the PPC. Then, the temperature at any time t is formulated as:

$$T(t) = T_{ss} - (T_{ss} - T_0) * e^{-t/c_s} \tag{5.1}$$

where c_s is a system-specific constant. By rearranging and simplifying equation 5.1 the expression of the temperature becomes:

$$T(t) = c_1 * e^{-t/c_s} + c_2 \tag{5.2}$$

To calculate the constants we conducted experiments with a single node executing the MD5 hash algorithm. At infinity equation 5.2 gives $T(\infty)=c_2$, while the initial temperature at t=0 results in $T(0)=c_1+c_2$. Measurements from the thermal sensor on the prototype resulted in corresponding values for an operating temperature of the chip which ranged from 52 to 64 degrees Celsius. The analytic-based prediction model we used exploits the slope of the derivative of the temperature from equation 5.2, which gives:

$$\frac{dT}{dt} = -c_1/c_s * e^{-t/c_s} \tag{5.3}$$

Equation 5.3 expresses the tendency of temperature to rise. The goal is to invoke DFS when the predictor detects the slope to rise above a predefined threshold. With the given thermal state of the

system and the computed derivative dT/dt at time t_i the prediction thread can identify whether the system reaches its steady state. Alternatively, we developed the predictor to trigger a frequency decrease through comparing the derivative computed at time t_{i-1} and t_i. In order to reduce computation complexity we coded the exponential function e^x as defined by the following power series $e^x = \sum_0^\infty \frac{x^n}{n!}$ [238]. Figure 5.22 shows the accuracy we can achieve by utilizing four to six terms to achieve better approximation while tracking the temperature slope. However, still the computation overhead is significant for this analytic-based thermal prediction scheme.

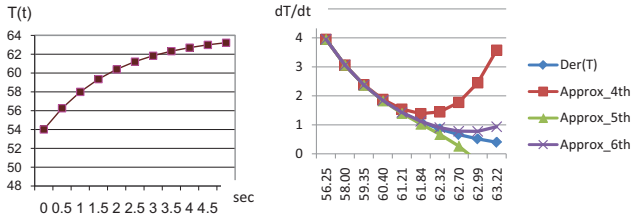

Figure 5.22: Temperature measurements for MD5 kernel and computed approximation of temperature derivative (Der(T)) on a single PPC node

Table 5.3 compares the proactive techniques overhead with respect to execution time for the benchmark applications. Note that predictor threads are running only on the PPC and are prioritized against the computation threads. Acceleration through the MicroBlaze cores is disabled for this experiment.

Next, we selected the most efficient prediction scheme in a different context. Figure 5.23 shows the efficiency of forecasting application's behavior using the ARMA predictor; the aim is to directly detect fluctuations of the incoming workload. The captured samples are used to compute the residuals which represent the number of iterations achieved for the processed workload: $R(t) = I_i(t) - I_i^p(t)$, where $I_i(t)$ is the number of iterations reported by the coprocessor and $I_i^p(t)$ represents the predicted value; one iteration represents the computation of MD5 hash on a block

Predictor	ARMA	LSQRT	Analytic (6 terms)
MD5	17%	53%	48.6%
ADPCM	16.5%	51%	43%
Sobel	23%	64%	52%
IDCT	19.5%	56%	49%

Table 5.3: Overhead of proactive techniques

of five variable-length strings for a total of 190-characters. The computations cost from 150ms up to 400ms when the residuals have a drift from the training data and we need to adjust the ARMA model. The obtained predictions are very efficient unless extreme fluctuations of more than 100 per-

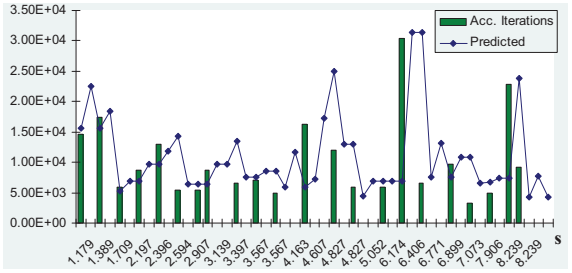

Figure 5.23: ARMA-based prediction of dynamic workload; prediction thread tracks workload assigned to single MicroBlaze running at 150 MHz. Number of iterations refers to MD5 hash computations on a block of five strings.

cent are present. The predictor currently does not account for non-memory stalls and uses constant memory latencies. These shortcomings are acceptable in the context of the developed environment of the coprocessors. Isci et al. [239] use global history tables to predict the phases of an application running on a single processor if these tend to be memory- or computation- bound, based on performance monitoring counters. Associativity searching using more than a hundred entries is proposed while we show that more intelligent predictors with shallow tables can also achieve satisfactory

results.

In conclusion, proactive dynamic management mechanisms are predominantly a trade-off between limiting power and hitting performance. DPM and DTM techniques do invariably cause some performance degradation. It must be noted that inconsiderate thread scheduling causing frequent migrations can make the temperature dynamics to vary with an impact to prediction accuracy. We believe that dynamic accelerator-based computing offers more efficient environment for various power and thermal prediction policies. Additionally, invoking the training phase very often, besides requiring several CPU cycles, is expected to increase the system power consumption and the temperature in turn. Moreover, prediction algorithms that utilize distributed current and thermal sensors can capture energy effects of various system components such as main memory, or energy consumption due to leakage. Different dynamic management methods disregard the correlation between frequency and energy consumption, assuming that lowering frequency is always beneficial to energy. However, this is not true when leakage power consumption is significant, or the overall optimization goal is to minimize system energy consumption instead of processor power consumption.

5.6.4 Enhancing Monitors with Hardware Predictors

While software-based pro-active management delivers promising results, it would be more efficient to off-load the processor, especially when considering multiple distributed tasks to perform local predictive management and decision making. Hence, ideally full, or partial computing effort of a pro-active algorithm can be shifted to customized hardware units. On the basis of the summary results given in Table 5.3 the less compute-intensive pro-active algorithm is ARMA predictor, which gives a hint for an effective hardware implementation. Figure 5.24 shows design exploration results for various implementation options of the sequential probability ratio test (SPRT) kernel computation of the ARMA model using the tool Vivado provided by Xilinx. Two main configurations are explored using float or double accuracy.

The implemented hardware predictor unit operates with a clock cycle that ranges from 3.72

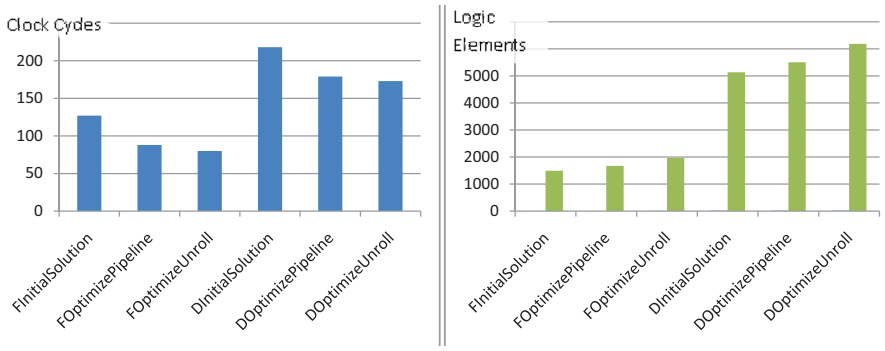

Figure 5.24: Design exploration of SPRT kernel in hardware in terms of latency and area on a xc7v2000tflg1925 device

to 4.21 ns for the particular device. Thus, as shown in figure 5.24 a predictor unit using floating point accuracy can sustain a throughput of one result every almost 80 clock cycles. The cost to use double accuracy is too high since the resulting complexity exceeds the area of a MicroBlaze soft-processor. Alternatively, more effective optimization can be achieved through using longer history. By employing pre-processing of the overall management scheme in hardware and close to the sensor readings many modular monitoring units can be integrated in a SoC freeing computing resources.

5.6.5 System-level Power and Thermal Management

In this section, we present a multi-island platform to implement system-wide policies which aim to maximize performance under power and thermal constraints. The system considers these restrictions to manage the varying pool of resources in a distributed manner. The goal is to evaluate system response and precise control of both individual and system constraints, while configurations and conditions may change more rapidly than software can adapt. At the system level, we built a 1-dimensional torus or equally a k-node ring which is depicted in figure 5.25 (k=4 for this experiment). This organization presents a scaled-down version of the concept of the multi-island emulation NoC platform, as envisioned in section 5.4. Each node integrates two Aurora serial links

to connect to the neighbors via the SATA interfaces. This is mostly due to the limited FPGA device resources and since the remaining on-chip RocketIO transceivers can be connected only to optical interfaces. In any case, even though a 2D mesh topology is commonly assumed in NoCs, implementing packet-switched architectures can result in high cost and complexity. For example, in the 2D mesh network in Intel Tera-flops chip, the router consumes approximately 25% of the total area and 28% of the power [240]. Recent research [241] proves that lower per-hop router latency and no serialization latency results in lower latency for the ring topology compared to a 2D mesh topology, although the ring topology has larger hop count with larger network diameter. Moreover, ring topologies have been recently used in modern small-scale, multicore processors including the IBM Cell processor [227] and the Intel Sandy Bridge processor [242]. The high-speed Aurora links perform light-weight routing and transfer control information, i.e. power and temperature indications and commands to initiate or throttle processing. For simplicity, data transfers are performed using fixed-size frames of eight flits. The communication threads and ISRs are extended in order to include message segmentation and re-assembly functions and thus enable transfers of large chunks of data among nodes or support task migration.

A software interface is developed to run on a monitor host which can be attached to any single node of the platform through the *UARTlite* hardware peripheral. At the identification phase this monitor application sends an ”inquiry” frame to the attached platform; this frame is relayed to all nodes in order for the application to determine the number of nodes and automatically present the corresponding interface to the user. If a monitoring host is detected to be attached to the system, then, one thread is spawn to run on the PowerPC of the attached node in order to communicate statistics and commands. Typically, in an autonomous embedded system this monitoring host stands for a system monitoring thread, responsible to set power constraints for all the nodes of the NoC with respect to user preferences or application requirements and runtime conditions. In manual mode, as shown in the top section of figure 5.26, each co-processor of an island can be controlled individually in terms of frequency. Thus, we can evaluate performance, power and thermal effects for an application kernel under specific constraints. On the other hand, through the user interface

one can set power and temperature constraints and rely on the dynamic mechanism inside each node which is invoked to determine automatically the number of active co-processors and their operating frequency. Afterward, the system operates in a closed form opting for maximum performance within the specified power budget in a distributed fashion.

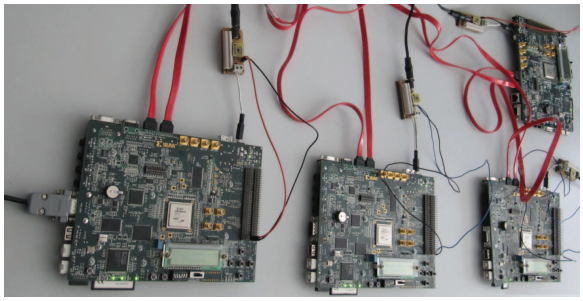

Figure 5.25: A four-node emulation prototype interconnected through Aurora interfaces at 1.5 Gbps using SATA cables. Each node is mapped on a board which integrates two temperature sensors on-board; an additional custom current sensor (TI's INA219) is adapted to the power supply of each board.

Power and temperature status updates of each node are broadcasted to all PowerPCs. The algorithm 1 in section 5.5 is modified to take into account neighbor activity by allowing each PPC to reduce performance to 5% for each neighbor that exceeds pre-configured power thresholds. In general, factors such as proximity of cores and physical topology (caches or processing elements at the center or periphery of each island) are determining the degree of impact of neighboring nodes and must be carefully define the coefficients or percentage by which to throttle performance.

The screenshot of figure 5.26 depicts a scenario where all nodes execute MD5 hash computations of a block of five variable-length strings for a total of 190-characters; the system automatically discovers three nodes. At the initialization phase a communication thread is spawn when one PPC detects the monitor host. This PPC commands each island to do a re-calibration of the current

sensors. The total number of iterations, along with the power and temperature status, are reported both at the user interface and on the LCD displays locally. The only packets exchanged between the nodes concern control information; data for processing are maintained locally for the given experiment. In any case, the burden carried by PPC involves handling communication functions in software (i.e., routing, forwarding) together with power management and computation threads. Aurora interfaces provide more throughput than PPC can handle.

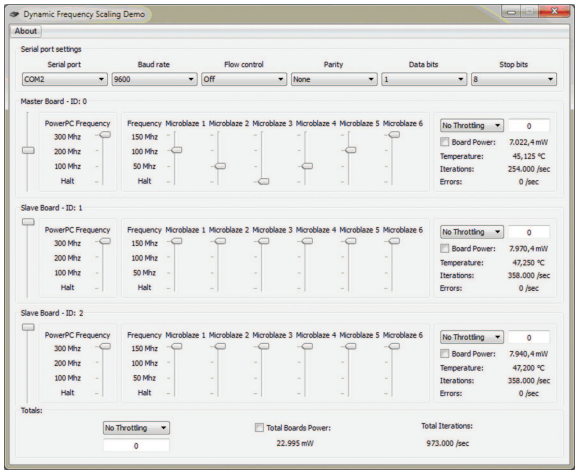

Figure 5.26: Management software interface through a host monitor for system-wide evaluation of power-aware processing.

Figure 5.27 shows the system power consumption where the host monitor controls the power thresholds of nodes one and three forcing them to increase their performance. Node two is also configured to specific power threshold and continuously tracks the power updates from the neighbors. The goal is to achieve the maximum performance of the co-processors without exceeding the total power budget of the system. Upon abrupt changes, PPC of island two adjusts the local power threshold at runtime, by reducing frequency or shutting down its coprocessors. During the time interval Δt_a island one activates two more coprocessors, in interval Δt_b only island three causes the power

consumption to increase, while after, in interval Δt_c both neighbors cause the island two to reduce its performance, and along to reduce the dissipated power by almost 60 mW. The response time of each node is negligible since the overhead to serve an incoming management flit from an Aurora link is measured to be less than a hundred clock cycles. By identifying which architectural resources and scheduling policies are inherently more power hungry we can do a better job in designing power efficient microarchitectures.

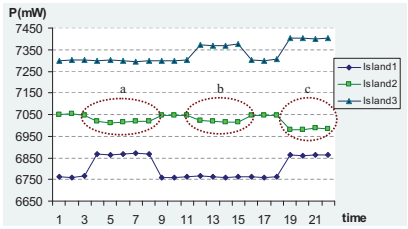

Figure 5.27: System-wide power management: impact of power transients to island2 (time is normalized 100μs).

In the last study, we differentiate the tasks executing on an island to maximize their performance for performance-critical tasks and "best-effort" tasks, which are allocated on specific islands to ensure safe thermal operation of the system. Last, we enhance tasks to be thermal-conscious as they attempt to optimize their performance in a weighted manner and at the same time respect the system's constraints. Each PPC divides the time into epochs. At the end of each epoch, set to 1ms, the processor decides to perform DFS according to performance requirements and system temperature as communicated from the neighbors. To avoid false triggering the DFS decision mechanism delays for a time interval of two epochs. Figure 5.28 illustrates the behavior of a four-island platform as the island two attempts to get the maximum performance while islands one and three execute "best-effort" tasks. Alternatively, we can call these tasks as "thermal-sensitive", since in the experiment it is assumed that performance loss for these tasks is affordable.

Islands zero and two perform MD5 hash computations, and one and three execute the ADPCM

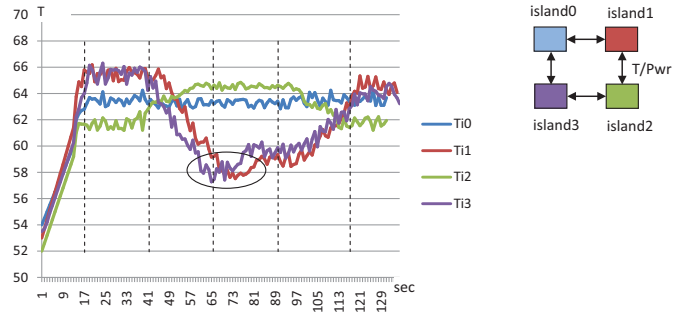

Figure 5.28: System-wide thermal management utilizing a four island heterogeneous system

kernel. The system starts operation with a moderate configuration: each node activates its accelerator coprocessors at 100 MHz. The island two is configured to increase the calculation throughput from six accelerators operating at 100 MHz to the maximum at 150 MHz. Islands one and three are set to operate as "best-effort" and immediately turn off all accelerators, with a noticeable drop to temperature since the ADPCM kernel exhibits significant power consumption requirements. After the deep "dive" which is marked in the plot, the two islands activate two accelerators at 100MHz, while examining at the same time the local and adjacent temperatures. Given the notifications for the power increase that arrive at island zero from a remote neighbor the PPC at island zero decreases the frequency of just two accelerators with a minimal impact to its performance, almost 5%, and to its local temperature. Note that in this experiment the PPC does not participate in workload processing. This allows the co-processors to actually be controlled in system-level, as a pool of 24 accelerators. Such configurations, although static, provide useful insight about the interactions of balancing hot and cold kernels. On top, similar cases allow to evaluate the thermal sensitivity of each task, which can be defined as $\Delta T/\Delta R$, the gain in temperature (positive or negative) over the difference of task's throughput rate R. Thus, the characterization and the intermixing of tasks with different sensitivity is a challenge for the developer of thermal-aware multicore SoCs.

Utilizing the random generator for load fluctuations that was used in the experiment shown in figure 5.23, we studied the distribution of hot-spots when the ARMA predictor is employed. The platform shown in figure 5.28 is used. The nodes with MD5 kernels appeared temporal hot spots which were reduced from 23 to 17, while nodes running ADPCM presented a reduction from 29 to 22, a 24% reduction. More spectacular reduction can be achieved for less aggressive workload spikes and with an optimized predictor history table.

In particular, the understanding of the tasks' attributes with respect to energy needs and the impact to the thermal dynamics of each node can guide the mapping of prediction algorithms to achieve an efficient distribution. Thus, prediction schemes that gather many sampling points within the most-recent time interval to build a precise prediction model can be relaxed, or even disabled, for the nodes whose load and neighbor's interference remain nearly constant. This permits significant reduction in energy consumption at the system level, due to savings of the computation overheads of many instances of the prediction algorithm.

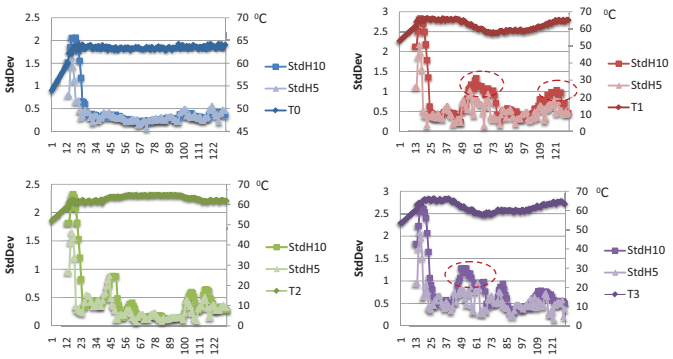

Figure 5.29: History-based threshold to enable proactive management through standard deviation function for the four-node island system

Figure 5.29 shows the usage of a history-based threshold function to determine when proactive management is beneficial to be activated. We added this threshold function to the management

software running on the PPCs inside the four-node island platform. The temperature variance of each node is estimated from a number of samples drawn from a short history vector. We use only five samples for a very light-cost function (StdH5) and a second version of variance calculation with ten samples for higher accuracy (StdH10). The complexity of such a threshold function should be minimal to reduce the performance and energy cost. Besides the initial warming-up phase, as figure 5.29 illustrates, there are various points for nodes one and three, highlighted with dotted circles, that present considerable variance. In these cases, a prediction algorithm can be engaged to monitor node's thermal levels. The threshold is set to value "1"; this is a rough estimate, indicating how aggressively we can adjust thresholds and dynamic proactive management. Note that the threshold function must consider instant application behavior to protect the system from abrupt changes.

Future directions of our research consist of extending the results to more accurate power modeling and evaluating the proximity of existing strategies to the optimum. Utilization of local memories as well, or on-chip instantiation of various communication schemes, are topics of their own. Additionally, it is important to minimize intrusiveness of power and thermal monitoring mechanisms along with tailoring of prediction schemes to the degree of variability of each application domain. Introduction of power states for interconnection links is yet another important field for study.

With the emergence of NoC-based multicore architectures in an array of application domains it is important to explore different power- and thermal- conscious microarchitectural and system level policies in advance.

Intelligence is a desired property for the hardware cores as a countermeasure to their inherently reduced flexibility. Towards this goal, one approach is to redesign all cores from scratch with self-adaptive properties in mind, which would be very costly and time-consuming. Instead, it is preferable to reuse existing intellectual property (IP) libraries and extend these with self-adaptive concepts. For this, it is necessary to monitor the behavior of the underlying component, and to be able to effect changes or actions in the component's operating parameters. The utilization of hardware monitoring support opens new directions towards introducing intelligence to system components, a trend that will become dominant in the near future in the view of emerging heterogeneous

multicore embedded systems.

5.7 Conclusions

A hardware monitoring infrastructure is introduced to address the run-time management and resource adaptation of multi-core SoCs in an non-invasive way, thus performing efficient on-line adjustments and optimizations. Special awareness can be achieved by integrating intelligent run-time monitoring agents which allow for dynamic optimizations that are hard to reach using traditional design space exploration methods. By providing programmable monitoring support and hooks and allowing the management-control system to make certain decisions at run time, the designer no longer has to consider and predict all aspects of system behavior. Not only does this alleviate the burden on the designer, but when performing decisions at run-time, much more information about the actual application being executed and the system's operating environment are available. Similar to examples from the HPC domain, such as IBM Blue Gene [243] or Tilera [15] that utilize separate communication infrastructures for synchronization, separate monitoring links become also of major importance for monitoring critical information and changes in multicore embedded systems. This paper has demonstrated that significant awareness for improvements of performance and resource utilization can be achieved by including specific hardware extensions in multicore embedded SoCs. This work does not deliver a set of optimum hardware components to this big challenge. However, it attempts to provide parametric, configurable at design time and programmable at run-time monitoring units as generic hardware building blocks for improving embedded systems operation.

Chapter 6

Conclusions

THIS DISSERTATION has explored on-chip monitoring in the light of assisting for efficient and energy-conscious system operation. The traditional concern of designers on maximizing performance and minimizing power becomes more challenging as new solutions have to be envisioned to offer a customisable and adaptable computing capability to efficiently respond to the on-demand computing quest that modern complex SoCs and applications pose. A growing proliferation of multi- and many-core chips increasingly embrace new dynamic techniques to offer a correct balance between energy consumption and attaining optimal performance. In this regard this work described a hardware monitoring infrastructure to address the run-time management and resource adaptation of multi-core SoCs in an non-invasive way in order to perform efficient on-line adjustments and optimizations.

Hardware performance counters were originally introduced as a mechanism for computer architects to aid in post-silicon validation (i.e., to debug and verify processor behavior after fabrication). After, it was shown that using these counters was useful for performance profiling as well. With feedback from the hardware, application programmers and compilers could target sources of inefficiency more accurately. Unfortunately, current support for hardware-assisted monitoring, which is mainly centered around hardware performance counters, debug registers and a couple of bits in various memory data structures, is lacking in several aspects. Shortcomings of existing hardware

monitoring include high overhead, lack of flexibility, inability to precisely monitor certain events (e.g., multiple concurrent memory instructions in the pipeline) and lack of standardization among multiple processor versions or families and in system-level aspects in the light of emerging NoC-based multi-core SoCs.

This dissertation addressed the need to define light-weight hardware monitoring primitives which can be organized in a distributed fashion to facilitate measuring and monitoring metrics of intereset in NoC-based systems. This monitoring infrastructure mainly provides counter-based agents with configurable and programmable filtering and an interface to provide local or remote services.

It is demonstrated that using these designed monitoring primitives multi-core Systems-on-Chip can be enhanced with the capacity to dynamically detect and predict the power requirements for different applications. Moreover, we presented a distributed algorithm to diffuse power changes smoothly to neighboring cores. The evaluation of the presented infrastructure shows that inaccuracies of off-line data profiling or environmental or process variation effects of multi-core chips, can be easily overcome with the aid of hardware monitors and corresponding reaction mechanisms.

Finally, we developed a monitoring-aware platform that provides centralized and distributed mechanisms, along with prediction algorithms, DFS and workload throttling, to support dynamic runtime control and adaptation of power consumption both at local and at system level. These are exploited to enhance designing of energy- and thermal-aware embedded SoCs. Actual current and temperature sensors per node are sampled to ensure that the NoC-based MPSoC operates within a safe operating range and at the optimal performance level. The system considers power and thermal restrictions to manage the varying pool of resources in a distributed manner. Monitoring is finally examined in terms of feeding various prediction policies to avoid localized or system-wide emergencies. Contributions proposed in this thesis can be extended in various directions, from low-level design of low-power monitoring circuitry to system-level monitoring-conscious policies. Despite the potentially detailed feedback available by hardware performance counters, precisely mapping this information to higher level entities is still a manual and ad hoc process.

The development of on-chip monitoring subsystem and corresponding system-level services

involves a number of trade-offs from architectural point of view, including communication protocols and software interfacing, interaction and interoperability as well. Through combining different methodologies monitoring the system behavior is a valuable vehicle to capture the changes in the behavior of the system and enable mechanisms to adapt to these changes and offer improved performance and energy efficiency.

Bibliography

[1] A. Mahmood and E. J. McCluskey, "Concurrent Error Detection Using Watchdog Processors-A Survey," *IEEE Trans. Comput.*, vol. 37, no. 2, pp. 160–174, Feb. 1988.

[2] J. Ohlsson and M. Rimen, "Implicit Signature Checking," in *Proceedings of the Twenty-Fifth International Symposium on Fault-Tolerant Computing*, 1995, FTCS '95, pp. 218–.

[3] ," The international technology roadmap for semiconductors, 2009, url: www.itrs.net.

[4] William J. Dally and Brian Towles, "Route packets, not wires: on-chip inteconnection networks," in *DAC '01: Proceedings of the 38th annual Design Automation Conference*, 2001, pp. 684–689.

[5] R. Ho, K. Mai, and M. Horowitz, "The Future of Wires," *Proceedings of the IEEE*, vol. 89, no. 4, pp. 490–504, 2001.

[6] Nir Magen, Avinoam Kolodny, Uri Weiser, and Nachum Shamir, "Interconnect-power dissipation in a microprocessor," in *SLIP '04: Proceedings of the 2004 International Workshop on System Level Interconnect Prediction*, 2004, pp. 7–13.

[7] H. Peter Hofstee, "Power Efficient Processor Architecture and The Cell Processor," in *HPCA '05: Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, 2005, pp. 258–262.

[8] Poonacha Kongetira, "A 32-Way Multithreaded SPARC Processor," in *Proceedings of the 16th Symposium on High Performance Chips*, 2004.

[9] Mark D. Hill and Michael R. Marty, "Amdahl's Law in the Multicore Era," *Computer*, vol. 41, no. 7, pp. 33–38, July 2008.

[10] Santanu Dutta, Rune Jensen, and Alf Rieckmann, "Viper: A Multiprocessor SOC for Advanced Set-Top Box and Digital TV Systems," *IEEE Des. Test*, vol. 18, no. 5, pp. 21–31, Sept. 2001.

[11] O. Semenov, A. Vassighi, and M. Sachdev, "Impact of self-heating effect on long-term reliability and performance degradation in CMOS circuits," *IEEE Trans. Devices Mater.*, vol. 6, no. 1, pp. 17–27, Mar 2006.

[12] Karl G. Heider, "The Rashomon Effect: When Ethnographers Disagree," *American Anthropologist*, vol. 90, no. 1, pp. 73–81, 1988.

[13] Michael J. Flynn and Patrick Hung, "Microprocessor Design Issues: Thoughts on the Road Ahead," *IEEE Micro*, vol. 25, pp. 16–31, May 2005.

[14] William J. Dally and Brian Towles, "Route packets, not wires: on-chip inteconnection networks," in *Proceedings of the 38th annual Design Automation Conference*, New York, NY, USA, 2001, DAC '01, pp. 684–689, ACM.

[15] Tilera Corporation. The TILE64 chip., ," 2009, www.tilera.com.

[16] Intel TeraFLOPS, ," 2010, http://www.intel.com/content/www/us/en/research/intel-labs-teraflops-research-chip.html.

[17] Nvidia Tegra 3, ," 2012, www.nvidia.com/object/tegra-superchip.html.

[18] Michael Floyd, Malcolm Allen-Ware, Karthick Rajamani, Bishop Brock, Charles Lefurgy, Alan J. Drake, Lorena Pesantez, Tilman Gloekler, Jose A. Tierno, Pradip Bose, and Alper

Buyuktosunoglu, "Introducing the Adaptive Energy Management Features of the Power7 Chip," *IEEE Micro*, vol. 31, pp. 60–75, 2011.

[19] Felix Salfner, Maren Lenk, and Miroslaw Malek, "A survey of online failure prediction methods," *ACM Comput. Surv.*, vol. 42, no. 3, pp. 10:1–10:42, Mar. 2010.

[20] Martin Radetzki, Chaochao Feng, Xueqian Zhao, and Axel Jantsch, "Methods for Fault Tolerance in Networks on Chip," *ACM Comput. Surv.*, pp. 1–35, Feb. 2012.

[21] Luca Benini, Alessandro Bogliolo, and Giovanni De Micheli, "A Survey of Design Techniques for System-Level Dynamic Power Management," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 8, no. 3, pp. 299–316, June 2000.

[22] Joonho Kong, Sung Woo Chung, and Kevin Skadron, "Recent thermal management techniques for microprocessors," *ACM Comput. Surv.*, vol. 44, no. 3, pp. 13:1–13:42, June 2012.

[23] Matthias Hauswirth, Peter F. Sweeney, Amer Diwan, and Michael Hind, "Vertical profiling: understanding the behavior of object-priented applications," *SIGPLAN Not.*, vol. 39, pp. 251–269, October 2004.

[24] Mani K. Chandy, "Event-Driven Applications: Costs, Benefits and Design Approaches," 2006.

[25] Line Pouchard, Steve Poole, Josh Lothian, and Chris Groer, "Open Standards for Sensor Information Processing," Tech. Rep. ORNL-TM-2009-145, Oak Ridge National Laboratory, 2009.

[26] R. Leatherman and N. Stollon, "An embedded debugging architecture for SoCs," *IEEE Potentials*, vol. 24, no. 1, pp. 12–16, Feb-Mar 2005.

[27] Bart Vermeulen and Kees Goossens, "Debugging Multi-Core Systems on Chip," in *Multi-Core Embedded Systems*, George Kornaros, Ed., Chapter 5, pp. 153–198. CRC Press, Taylor & Francis Group, Apr 2010.

[28] E.A. Daoud and N. Nicolici, "Embedded Debug Architecture for Bypassing Blocking Bugs During Post-Silicon Validation," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 19, no. 4, pp. 559 –570, april 2011.

[29] Xilinx Chipscope, ," [Online] Available at: www.xilinx.com/tools/cspro.htm.

[30] ARM Coresight, ," [Online] Available: www.arm.com/products/solutions/CoreSight.html.

[31] R. Leatherman, "On-Chip Instrumentation Approach to System-On-Chip Development," 1997.

[32] IEEE-ISTO, ," The Nexus 5001 Forum Standard for a Global Embedded Processor Debug Interface, http://www.nexus5001.org, Nov. 2009.

[33] Andrew B. T. Hopkins and Klaus D. McDonald-Maier, "Debug Support Strategy for Systems-on-Chips with Multiple Processor Cores," *IEEE Trans. Comput.*, vol. 55, no. 2, pp. 174–184, Feb. 2006.

[34] Miron Abramovici, Paul Bradley, Kumar Dwarakanath, Peter Levin, Gerard Memmi, and Dave Miller, "A reconfigurable design-for-debug infrastructure for SoCs," in *Proceedings of the 43rd annual Design Automation Conference*, New York, NY, USA, 2006, DAC '06, pp. 7–12, ACM.

[35] Flavio M. de Paula, Amir Nahir, Ziv Nevo, Avigail Orni, and Alan J. Hu, "TAB-BackSpace: unlimited-length trace buffers with zero additional on-chip overhead," in *Proceedings of the 48th Design Automation Conference*, New York, NY, USA, 2011, DAC '11, pp. 411–416, ACM.

[36] Sung-Boem Park and Subhasish Mitra, "Post-silicon bug localization for processors using IFRA," *Commun. ACM*, vol. 53, no. 2, pp. 106–113, Feb. 2010.

[37] Calin Ciordas, Twan Basten, Andrei Rădulescu, Kees Goossens, and Jef Van Meerbergen, "An event-based monitoring service for networks on chip," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 10, no. 4, pp. 702–723, Oct. 2005.

[38] Mohammed El Shobaki and Lennart Lindh, "A hardware and software monitor for high-level system-on-chip verification," in *ISQED '01: Proceedings of the Intl. Symp. on Quality Electronic Design*, 2001, pp. 56–61.

[39] Érika Cota, Luigi Carro, and Marcelo Lubaszewski, "Reusing an on-chip network for the test of core-based systems," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 9, no. 4, pp. 471–499, 2004.

[40] Min Xu, Rastislav Bodik, and Mark D. Hill, "A "flight data recorder" for enabling full-system multiprocessor deterministic replay," in *Proceedings of the 30th annual international symposium on Computer architecture*, New York, NY, USA, 2003, ISCA '03, pp. 122–135, ACM.

[41] Arkaprava Basu, Jayaram Bobba, and Mark D. Hill, "Karma: scalable deterministic record-replay," in *Proceedings of the international conference on Supercomputing*, New York, NY, USA, 2011, ICS '11, pp. 359–368, ACM.

[42] Derek R. Hower, Pablo Montesinos, Luis Ceze, Mark D. Hill, and Josep Torrellas, "Two hardware-based approaches for deterministic multiprocessor replay," *Commun. ACM*, vol. 52, pp. 93–100, June 2009.

[43] Pin Zhou, Radu Teodorescu, and Yuanyuan Zhou, "HARD: Hardware-Assisted Lockset-based Race Detection," in *Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture*, Washington, DC, USA, 2007, pp. 121–132, IEEE Computer Society.

[44] Chi-Neng Wen, Shu-Hsuan Chou, Chien-Chih Chen, and Tien-Fu Chen, "NUDA: A Non-Uniform Debugging Architecture and Nonintrusive Race Detection for Many-Core Systems," *Computers, IEEE Transactions on*, vol. 61, no. 2, pp. 199 –212, feb. 2012.

[45] Brinkley Sprunt, "The Basics of Performance-Monitoring Hardware," *IEEE Micro*, vol. 22, pp. 64–71, July 2002.

[46] John Demme and Simha Sethumadhavan, "Rapid identification of architectural bottlenecks via precise event counting," in *Proceedings of the 38th annual international symposium on Computer architecture*, New York, NY, USA, 2011, ISCA '11, pp. 353–364, ACM.

[47] Jeffrey Dean, James E. Hicks, Carl A. Waldspurger, William E. Weihl, and George Chrysos, "ProfileMe: hardware support for instruction-level profiling on out-of-order processors," in *Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture*, Washington, DC, USA, 1997, MICRO 30, pp. 292–302, IEEE Computer Society.

[48] Hyun-min Kyung, Gi-ho Park, Jong Wook Kwak, WooKyeong Jeong, Tae-Jin Kim, and Sung-Bae Park, "Performance monitor unit design for an AXI-based multi-core SoC platform," in *Proceedings of the 2007 ACM symposium on Applied computing*, New York, NY, USA, 2007, SAC '07, pp. 1565–1572, ACM.

[49] Valentina Salapura, Karthik Ganesan, Alan Gara, Michael Gschwind, James C. Sexton, and Robert E. Walkup, "Next-Generation Performance Counters: Towards Monitoring Over Thousand Concurrent Events," in *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software*, Washington, DC, USA, 2008, ISPASS '08, pp. 139–146, IEEE Computer Society.

[50] Intel ® Xeon ®, ," Processor E5-2600 Product Family Uncore Performance Monitoring Guide, www.intel.com, March 2012.

[51] Théodore Marescaux and Henk Corporaal, "Introducing the SuperGT network-on-chip: SuperGT QoS: more than just GT," in *Proceedings of the 44th annual Design Automation Conference*, New York, NY, USA, 2007, DAC '07, pp. 116–121, ACM.

[52] Daniele Mangano and Giovanni Strano, "Enabling dynamic and programmable QoS in SoCs," in *Proceedings of the Third International Workshop on Network on Chip Architectures*, New York, NY, USA, 2010, NoCArc '10, pp. 17–22, ACM.

[53] Akbar Sharifi, Hui Zhao, and Mahmut Kandemir, "Feedback control for providing QoS in NoC-based multicores," in *Proceedings of the Conference on Design, Automation and Test in Europe*. 2010, DATE '10, pp. 1384–1389, European Design and Automation Association.

[54] J. W. van den Brand, C. Ciordas, K. Goossens, and T. Basten, "Congestion-controlled best-effort communication for networks-on-chip," in *Proceedings of the conference on Design, automation and test in Europe*, San Jose, CA, USA, 2007, DATE '07, pp. 948–953, EDA Consortium.

[55] Mohammad Abdullah Al Faruque, Thomas Ebi, and Jörg Henkel, "Run-time adaptive on-chip communication scheme," in *Proceedings of the 2007 IEEE/ACM international conference on Computer-aided design*, Piscataway, NJ, USA, 2007, ICCAD '07, pp. 26–31, IEEE Press.

[56] Mohammad Abdullah Al Faruque, Rudolf Krist, and Jörg Henkel, "ADAM: run-time agent-based distributed application mapping for on-chip communication," in *Proceedings of the 45th annual Design Automation Conference*, New York, NY, USA, 2008, DAC '08, pp. 760–765, ACM.

[57] Leonel Tedesco, Fabien Clermidy, and Fernando Moraes, "A monitoring and adaptive routing mechanism for QoS traffic on mesh NoC architectures," in *Proceedings of the 7th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, New York, NY, USA, 2009, CODES+ISSS '09, pp. 109–118, ACM.

[58] Chenyang Lu, Xiaorui Wang, and Xenofon Koutsoukos, "Feedback Utilization Control in Distributed Real-Time Systems with End-to-End Tasks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 16, pp. 550–561, June 2005.

[59] Jianguo Yao, Xue Liu, Mingxuan Yuan, and Zonghua Gu, "Online adaptive utilization control for real-time embedded multiprocessor systems," in *Proceedings of the 6th IEEE/ACM/IFIP international conference on Hardware/Software codesign and system synthesis*, New York, NY, USA, 2008, CODES+ISSS '08, pp. 85–90, ACM.

[60] Milan Pastrnak, Peter H. N. de With, and Jef van Meerbergen, "QoS Concept for Scalable MPEG-4 Video Object Decoding on Multimedia (NoC) Chips," *IEEE Transactions on Consumer Electronics*, vol. 52, pp. 1418–1426, November 2006.

[61] Kevin Skadron, T. Abdelzaher, and Mircea R. Stan, "Control-Theoretic Techniques and Thermal-RC Modeling for Accurate and Localized Dynamic Thermal Management," in *HPCA '02: Proceedings of the 2002 High Perf. Comp. Arch.*, 2002, pp. 17–28.

[62] Longhao Shu and Xi Li, "Temperature-aware energy minimization technique through dynamic voltage frequency scaling for embedded systems," in *Education Technology and Computer (ICETC), 2010 2nd International Conference on*, june 2010, vol. 2, pp. V2–515 –V2–519.

[63] Kihwan Choi, Ramakrishna Soma, and Massoud Pedram, "Fine-grained dynamic voltage and frequency scaling for precise energy and performance tradeoff based on the ratio of off-chip access to on-chip computation times," *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, vol. 24, pp. 18–28, 2005.

[64] Andreas Merkel and Frank Bellosa, "Task activity vectors: a new metric for temperature-aware scheduling," in *Eurosys '08: Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008*, 2008, pp. 1–12.

[65] Canturk Isci and Margaret Martonosi, "Runtime Power Monitoring in High-End Processors: Methodology and Empirical Data," in *MICRO 36: Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, 2003, p. 93.

[66] Hamid Reza Pourshaghaghi and José Pineda de Gyvez, "Dynamic voltage scaling based on supply current tracking using fuzzy Logic controller," in *ICECS*, 2009, pp. 779–782.

[67] Omer Khan and Sandip Kundu, "A framework for predictive dynamic temperature management of microprocessor systems," in *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design*, Piscataway, NJ, USA, 2008, ICCAD '08, pp. 258–263, IEEE Press.

[68] M. S. Floyd, S. Ghiasi, T. W. Keller, K. Rajamani, F. L. Rawson, J. C. Rubio, and M. S. Ware, "System power management support in the IBM Power6 microprocessor," *IBM Journal of Research and Development*, vol. 51, no. 6, pp. 733 –746, nov. 2007.

[69] Vincent Nollet, Théodore Marescaux, Diederik Verkest, Jean-Yves Mignolet, and Serge Vernalde, "Operating-system controlled network on chip," in *Proceedings of the 41st annual Design Automation Conference*, New York, NY, USA, 2004, DAC '04, pp. 256–259, ACM.

[70] Andrea Alimonda, Salvatore Carta, Andrea Acquaviva, Alessandro Pisano, and Luca Benini, "A feedback-based approach to DVFS in data-flow applications," *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, vol. 28, no. 11, pp. 1691–1704, Nov. 2009.

[71] Jean-Michel Chabloz and Ahmed Hemani, "Distributed DVFS using rationally-related frequencies and discrete voltage levels," in *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design*, New York, NY, USA, 2010, ISLPED '10, pp. 247–252, ACM.

[72] Ke Meng, Russ Joseph, Robert P. Dick, and Li Shang, "Multi-optimization power management for chip multiprocessors," in *Proceedings of the 17th international conference on*

Parallel architectures and compilation techniques, New York, NY, USA, 2008, PACT '08, pp. 177–186, ACM.

[73] Ayse Kivilcim Coskun, Tajana Šimunic Rosing, and Kenny C. Gross, "Utilizing predictors for efficient thermal management in multiprocessor SoCs," *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, vol. 28, no. 10, pp. 1503–1516, 2009.

[74] Yefu Wang, Kai Ma, and Xiaorui Wang, "Temperature-constrained power control for chip multiprocessors with online model estimation," in *Proceedings of the 36th annual international symposium on Computer architecture*, New York, NY, USA, 2009, ISCA '09, pp. 314–324, ACM.

[75] Andreas Weissel and Frank Bellosa, "Dynamic Thermal Management for Distributed Systems," in *Proceedings of the first workshop on temperature-aware computer systems (TACS'04)*, 2004.

[76] Yong Fu, Nicholas Kottenstette, Yingming Chen, Chenyang Lu, Xenofon D. Koutsoukos, and Hongan Wang, "Feedback Thermal Control for Real-time Systems," in *Proceedings of the 2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, Washington, DC, USA, 2010, RTAS '10, pp. 111–120, IEEE Computer Society.

[77] Rich McGowen, Christopher A. Poirier, Chris Bostak, Jim Ignowski, Mark Millican, Warren H. Parks, and Samuel Naffziger, "Power and Temperature Control on a 90-nm Itanium Family Processor," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 1, pp. 229–237, Jan. 2006.

[78] M. Saen, K. Osada, S. Misaka, T. Yamada, Y. Tsujimoto, Y. Kondoh, T. Kamei, Y. Yoshida, E. Nagahama, Y. Nitta, T. Ito, T. Kameyama, and N. Irie, "Embedded SoC Resource Manager to Control Temperature and Data Bandwidth," in *Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers. IEEE International*, feb. 2007, pp. 296 –604.

[79] Kevin Skadron, Mircea R. Stan, Karthik Sankaranarayanan, Wei Huang, Sivakumar Velusamy, and David Tarjan, "Temperature-aware microarchitecture: Modeling and implementation," *ACM Transactions on Architecture and Code Optimization*, vol. 1, no. 1, pp. 94–125, Sep 2004.

[80] Dennis Sylvester, David Blaauw, and Eric Karl, "ElastIC: An Adaptive Self-Healing Architecture for Unpredictable Silicon," *IEEE Des. Test*, vol. 23, pp. 484–490, November 2006.

[81] David Atienza, Giovanni De Micheli, Luca Benini, José L. Ayala, Pablo G. Del Valle, Michael DeBole, and Vijay Narayanan, "Reliability-aware design for nanometer-scale devices," in *Proceedings of the 2008 Asia and South Pacific Design Automation Conference*, Los Alamitos, CA, USA, 2008, ASP-DAC '08, pp. 549–554, IEEE Computer Society Press.

[82] Philip M. Wells, Koushik Chakraborty, and Gurindar S. Sohi, "Mixed-mode multicore reliability," in *Proceedings of the 14th international conference on Architectural support for programming languages and operating systems*, New York, NY, USA, 2009, ASPLOS '09, pp. 169–180, ACM.

[83] David Fick, Andrew DeOrio, Jin Hu, Valeria Bertacco, David Blaauw, and Dennis Sylvester, "Vicis: a reliable network for unreliable silicon," in *Proceedings of the 46th Annual Design Automation Conference*, New York, NY, USA, 2009, DAC '09, pp. 812–817, ACM.

[84] James Tschanz, Keith Bowman, Chris Wilkerson, Shih-Lien Lu, and Tanay Karnik, "Resilient circuits: enabling energy-efficient performance and reliability," in *Proceedings of the 2009 International Conference on Computer-Aided Design*, New York, NY, USA, 2009, ICCAD '09, pp. 71–73, ACM.

[85] Srinivasan Murali, Theocharis Theocharides, N. Vijaykrishnan, Mary Jane Irwin, Luca Benini, and Giovanni De Micheli, "Analysis of Error Recovery Schemes for Networks on Chips," *IEEE Des. Test*, vol. 22, pp. 434–442, September 2005.

[86] Milos Prvulovic, Zheng Zhang, and Josep Torrellas, "ReVive: cost-effective architectural support for rollback recovery in shared-memory multiprocessors," in *Proceedings of the 29th annual international symposium on Computer architecture*, Washington, DC, USA, 2002, ISCA '02, pp. 111–122, IEEE Computer Society.

[87] Jared C. Smolens, Brian T. Gold, Jangwoo Kim, Babak Falsafi, James C. Hoe, and Andreas G. Nowatzyk, "Fingerprinting: bounding soft-error detection latency and bandwidth," in *Proceedings of the 11th international conference on Architectural support for programming languages and operating systems*, New York, NY, USA, 2004, ASPLOS-XI, pp. 224–234, ACM.

[88] Divya Arora, Srivaths Ravi, Anand Raghunathan, and Niraj K. Jha, "Secure Embedded Processing through Hardware-Assisted Run-Time Monitoring," in *Proceedings of the conference on Design, Automation and Test in Europe - Volume 1*, Washington, DC, USA, 2005, DATE '05, pp. 178–183, IEEE Computer Society.

[89] Milena Milenković, Aleksandar Milenković, and Emil Jovanov, "Hardware support for code integrity in embedded processors," in *Proceedings of the 2005 international conference on Compilers, architectures and synthesis for embedded systems*, New York, NY, USA, 2005, CASES '05, pp. 55–65, ACM.

[90] Leandro Fiorin, Gianluca Palermo, Slobodan Lukovic, Valerio Catalano, and Cristina Silvano, "Secure Memory Accesses on Networks-on-Chip," *IEEE Trans. Comput.*, vol. 57, pp. 1216–1229, September 2008.

[91] N. Nakka, Z. Kalbarczyk, R. K. Iyer, and J. Xu, "An Architectural Framework for Providing Reliability and Security Support," in *Proceedings of the 2004 International Conference on Dependable Systems and Networks*, Washington, DC, USA, 2004, DSN '04, pp. 585–, IEEE Computer Society.

[92] Weidong Shi, Hsien-Hsin S. Lee, Laura 'Falk, and Mrinmoy Ghosh, "An Integrated Framework for Dependable and Revivable Architectures Using Multicore Processors," *SIGARCH Comput. Archit. News*, vol. 34, no. 2, pp. 102–113, May 2006.

[93] Slobodan Lukovic and Nikolaos Christianos, "Hierarchical multi-agent protection system for NoC based MPSoCs," in *Proceedings of the International Workshop on Security and Dependability for Resource Constrained Embedded Systems*, New York, NY, USA, 2010, S&D4RCES '10, pp. 6:1–6:7, ACM.

[94] Lars Bauer, Muhammad Shafique, and Jörg Henkel, "Efficient resource utilization for an extensible processor through dynamic instruction set adaptation," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 16, no. 10, pp. 1295–1308, Oct. 2008.

[95] Roshan G. Ragel, Sri Parameswaran, and Sayed Mohammad Kia, "Micro embedded monitoring for security in application specific instruction-set processors," in *Proceedings of the 2005 international conference on Compilers, architectures and synthesis for embedded systems*, New York, NY, USA, 2005, CASES '05, pp. 304–314, ACM.

[96] Ann Gordon-Ross, Pablo Viana, Frank Vahid, Walid Najjar, and Edna Barros, "A one-shot configurable-cache tuner for improved energy and performance," in *Proceedings of the conference on Design, automation and test in Europe*, San Jose, CA, USA, 2007, DATE '07, pp. 755–760, EDA Consortium.

[97] Moinuddin K. Qureshi and Yale N. Patt, "Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches," in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, Washington, DC, USA, 2006, MICRO 39, pp. 423–432, IEEE Computer Society.

[98] Andreas Moshovos, Gokhan Memik, Alok Choudhary, and Babak Falsafi, "JETTY: Filtering Snoops for Reduced Energy Consumption in SMP Servers," in *Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, Washington, DC, USA,

2001, HPCA '01, pp. 85–, IEEE Computer Society.

[99] Mehdi Modarressi, Hamid Sarbazi-Azad, and Arash Tavakkol, "An efficient dynamically reconfigurable on-chip network architecture," in *Proceedings of the 47th Design Automation Conference*, New York, NY, USA, 2010, DAC '10, pp. 166–169, ACM.

[100] Tae-Hyoung Kim, Randy Persaud, and Chris H. Kim, "Silicon odometer: An on-chip reliability monitor for measuring frequency degradation of digital circuits," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 4, pp. 874–880, 2008.

[101] Prashant Singh, Eric Karl, David Blaauw, and Dennis Sylvester, "Compact Degradation Sensors for Monitoring NBTI and Oxide Degradation," *IEEE Trans. VLSI Syst.*, vol. 20, no. 9, pp. 1645–1655, 2012.

[102] D. Bull, S. Das, K. Shivashankar, G.S. Dasika, K. Flautner, and D. Blaauw, "A Power-Efficient 32 bit ARM Processor Using Timing-Error Detection and Correction for Transient-Error Tolerance and Adaptation to PVT Variation," *Solid-State Circuits, IEEE Journal of*, vol. 46, no. 1, pp. 18 –31, jan. 2011.

[103] Keith A. Bowman, Carlos Tokunaga, James W. Tschanz, Arijit Raychowdhury, Muhammad M. Khellah, Bibiche M. Geuskens, Shih-Lien Lu, Paolo A. Aseron, Tanay Karnik, and Vivek K. De, "All-Digital Circuit-Level Dynamic Variation Monitor for Silicon Debug and Adaptive Clock Control," *IEEE Trans. on Circuits and Systems*, vol. 58-I, no. 9, pp. 2017–2025, 2011.

[104] Victor Champac, Victor Avendaño, and Joan Figueras, "Built-in sensor for signal integrity faults in digital interconnect signals," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 18, no. 2, pp. 256–269, Feb. 2010.

[105] Dongwan Ha, Kyoungho Woo, Scott Meninger, Thucydides Xanthopoulos, Ethan Crain, and Donhee Ham, "Time-Domain CMOS Temperature Sensors With Dual Delay-Locked Loops

for Microprocessor Thermal Monitoring," *IEEE Trans. VLSI Syst.*, vol. 20, no. 9, pp. 1590–1601, 2012.

[106] Poki Chen, Chun-Chi Chen, Yu-Han Peng, Kai-Ming Wang, and Yu-Shin Wang, "A Time-Domain SAR Smart Temperature Sensor With Curvature Compensation and a 3 Inaccuracy of -0.4C ; +0.6C Over a 0C to 90C Range," *J. Solid-State Circuits*, vol. 45, no. 3, pp. 600–609, 2010.

[107] Stefanos Kaxiras and Polychronis Xekalakis, "4T-decay sensors: a new class of small, fast, robust, and low-power, temperature/leakage sensors," in *Proceedings of the 2004 international symposium on Low power electronics and design*, New York, NY, USA, 2004, ISLPED '04, pp. 108–113, ACM.

[108] Srikar Bhagavatula and Byunghoo Jung, "A Low Power Real-time On-Chip Power Sensor in 45-nm SOI," *IEEE Trans. on Circuits and Systems*, vol. 59-I, no. 7, pp. 1577–1587, 2012.

[109] Jieyi Long, Seda Ogrenci Memik, Gokhan Memik, and Rajarshi Mukherjee, "Thermal monitoring mechanisms for chip multiprocessors," *ACM Trans. Archit. Code Optim.*, vol. 5, pp. 9:1–9:33, September 2008.

[110] Yufu Zhang and A. Srivastava, "Accurate Temperature Estimation Using Noisy Thermal Sensors for Gaussian and Non-Gaussian Cases," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 19, no. 9, pp. 1617–1626, Sept. 2011.

[111] Sunpyo Hong and Hyesoon Kim, "An integrated GPU power and performance model," in *Proceedings of the 37th annual international symposium on Computer architecture*, New York, NY, USA, 2010, ISCA '10, pp. 280–289, ACM.

[112] Po-Han Wang, Chia-Lin Yang, Yen-Ming Chen, and Yu-Jung Cheng, "Power gating strategies on GPUs," *ACM Trans. Archit. Code Optim.*, vol. 8, no. 3, pp. 13:1–13:25, Oct. 2011.

[113] Bren C. Mochocki, Kanishka Lahiri, Srihari Cadambi, and X. Sharon Hu, "Signature-based workload estimation for mobile 3D graphics," in *Proceedings of the 43rd annual Design Automation Conference*, New York, NY, USA, 2006, DAC '06, pp. 592–597, ACM.

[114] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood, "Pin: building customized program analysis tools with dynamic instrumentation," *SIGPLAN Not.*, vol. 40, pp. 190–200, June 2005.

[115] Nicholas Nethercote and Julian Seward, "Valgrind: A Framework for Heavyweight Dynamic Binary Instrumentation," in *Proceedings of ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation (PLDI 2007)*, jun 2007, pp. 89–100.

[116] Peter F. Sweeney, Matthias Hauswirth, Brendon Cahoon, Perry Cheng, Amer Diwan, David Grove, and Michael Hind, "Using hardware performance monitors to understand the behavior of java applications," in *Proceedings of the 3rd conference on Virtual Machine Research And Technology Symposium - Volume 3*, Berkeley, CA, USA, 2004, pp. 5–5, USENIX Association.

[117] Thomas E. Bihari and Karsten Schwan, "Dynamic adaptation of real-time software," *ACM Trans. Comput. Syst.*, vol. 9, pp. 143–174, May 1991.

[118] Ahmed Gheith and Karsten Schwan, "CHAOSarc: kernel support for multiweight objects, invocations, and atomicity in real-time multiprocessor applications," *ACM Trans. Comput. Syst.*, vol. 11, pp. 33–72, February 1993.

[119] Shimin Chen, Michael Kozuch, Theodoros Strigkos, Babak Falsafi, Phillip B. Gibbons, Todd C. Mowry, Vijaya Ramachandran, Olatunji Ruwase, Michael Ryan, and Evangelos Vlachos, "Flexible Hardware Acceleration for Instruction-Grain Program Monitoring," in *Proceedings of the 35th Annual International Symposium on Computer Architecture*, Washington, DC, USA, 2008, ISCA '08, pp. 377–388, IEEE Computer Society.

[120] Evangelos Vlachos, Michelle L. Goodstein, Michael A. Kozuch, Shimin Chen, Babak Falsafi, Phillip B. Gibbons, and Todd C. Mowry, "ParaLog: enabling and accelerating online parallel monitoring of multithreaded applications," in *Proceedings of the fifteenth edition of ASPLOS on Architectural support for programming languages and operating systems*, New York, NY, USA, 2010, ASPLOS XV, pp. 271–284, ACM.

[121] Jrgen Teich, "Invasive Algorithms and Architectures (Invasive Algorithmen und Architekturen)," *it - Information Technology*, pp. 300–310, 2008.

[122] Isaías A. Comprés Ureña, Michael Riepen, Michael Konow, and Michael Gerndt, "Invasive MPI on Intel's Single-chip Cloud Computer," in *Proceedings of the 25th International Conference on Architecture of Computing System (ARCS)*, Feb. 2012, pp. 74–85.

[123] Dan Ernst, Nam Sung Kim, Shidhartha Das, Sanjay Pant, Rajeev Rao, Toan Pham, Conrad Ziesler, David Blaauw, Todd Austin, Krisztian Flautner, and Trevor Mudge, "Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation," in *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, Washington, DC, USA, 2003, MICRO 36, pp. 7–, IEEE Computer Society.

[124] J. Tschanz, K. Bowman, Shih-Lien Lu, P. Aseron, M. Khellah, A. Raychowdhury, B. Geuskens, C. Tokunaga, C. Wilkerson, T. Karnik, and V. De, "A 45nm resilient and adaptive microprocessor core for dynamic variation tolerance," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, feb. 2010, pp. 282 –283.

[125] José F. Martínez, Jose Renau, Michael C. Huang, Milos Prvulovic, and Josep Torrellas, "Cherry: checkpointed early resource recycling in out-of-order microprocessors," in *Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture*, Los Alamitos, CA, USA, 2002, MICRO 35, pp. 3–14, IEEE Computer Society Press.

[126] G. Edward Suh, Jae W. Lee, David Zhang, and Srinivas Devadas, "Secure program execution via dynamic information flow tracking," *SIGARCH Comput. Archit. News*, vol. 32, pp. 85–96, October 2004.

[127] S. Subramanya Sastry, Rastislav Bodík, and James E. Smith, "Rapid profiling via stratified sampling," *SIGARCH Comput. Archit. News*, vol. 29, pp. 278–289, May 2001.

[128] Hussam Mousa and Chandra Krintz, "HPS: Hybrid Profiling Support," in *Proceedings of the 14th International Conference on Parallel Architectures and Compilation Techniques*, Washington, DC, USA, 2005, PACT '05, pp. 38–50, IEEE Computer Society.

[129] Abdullah Nazma Nowroz, Ryan Cochran, and Sherief Reda, "Thermal monitoring of real processors: techniques for sensor allocation and full characterization," in *Proceedings of the 47th Design Automation Conference*, New York, NY, USA, 2010, DAC '10, pp. 56–61, ACM.

[130] Dehao Chen, Neil Vachharajani, Robert Hundt, Shih-wei Liao, Vinodha Ramasamy, Paul Yuan, Wenguang Chen, and Weimin Zheng, "Taming hardware event samples for FDO compilation," in *Proceedings of the 8th annual IEEE/ACM international symposium on Code generation and optimization*, New York, NY, USA, 2010, CGO '10, pp. 42–52, ACM.

[131] Wonil Choi, Hyunhee Kim, Wook Song, Jiseok Song, and Jihong Kim, "ePRO-MP: A tool for profiling and optimizing energy and performance of mobile multiprocessor applications," *Sci. Program.*, vol. 17, pp. 285–294, December 2009.

[132] Xiaolan Zhang, Zheng Wang, Nicholas Gloy, J. Bradley Chen, and Michael D. Smith, "System Support for Automatic Profiling and Optimization," *SIGOPS Operating Systems Review*, vol. 31, no. 5, pp. 15–26, 1997.

[133] Jennifer M. Anderson, Lance M. Berc, Jeffrey Dean, Sanjay Ghemawat, Monika R. Henzinger, Shun-Tak A. Leung, Richard L. Sites, Mark T. Vandevoorde, Carl A. Waldspurger,

and William E. Weihl, "Continuous profiling: where have all the cycles gone?," in *Proceedings of the sixteenth ACM symposium on Operating systems principles*, New York, NY, USA, 1997, SOSP '97, pp. 1–14, ACM.

[134] A. Papadogiannakis, A. Kapravelos, M. Polychronakis, E. P. Markatos, and A. Ciuffoletti, "Passive end-to-end packet loss estimation for grid traffic monitoring," in *In Proceedings of the CoreGRID Integration Workshop*, 2006.

[135] Céline Boutros Saab and Xavier Bonnaire, "Cluster Monitoring Platform Based on Self Adaptable Probes," in *IFIP Symposium on Computer Architecture and High Performance Computing*, 2000.

[136] Jia Zhao, S. Madduri, R. Vadlamani, W. Burleson, and R. Tessier, "A Dedicated Monitoring Infrastructure for Multicore Processors," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 19, no. 6, pp. 1011–1022, june 2011.

[137] Hlne N. Lim Choi Keung, Justin R. D. Dyson, Stephen A. Jarvis, and Graham R. Nudd, "Self-Adaptive and Self-Optimising Resource Monitoring for Dynamic Grid Environments," *Database and Expert Systems Applications, International Workshop on*, vol. 0, pp. 689–693, 2004.

[138] M. Munawar and P. Ward, "Adaptive Monitoring in Enterprise Software Systems," in *In SIGMETRICS 2006 Workshop on Tackling Computer Systems Problems with Machine Learning Techniques (SysML)*, 2006.

[139] Keith Bowman and James Tschanz, "Resilient Microprocessor Design for Improving Performance and Energy Efficiency," in *Proceedings of the 2010 International Conference on Computer-Aided Design*, 2010, ICCAD '10, pp. 85–88.

[140] Calin Ciordas, Andreas Hansson, Kees Goossens, and Twan Basten, "A monitoring-aware network-on-chip design flow," *J. Syst. Archit.*, vol. 54, no. 3-4, pp. 397–410, 2008.

[141] P. Gratz, B. Grot, and S.W. Keckler, "Regional congestion awareness for load balance in networks-on-chip," in *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*, feb. 2008, pp. 203 –214.

[142] Christophe Dubach, Timothy M. Jones, Edwin V. Bonilla, and Michael F. P. O'Boyle, "A Predictive Model for Dynamic Microarchitectural Adaptivity Control," in *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, Washington, DC, USA, 2010, MICRO '43, pp. 485–496, IEEE Computer Society.

[143] Konstantinos Aisopos, Chia-Hsin Owen Chen, and Li-Shiuan Peh, "Enabling System-Level Modeling of Variation-Induced Faults in Networks-on-Chip," in *Proceedings of the 48th Design Automation Conference*, 2011, DAC '11.

[144] Babaoglu, Canright, Deutch, Caro, Ducatelle, Gambardella, Ganguly, Jelasity, Montemanni, Montresor, and Urnes, "Design Patterns from Biology for Distributed Computing," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 1, 2006.

[145] Adrian Stoica and Radu Andrei, "Adaptive and Evolvable Hardware - A Multifaceted Analysis," in *Proceedings of the Second NASA/ESA Conference on Adaptive Hardware and Systems*, Washington, DC, USA, 2007, AHS '07, pp. 486–498, IEEE Computer Society.

[146] Yoshiteru Ishida, *Immunity-based systems: a design perspective*, Advanced information processing. Springer, 2004.

[147] Jürgen Becker, Kurt Brändle, Uwe Brinkschulte, Jörg Henkel, Wolfgang Karl, Thorsten Köster, Michael Wenz, and Heinz Wörn, "Digital On-Demand Computing Organism for Real-Time Systems," in *ARCS Workshops*, 2006, pp. 230–245.

[148] David Kramer, Rainer Buchty, and Wolfgang Karl, "A Light-Weight Approach for Online State Classification of Self-organizing Parallel Systems," in *ARCS*, 2011, pp. 183–194.

[149] Hartmut Schmeck, Christian Müller-Schloer, Emre Çakar, Moez Mnif, and Urban Richter, "Adaptivity and self-organization in organic computing systems," *ACM Trans. Auton. Adapt. Syst.*, vol. 5, no. 3, pp. 10:1–10:32, Sept. 2010.

[150] Matthew L. Massie, Brent N. Chun, and David E. Culler, "The ganglia distributed monitoring system: design, implementation, and experience," *Parallel Computing*, vol. 30, no. 5-6, pp. 817–840, 2004.

[151] Steve B. Furber, David R. Lester, Luis A. Plana, Jim D. Garside, Eustace Painkras, Steve Temple, and Brown Andrew D., "Overview of the SpiNNaker system architecture," *IEEE Transactions on Computers*, vol. PP, pp. 1, 2012.

[152] Rajagopal Ananthanarayanan, Steven K. Esser, Horst D. Simon, and Dharmendra S. Modha, "The cat is out of the bag: cortical simulations with 109 neurons, 1013 synapses," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, New York, NY, USA, 2009, SC '09, pp. 63:1–63:12, ACM.

[153] Dominik Fisch, Dominik Fisch, Martin Jänicke, Edgar Kalkowski, and Bernhard Sick, "Techniques for knowledge acquisition in dynamically changing environments," *ACM Trans. Auton. Adapt. Syst.*, vol. 7, no. 1, pp. 16:1–16:25, May 2012.

[154] M. Dilman and D. Raz, "Efficient reactive monitoring," *IEEE J.Sel. A. Commun.*, vol. 20, no. 4, pp. 668–676, Sept. 2006.

[155] Céline Boutros Saab, Xavier Bonnaire, and Bertil Folliot, "PHOENIX: A Self Adaptable Monitoring Platform for Cluster Management," *Cluster Computing*, vol. 5, no. 1, pp. 75–85, Jan. 2002.

[156] Shekhar Borkar, "Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation," *IEEE Micro*, vol. 25, no. 6, pp. 10–16, Nov. 2005.

[157] "CoreSight," .

[158] R. Leatherman, "On-Chip Instrumentation Approach to System-On-Chip Development," 1997.

[159] Érika Cota, Luigi Carro, and Marcelo Lubaszewski, "Reusing an on-chip network for the test of core-based systems," *ACM Transactions on Design Automation of Electronic Systems (TOADES)*, vol. 9, no. 4, pp. 471–499, 2004.

[160] A. Ahmadinia, C. Bobda, J. Ding, M. Majer, J. Teich, S.P. Fekete, and J.C. van der Veen, "A practical approach for circuit routing on dynamic reconfigurable devices," *Rapid System Prototyping, 2005. (RSP 2005). The 16th IEEE International Workshop on*, pp. 84–90, 8-10 June 2005.

[161] T.A. Bartic., J.-Y. Mignolet, V. Nollet, T. Marescaux, D. Verkest, S. Vernalde, and R. Lauwereins, "Topology adaptive network-on-chip design and implementation," *IEE Proceedings on Computers and Digital Techniques*, vol. 152, no. 4, pp. 467–472, July 2005.

[162] C. Zeferino, M. E. Kreutz, and A. A. Susin, "RASoC: a router soft-core for networks-on-chip," in *Design, Automation and Test in Europe Conference (DATE) Proceedings*, Feb. 2004, pp. 198–203.

[163] Balasubramanian Sethuraman, Prasun Bhattacharya, Jawad Khan, and Ranga Vemuri, "LiPaR: A light-weight parallel router for FPGA-based networks-on-chip," in *GLSVSLI '05: Proceedings of the 15th ACM Great Lakes symposium on VLSI*, New York, NY, USA, 2005, pp. 452–457, ACM.

[164] Stamatis Vassiliadis and Ioannis Sourdis, "FLUX interconnection networks on demand," *Journal of Systems Architecture*, vol. 53, no. 10, pp. 777–793, 2007.

[165] A.M. Amory, E. Briao, E. Cota, M. Lubaszewski, and F.G. Moraes, "A scalable test strategy for network-on-chip routers," *Proceedings of the IEEE International Test Conference (ITC 2005)*, pp. 9 pp.–, Nov. 2005.

[166] Leandro Möller, Ismael Grehs, Ewerson Carvalho, Rafael Soares, Ney Calazans, and Fernando Moraes, "A NoC-based Infrastructure to Enable Dynamic Self Reconfigurable Systems," in *Proceedings of the 3rd International Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, 2007, pp. 23–30.

[167] R.B. Mouhoub and O. Hammami, "NoC Monitoring Hardware Support for Fast NoC Design Space Exploration and Potential NoC Partial Dynamic Reconfiguration," *International Symposium on Industrial Embedded Systems (IES '06)*, pp. 1–10, Oct. 2006.

[168] Calin Ciordas, Twan Basten, Andrei Rădulescu, Kees Goossens, and Jef Van Meerbergen, "An event-based monitoring service for networks on chip," *ACM Transactions on Design Automation of Electronic Systems (TOADES)*, vol. 10, no. 4, pp. 702–723, 2005.

[169] M. Mansouri-Samani and M. Sloman, "A configurable event service for distributed systems," *Proceedings of the Third International Conference on Configurable Distributed Systems*, pp. 210–217, 1996.

[170] A. Radulescu, J. Dielissen, S.G. Pestana, O.P. Gangwal, E. Rijpkema, P. Wielage, and K. Goossens, "An efficient on-chip NI offering guaranteed services, shared-memory abstraction, and flexible network configuration," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 1, pp. 4–17, Jan. 2005.

[171] P.D. Amer and L.N. Cassel, "Management of sampled real-time network measurements," *Proceedings of the 14th Conference on Local Computer Networks*, pp. 62–68, 10-12 Oct 1989.

[172] M. Hulboj R. Jurga, "Packet Sampling for Network Monitoring," Tech. Rep., CERN Technical Report, Dec. 2007.

[173] Guanghui He and Jennifer C. Hou, "An In-Depth, Analytical Study of Sampling Techniques for Self-Similar Internet Traffic," in *ICDCS '05: Proceedings of the 25th IEEE International Conference on Distributed Computing Systems*, 2005, pp. 404–413.

[174] M. Schöller, T. Gamer, R. Bless, and M. Zitterbart, "An Extension to Packet Filtering of Programmable Networks," Sophia Antipolis, France, Nov. 2005, Proceedings of the 7th International Working Conference on Active Networking (IWAN).

[175] B. Harangsri, J. Shepherd, and A. Ngu, "Selectivity estimation for joins using systematic sampling," in *Proceedings of the Eighth International Workshop on Database and Expert Systems Applications*, 1-2 Sep 1997, pp. 384–389.

[176] Baek-Young Choi and Supratik Bhattacharrya, "On the Accuracy and Overhead of Cisco Sampled Netflow," in *ACM Sigmetrics Workshop on Large-Scale Network Inference (LSNI)*, Banff, Canada, June 2005.

[177] V. Nollet, T. Marescaux, and D. Verkest, "Operating-system controlled network on chip," in *Proceedings of the 41st Design Automation Conference*, 2004, pp. 256–259.

[178] Milan Pastrnak, Peter H. N. de With, Calin Ciordas, Jef van Meerbergen, and Kees Goossens, "Mixed adaptation and fixed-reservation QoS for improving picture quality and resource usage of multimedia (NoC) chips," in *10th IEEE International Symposium on Consumer Electronics (ISCE)*, 2006.

[179] Calin Ciordas, Andreas Hansson, Kees Goossens, and Twan Basten, "A Monitoring-Aware Network-on-Chip Design Flow," *Journal of Systems Architecture*, 2007.

[180] Calin Ciordas, Andreas Hansson, Kees Goossens, and Twan Basten, "A Monitoring-Aware Network-on-Chip Design Flow," in *DSD '06: Proceedings of the 9th EUROMICRO Conference on Digital System Design*, Washington, DC, USA, 2006, pp. 97–106, IEEE Computer Society.

[181] G. Kornaros, Y. Papaefstathiou, and D. Pnevmatikatos, "Dynamic Software-Assisted Monitoring of On- Chip Interconnects," *DATE'07 Workshop on Diagnostic Services in Network-on-Chips*, Apr. 2007.

[182] K. Goossens, J. Dielissen, and A. Radulescu, "AEthereal network on chip: concepts, architectures, and implementations," *Design and Test of Computers, IEEE*, vol. 22, no. 5, pp. 414–421, Sept.-Oct. 2005.

[183] E. Correa, R. Cardozo, E. Cota, A. Beck, F.R. Wagner, L. Carro, A. Susin, and M. Lubaszewski, "Testing the Wrappers of a Network on Chip: a Case Study," in *Natal, Brazil*, 2003, pp. 159–163.

[184] K. Kim, D. Kim, K. Lee, and H. Yoo, "Cost-Efficient Network-on-Chip Design Using traffic Monitoring System," *DATE'07 Workshop on Diagnostic Services in Network-on-Chips*, Apr. 2007.

[185] Christian El Salloum, Roman Obermaisser, Bernhard Huber, Harald Paulitsch, and Hermann Kopetz, "A time-triggered system-on-a-chip architecture with integrated support for diagnosis," *DATE'07 Workshop on Diagnostic Services in Network-on-Chips*, Apr. 2007.

[186] Jiang Wang, Angelos Stavrou, and Anup Ghosh, "HyperCheck: a hardware-assisted integrity monitor," in *Proceedings of the 13th international conference on Recent advances in intrusion detection*, 2010, RAID'10, pp. 158–177.

[187] Jason Gait, "A probe effect in concurrent programs," *Softw. Pract. Exper.*, vol. 16, no. 3, pp. 225–233, Mar. 1986.

[188] Charles E. McDowell and David P. Helmbold, "Debugging concurrent programs," *ACM Comput. Surv.*, vol. 21, no. 4, pp. 593–622, Dec. 1989.

[189] J.O. Kephart and D.M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, jan 2003.

[190] IBM White Paper, "An architectural blueprint for autonomic computing," 2006.

[191] Nikolaos Voros, Alberto Rosti, and Michael Hbner, *Dynamic System Reconfiguration in Heterogeneous Platforms: The MORPHEUS Approach*, Springer Publishing Company, Incorporated, 1st Edition, 2009.

[192] Andrea Cappelli, Andrea Lodi, Claudio Mucci, Mario Toma, and Fabio Campi, "A Dataflow Control Unit for C-to-Configurable Pipelines Compilation Flow," in *Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2004, FCCM '04, pp. 332–333.

[193] Mladen Berekovic, Andreas Kanstein, Bingfeng Mei, and Bjorn De Sutter, "Mapping of nomadic multimedia applications on the ADRES reconfigurable array processor," *Microprocess. Microsyst.*, vol. 33, no. 4, pp. 290–294, June 2009.

[194] D. Gohringer, M. Hubner, V. Schatz, and J. Becker, "Runtime adaptive multi-processor system-on-chip: RAMPSoC," in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, april 2008, pp. 1 –7.

[195] T. Marescaux, V. Nollet, J.-Y. Mignolet, A. Bartic, W. Moffat, P. Avasare, P. Coene, D. Verkest, S. Vernalde, and R. Lauwereins, "Run-time support for heterogeneous multitasking on reconfigurable SoCs," *Integr. VLSI J.*, vol. 38, no. 1, pp. 107–130, Oct. 2004.

[196] Liang Guang, Ethiopia Nigussie, Jouni Isoaho, Pekka Rantala, and Hannu Tenhunen, "Interconnection alternatives for hierarchical monitoring communication in parallel SoCs," *Microprocess. Microsyst.*, vol. 34, pp. 118–128, August 2010.

[197] J. Zhao, S. Madduri, R. Vadlamani, W. Burleson, and R. Tessier, "A Dedicated Monitoring Infrastructure for Multicore Processors," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 19, no. 6, pp. 1011–1022, June 2011.

[198] Sorin Manolache, Petru Eles, and Zebo Peng, "Buffer space optimisation with communication synthesis and traffic shaping for NoCs," in *Proceedings of the conference on Design,*

automation and test in Europe: Proceedings, 3001 Leuven, Belgium, Belgium, 2006, DATE '06, pp. 718–723, European Design and Automation Association.

[199] Umit Y. Ogras and Radu Marculescu, "Analysis and optimization of prediction-based flow control in networks-on-chip," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 13, pp. 11:1–11:28, February 2008.

[200] Leonel Tedesco, Fabien Clermidy, and Fernando Moraes, "A monitoring and adaptive routing mechanism for QoS traffic on mesh NoC architectures," in *Proceedings of the 7th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, New York, NY, USA, 2009, CODES+ISSS '09, pp. 109–118, ACM.

[201] P. Amer and L. Cassel, "Management of sampled real-time network measurements," in *Proceedings of the 14th Conference on Local Computer Networks*, 1989, pp. 62–68.

[202] Sung-Yong Bang, Kwanhu Bang, Sungroh Yoon, and Eui-Young Chung, "Run-time adaptive workload estimation for dynamic voltage scaling," *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, vol. 28, no. 9, pp. 1334–1347, 2009.

[203] A. Dhodapkar and J.E. Smith, "Managing multi-configuration hardware via dynamic working set analysis," in *ISCA '29: Proceedings of the 29th International Symposium on Computer Architecture*, May 2002.

[204] James Donald and Margaret Martonosi, "Power Efficiency for Variation-Tolerant Multicore Processors," in *Proceedings of the Intl. Symp. on Low Power Electronics and Design*, 2006, pp. 304–309.

[205] Ke Meng, Frank Huebbers, Russ Joseph, and Yehea Ismail, "Physical Resource Matching Under Power Asymmetry," in *P=ac2 Conference, IBM TJ Watson Research Center, Yorktown, NY*, 2006.

[206] Sung Woo Chung and Kevin Skadron, "Using on-chip event counters for high-resolution, real-time temperature measurements," in *In Thermal and Thermomechanical Phenomena in Electronics Systems, 2006*, 2006, pp. 114–120.

[207] Chingren Lee, Jenq Kuen Lee, Tingting Hwang, and Shi-Chun Tsai, "Compiler optimization on VLIW instruction scheduling for low power," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 8, no. 2, pp. 252–268, Apr. 2003.

[208] Fernando Gehm Moraes, Ney Calazans, Aline Mello, Leandro Möller, and Luciano Ost, "HERMES: an infrastructure for low area overhead packet-switching networks on chip," *Integration*, vol. 38, no. 1, pp. 69–93, 2004.

[209] Jia Zhao, Russell Tessier, and Wayne Burleson, "Distributed sensor data processing for many-cores," in *Proceedings of the great lakes symposium on VLSI*, New York, NY, USA, 2012, GLSVLSI '12, pp. 159–164, ACM.

[210] Antonio Flores, Juan L. Aragon, and Manuel E. Acacio, "Heterogeneous Interconnects for Energy-Efficient Message Management in CMPs," *IEEE Transactions on Computers*, vol. 59, pp. 16–28, Jan 2010.

[211] Tushar Krishna, Amit Kumar, Patrick Chiang, Mattan Erez, and Li-Shiuan Peh, "NoC with Near-Ideal Express Virtual Channels Using Global-Line Communication," in *Proceedings of the 16th Symposium on High-Performance Interconnects*, 2008, pp. 11–20.

[212] R.W. Floyd and L. Steinberg, "An adaptive algorithm for spatial grey scale," *Proceedings of the Society of Information Display*, vol. 17, pp. 75–77, 1976.

[213] Amit Kumar, Li Shang, Li-Shiuan Peh, and Niraj K. Jha, "System-Level Dynamic Thermal Management for High-Performance Microprocessors," *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, vol. 27, no. 1, pp. 96–108, 2008.

[214] Inchoon Yeo, Chih Chun Liu, and Eun Jung Kim, "Predictive dynamic thermal management for multicore systems," in *Proceedings of the 45th annual Design Automation Conference*, New York, NY, USA, 2008, DAC '08, pp. 734–739, ACM.

[215] Ryan Cochran and Sherief Reda, "Consistent runtime thermal prediction and control through workload phase detection," in *Proceedings of the 47th Design Automation Conference*, 2010, pp. 62–67.

[216] Seongmoo Heo, Kenneth Barr, and Krste Asanović, "Reducing power density through activity migration," in *Proceedings of the 2003 International Symposium on Low Power Electronics and Design*, 2003, pp. 217–222.

[217] Jingzhao Ou and Viktor K. Prasanna, "A cooperative management scheme for power efficient implementations of real-time operating systems on soft processors," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 16, pp. 45–56, Jan 2008.

[218] Umit Y. Ogras, Radu Marculescu, Puru Choudhary, and Diana Marculescu, "Voltage-Frequency Island Partitioning for GALS-based Networks-on-Chip," in *Proceedings of the 44th Annual Conference on Design Automation*, 2007, pp. 110–115.

[219] George Kornaros, *Application Specific Customizable Embedded Systems*, book chapter 2 in Multi-Core Embedded Systems, CRC Press, Apr 2010.

[220] Andrei Radulescu, John Dielissen, Gonzalez Pestana, Santiago, Prakash Gangwal, Om, Edwin Rijpkema, Paul Wielage, and Kees Goossens, "An Efficient On-Chip NI Offering Guaranteed Services, Shared-Memory Abstraction, and Flexible Network Configuration," *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, vol. 24, no. 1, pp. 4–17, 2005.

[221] Krishnan Srinivasan and Chatha S. Karam, "A technique for low energy mapping and routing in network-on-chip architectures," in *Proceedings of the International Symposium on Low Power Electronics and Design*, 2005, pp. 387–392.

[222] Chenjie Yu and Peter Petrov, "Adaptive multi-threading for dynamic workloads in embedded multiprocessors," in *Proceedings of the 23rd Symposium on Integrated Circuits and System Design*, 2010, pp. 67–72.

[223] Weijia Che and K. S. Chatha, "Scheduling of synchronous data flow models on scratchpad memory based embedded processors," in *Proc. of the International Conference on Computer-Aided Design*, 2010, pp. 205–212.

[224] Andreas Merkel and Frank Bellosa, "Memory-aware scheduling for energy efficiency on multicore processors," in *Proceedings of the 2008 Conference on Power Aware Computing and Systems*, 2008, pp. 1–1.

[225] Xiao Zhang, Kai Shen, Sandhya Dwarkadas, and Rongrong Zhong, "An evaluation of per-chip nonuniform frequency scaling on multicores," in *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, 2010, pp. 19–19.

[226] Jeonghwan Choi, Chen-Yong Cher, Hubertus Franke, Henrdrik Hamann, Alan Weger, and Pradip Bose, "Thermal-aware task scheduling at the system software level," in *Proceedings of the 2007 International Symposium on Low Power Electronics and Design*, 2007, pp. 213–218.

[227] D.C. Pham, T. Aipperspach, D. Boerstler, M. Bolliger, R. Chaudhry, D. Cox, P. Harvey, P.M. Harvey, H.P. Hofstee, C. Johns, J. Kahle, A. Kameyama, J. Keaty, Y. Masubuchi, M. Pham, J. Pille, S. Posluszny, M. Riley, D.L. Stasiak, M. Suzuoki, O. Takahashi, J. Warnock, S. Weitzel, D. Wendel, and K. Yazawa, "Overview of the architecture, circuit design, and physical implementation of a first-generation cell processor," *Solid-State Circuits, IEEE Journal of*, vol. 41, no. 1, pp. 179 – 196, Jan 2006.

[228] Xiuyi Zhou, Jun Yang, Marek Chrobak, and Youtao Zhang, "Performance-aware thermal management via task scheduling," *ACM Trans. Archit. Code Optim.*, vol. 7, pp. 5:1–5:31, May 2010.

[229] Xilinx, Inc., Aurora 8B/10B for Virtex-4 FX FPGA User Guide, UG061, (v3.1), ," [Online] Available: www.xilinx.com/ support/documentation/ip_documentation/virtex_4fx_aurora_8b10b_ug061.pdf, Apr. 2009.

[230] Efraim Rotem, Avi Mendelson, Ran Ginosar, and Uri Weiser, "Multiple clock and voltage domains for chip multi processors," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2009, pp. 459–468.

[231] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *Proceedings of the Workload Characterization, WWC-4*, 2001, pp. 3–14.

[232] Man lap Li, Ruchira Sasanka, Sarita V. Adve, Yen kuang Chen, and Eric Debes, "The ALP-Bench Benchmark Suite for Complex Multimedia Applications," in *Proc. of the IEEE Int. Symp. on Workload Characterization*, 2005, pp. 34–45.

[233] David Brooks, Pradip Bose, and Margaret Martonosi, "Power-performance simulation: design and validation strategies," *SIGMETRICS Performance Evaluation Review*, vol. 31, no. 4, pp. 13–18, 2004.

[234] Chung-hsing Hsu and Wu-chun Feng, "A Power-Aware Run-Time System for High-Performance Computing," in *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, 2005, pp. 1–.

[235] George Kornaros and Dionisios Pnevmatikatos, "Hardware-assisted dynamic power and thermal management in multi-core SoCs," in *Proceedings of the 21st edition of the Great Lakes Symposium on VLSI*, 2011, pp. 115–120.

[236] N.R. Draper and H. Smith, *Applied Regression Analysis*, Wiley-Interscience, 1998.

[237] Min Bao, Alexandru Andrei, Petru Eles, Zebo Peng, and Petru Eles, "Temperature-aware idle time distribution for energy optimization with dynamic voltage scaling," in *Proceedings of the Conference on Design, Automation and Test in Europe*, 2010, pp. 21–26.

[238] Walter Rudin, *Real and complex analysis, 3rd ed.*, McGraw-Hill, Inc., New York, NY, USA, 1987.

[239] Canturk Isci, Gilberto Contreras, and Margaret Martonosi, "Live, Runtime Phase Monitoring and Prediction on Real Systems with Application to Dynamic Power Management," in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, 2006, pp. 359–370.

[240] Yatin Hoskote, Sriram Vangal, Arvind Singh, Nitin Borkar, and Shekhar Borkar, "A 5-GHz Mesh Interconnect for a Teraflops Processor," *IEEE Micro*, vol. 27, pp. 51–61, Sep 2007.

[241] John Kim and Hanjoon Kim, "Router microarchitecture and scalability of ring topology in on-chip networks," in *Proceedings of the 2nd International Workshop on Network on Chip Architectures*, 2009, pp. 5–10.

[242] Marcelo Yuffe, Ernest Knoll, Moty Mehalel, Joseph Shor, and Tsvika Kurts, "A Fully Integrated Multi-CPU, GPU and Memory Controller 32nm Processor," in *Proceedings of the 2011 IEEE International Solid-State Circuits Conference*, 2011, pp. 264–265.

[243] NR Adiga and et al, "An overview of the BlueGene/L Supercomputer," in *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, 2002, Supercomputing '02, pp. 1–22.