A Survey and Taxonomy of On-Chip Monitoring of Multicore Systems-on-Chip

GEORGIOS KORNAROS and DIONISIOS PNEVMATIKATOS, Technical University of Crete

Billion transistor systems-on-chip increasingly require dynamic management of their hardware components and careful coordination of the tasks that they carry out. Diverse real-time monitoring functions assist towards this objective through the collection of important system metrics, such as throughput of processing elements, communication latency, or resource utilization for each application. The online evaluation of these metrics can result in localized or global decisions that attempt to improve aspects of system behavior, system performance, quality-of-service, power and thermal effects under nominal conditions. This work provides a comprehensive categorization of monitoring approaches used in multiprocessor SoCs. As adaptive systems are encountered in many disciplines, it is imperative to present the prominent research efforts in developing online monitoring methods. To this end we offer a taxonomy that groups strongly related techniques that designers increasingly use to produce more efficient and adaptive chips. The provided classification helps to understand and compare architectural mechanisms that can be used in systems, while one can envisage the innovations required to build real adaptive and intelligent systems-on-chip.

Categories and Subject Descriptors: A.1 [**General Literature**]: Introductory and survey general; C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures (Multiprocessors); C.1.3 [**Processor Architectures**]: Other Architecture Styles—*Adaptable architectures*

General Terms: Design, Performance, Reliability, Security

Additional Key Words and Phrases: On-chip monitoring, dynamic management, multicore, network-on-chip, adaptive SoC, on-line debugging, run-time optimization, reconfiguration, diagnosis, fault tolerance, security, proactive SoC management

ACM Reference Format:
Kornaros, G. and Pnevmatikatos, D. 2013. A survey and taxonomy of on-chip monitoring of multicore systems-on-chip. ACM Trans. Des. Autom. Electron. Syst. 18, 2, Article 17 (March 2013), 38 pages.
DOI: http://dx.doi.org/10.1145/2442087.2442088

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

This work is supported in part by the European Community FP7 virtical project under contract FP7-ICT-2011-7, No 288574.
Authors' addresses: G. Kornaros: Technical University of Crete, Dept. of Electronics and Computer Engineering, 73100, Chania, Greece; email: kornaros@mhl.tuc.gr; Technological Educational Institute of Crete, Dept. of Applied Informatics and Multimedia, Estavromenos, 71004, Heraklion, Greece; email: kornaros@epp.teicrete.gr D. Pnevmatikatos: Technical University of Crete, Dept. of Electronics and Computer Engineering, Chania, Greece; email: pnevmati@mhl.tuc.gr; Institute of Computer Science, Foundation for Research and Technology Hellas (FORTH), Vassilika Vouton, Heraklion, Crete, 70013, Greece; email: pnevmati@ics.forth.gr

DOI: http://dx.doi.org/10.1145/2442087.2442088

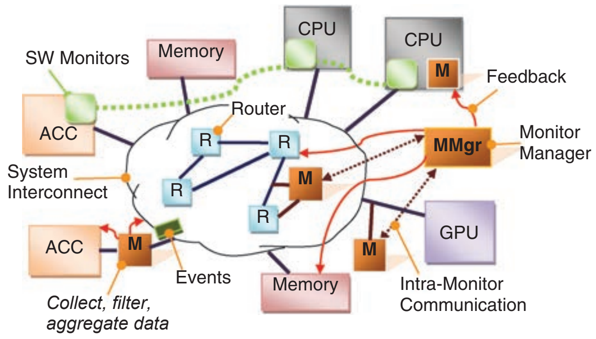

Fig. 1. Monitoring components in modern systems-on-chip involve hardware probes (M) and monitor managers (MMgr) that cooperate with reactors to optimize system performance metrics, energy efficiency, or reliability. The monitoring process consists of events generation, collecting the monitor data of interest, processing these data to decide and providing the right feedback for adaptation.

needs of every application with the offered system platform services. Runtime 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 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 1 depicts an example of a multicore system with hardware observation probes (also interchangeably called *monitors* (M) throughout this article).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 predefined 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.

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* [Heider 1988]. 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 1), may embed monitoring units in either integral way (as dedicated circuits, and/or as software threads), or as external components.

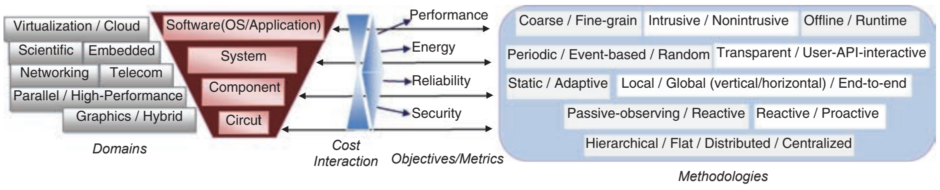

Fig. 2. Monitoring for systems, a wide spectrum of methodologies endorsed for various different objectives; the versatile concept of monitoring has been employed in many domains and disciplines to assist in cost-effective tackling of various optimization or operational problems inside modern SoCs.

The underlying motivation for employing monitoring is to compensate deficiencies of prefabrication 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 nondeterminism, that often lead to the activation of subtle faults. The challenging process of multicore SoC monitoring is analyzed and discussed in the following sections.

1.1. Focus of this Work

This article presents a classification of *monitoring* techniques highlighting the distinctions between major methodologies and revealing underlying relationships as well. The primary emphasis throughout this article 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 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.

Different methods and strategies outlined in Figure 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.

1.2. Significance of Monitoring in the Multicore Era

Emerging multicore architectures are becoming more diverse both at the architecture and at the programming level, while varying application requirements at runtime 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 online 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 [Flynn and Hung 2005], performance is becoming a secondary concern for most designers. Additionally, as the scale and complexity of systems continues to increase, monitoring the system at runtime 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 [Dally and Towles 2001]. 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 [Tilera Corp. 2009] 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 [Intel TeraFLOPS 2010] consists of an 8×10 mesh, which runs at an aggressive clock of 5GHz on a 65nm process. Nvidia's Tegra 3 [Nvidia Tegra 3 2012] 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 operation within the thermal envelope of the chip. For instance, in Power7 chip [Floyd et al. 2011] shown in Figure 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[Nvidia Tegra 3 2012] by individually enabling all five CPU cores (via aggressive power gating) on the basis of the workload.

Figure 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 our work, exploration and classification of various methods and models have appeared in the literature. The growing complexity of modern SoCs has provided a

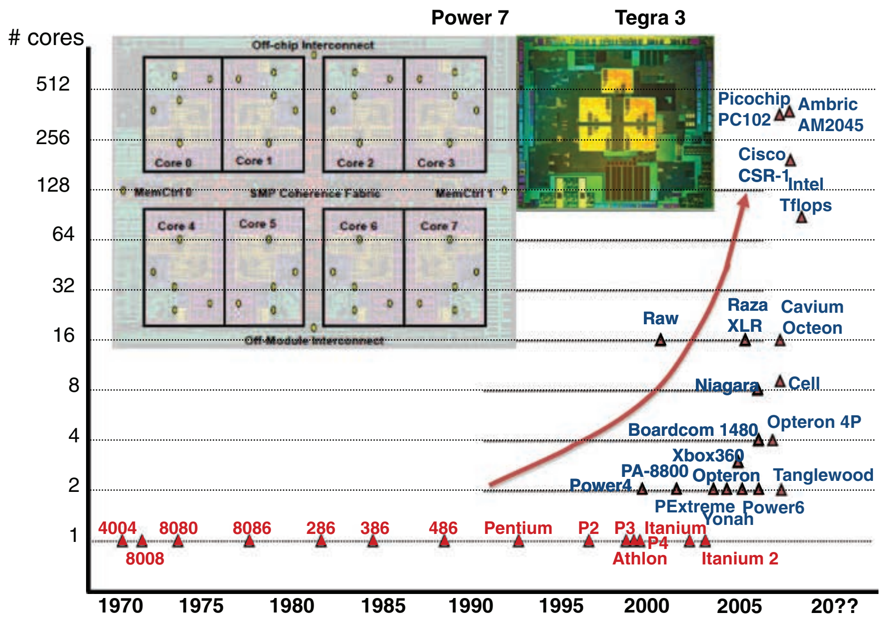

Fig. 3. Exponential growth of multiprocessor SoCs in the recent decade challenge the design of single-chip systems. Chip layouts are adapted from IBM's Power7 chip [Floyd et al. 2011] and Nvidia's Tegra3 [Nvidia Tegra 3].

strong incentive for surveying the methods for online failure prediction through run-time monitoring [Salfner et al. 2010] and methods for fault tolerance in the scope of NoCs [Radetzki et al. 2012]. Benini et al. [2000] have surveyed several approaches to dynamic power management at system-level with a view towards the tradeoffs involved in designing and implementing power-managed systems. Moreover, Kong et al. [2012] provided a comprehensive overview of thermal management techniques for microprocessors. So far, in comparison with this article, 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 our 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.

In Section 2 we give an overview of basic terms and objectives used throughout this article. In Section 3 we develop a functional taxonomy of monitoring techniques designed For processor, chip component and system-level. Additionally, a methodology-based classification is presented in Section 4 through organizing various approaches for monitoring multicore SoCs. Finally, open issues and trends appear in Section 5, which also concludes the article.

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 article 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

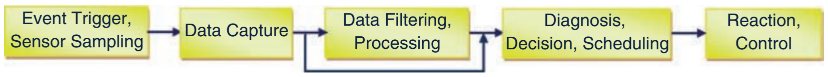
Event Trigger, Sensor Sampling
Data Capture
Data Filtering, Processing
Diagnosis, Decision, Scheduling
Reaction, Control

Fig. 4. Basic components of the online monitoring process.

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" [Hauswirth et al. 2004; Chandy 2006]. 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 [Pouchard et al. 2009].

Essentially, the monitoring process consists of the following steps (see Figure 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

Table I. 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.

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*

calls, context switches, interrupts, shared variable references, and application-specific events, or, (ii) processor, on-chip communication, memory hierarchy, and coprocessing 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 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 I. 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.

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.

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 [Leatherman and Stollon 2005; Vermeulen and Goossens 2010; Daoud and Nicolici 2011]. In industry, various solutions are employed; Xilinx developed Chipscope tool [Xilinx Chipscope] to provide on-chip debug and real-time system visibility for reconfigurable platforms. Embedded processor paradigms such as ARM CoreSight [Coresight] and MIPS on-chip instrumentation [Leatherman 1997] technologies feature nonintrusive 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 [IEEE-ISTO]. 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 [Hopkins and McDonald-Maier 2006], and debug structures for instrumentation, such as assertion-based on-chip debug checkers, triggers and event counters [Abramovici et al. 2006; Daoud and Nicolici 2011]. 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 [de Paula et al. 2011]. Due to the vast amount of debug data, it is more efficient to record higher level information, such as instruction footprints for processors[Park and Mitra 2010].

In addition, infrastructures for Networks-on-Chip have addressed debugging issues [Ciordas et al. 2005], while earlier El Shobaki and Lindh [2001] presented MAMon for event-based debugging of SoC applications, and Cota et al. [2004] described a Test

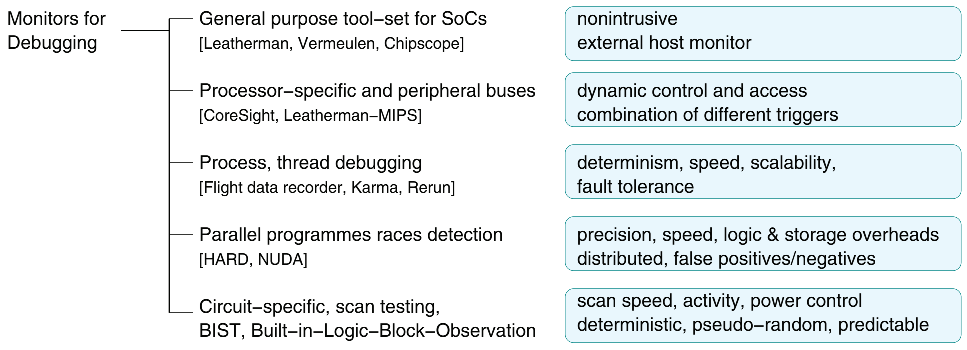
Monitors for Debugging
General purpose tool–set for SoCs
[Leatherman, Vermeulen, Chipscope]
nonintrusive
external host monitor
Processor–specific and peripheral buses
[CoreSight, Leatherman–MIPS]
dynamic control and access
combination of different triggers
Process, thread debugging
[Flight data recorder, Karma, Rerun]
determinism, speed, scalability,
fault tolerance
Parallel programmes races detection
[HARD, NUDA]
precision, speed, logic & storage overheads
distributed, false positives/negatives
Circuit–specific, scan testing,
BIST, Built–in–Logic–Block–Observation
scan speed, activity, power control
deterministic, pseudo–random, predictable

Fig. 5. Subcategories of monitors for debug principally relying on hardware support.

Access Mechanism (TAM) for observing and testing the nodes of a SoC through reusing 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 [Xu et al. 2003], Karma [Basu et al. 2011] and Rerun [Hower et al. 2009] 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 [Zhou et al. 2007] and NUDA [Wen et al. 2012] provide hardware-assisted race detection approaches, using a nonuniform memory mapping. HARD design is architecture-dependent, while NUDA offers a distributed and simultaneous notify mechanism for debugging control.

Figure 5 summarizes the main directions towards embedding special circuitry and mechanisms for debugging and desired goals for each direction.

Even though the power of simulators continues to improve in a linear fashion, the inability to adequately simulate all the possible permutations in presilicon 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 multicore systems, especially for detecting race conditions or deadlocks, requiring architectural enhancements and tools support.

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 runtime 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 [Sprunt 2002], or architectural bottlenecks [Demme and Sethumadhavan 2011]. 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 [Dean et al. 1997]. 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 [Kyung et al. 2007], even in parallel multiprocessors such as Blue Gene [Salapura et al. 2008]. 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 [Intel Xeon], 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.

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 [Marescaux and Corporaal 2007; Mangano and Strano 2010; Sharifi et al. 2010] 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 6 shows monitor units located in NoC routers, which monitor congestion by estimating link utilization [van den Brand et al. 2007], 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 [Marescaux and Corporaal 2007]. 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 runtime [Al Faruque et al. 2007, 2008; Tedesco et al. 2009]. Monitors are

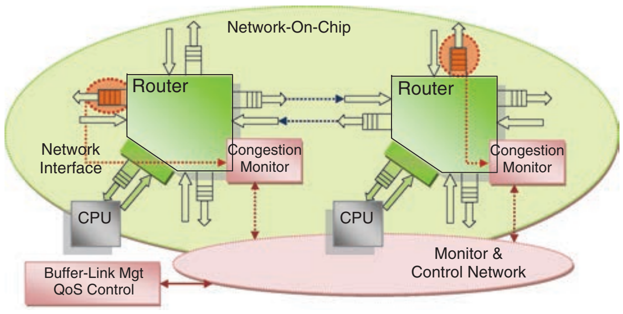

Fig. 6. Distributed approach for QoS control of a NoC through monitoring buffer or link utilization.

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 [Lu et al. 2005; Yao et al. 2008]. 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. 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 multicore 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 [Pastrnak et al. 2006].

3.4. Monitors for Power, Energy and Temperature

As the power densities increase with the continuous shrinking geometries, thermal hotspots 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 scaling (DVFS) configuration for specific or different workloads.

The past decade has witnessed many research efforts in this domain since the design space exploration for runtime power and temperature management involves a very large number of parameters. Figure 7 shows a state-of-the-art summary of approaches for a single processor and some in the multicore era [Skadron et al. 2002; Shu and Li 2010; Choi et al. 2005; Merkel and Bellosa 2008; Isci and Martonosi 2003; Pourshaghaghi and de Gyvez 2009; Khan and Kundu 2008; Floyd et al. 2007; Nollet et al. 2004; Alimonda et al. 2009; Chabloz and Hemani 2010; Meng et al. 2008; Coskun et al. 2009; Wang et al. 2009]. The main goal is to fit the best monitoring strategy

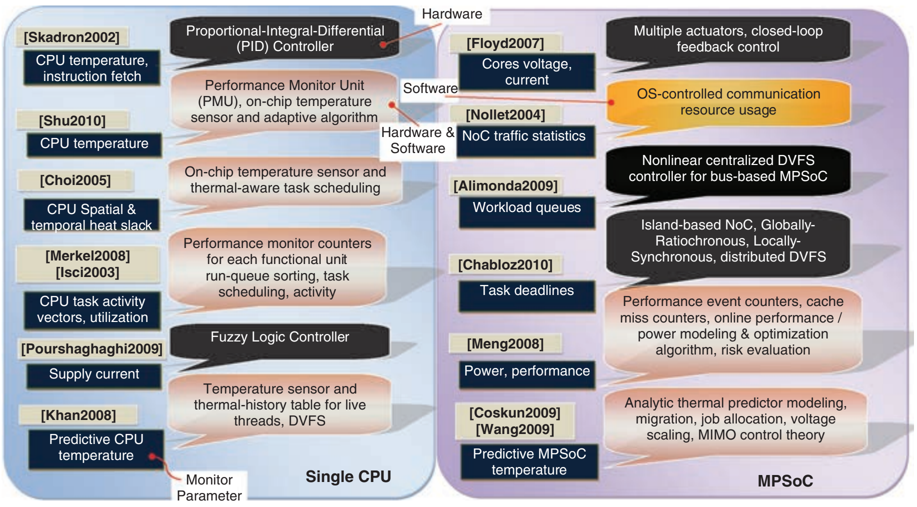
Hardware
Software
Hardware & Software
Monitor Parameter
[Skadron2002]
CPU temperature, instruction fetch
Proportional-Integral-Differential (PID) Controller
[Shu2010]
CPU temperature
Performance Monitor Unit (PMU), on-chip temperature sensor and adaptive algorithm
[Choi2005]
CPU Spatial & temporal heat slack
On-chip temperature sensor and thermal-aware task scheduling
[Merkel2008] [Isci2003]
CPU task activity vectors, utilization
Performance monitor counters for each functional unit run-queue sorting, task scheduling, activity
[Pourshaghaghi2009]
Supply current
Fuzzy Logic Controller
[Khan2008]
Predictive CPU temperature
Temperature sensor and thermal-history table for live threads, DVFS
Single CPU
[Floyd2007]
Cores voltage, current
Multiple actuators, closed-loop feedback control
[Nollet2004]
NoC traffic statistics
OS-controlled communication resource usage
[Alimonda2009]
Workload queues
Nonlinear centralized DVFS controller for bus-based MPSoC
[Chabloz2010]
Task deadlines
Island-based NoC, Globally-Ratiochronous, Locally-Synchronous, distributed DVFS
[Meng2008]
Power, performance
Performance event counters, cache miss counters, online performance / power modeling & optimization algorithm, risk evaluation
[Coskun2009] [Wang2009]
Predictive MPSoC temperature
Analytic thermal predictor modeling, migration, job allocation, voltage scaling, MIMO control theory
MPSoC

Fig. 7. Monitor-based approaches for energy and temperature management for single- and multi- processor systems.

(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, multithreaded environment, high-performance processors, multilevel memory hierarchies, embedded systems, real-time, or embedded OS.

An intrusive or nonintrusive 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 [Isci and Martonosi 2003]. Through performance monitoring counters the functional units utilization can be determined as a way to infer energy requirements and corresponding temperature [Weissel and Bellosa 2004]. Accurate monitor information is of prime importance to develop *task activity vectors*, as defined in [Merkel and Bellosa 2008], 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 [Skadron et al. 2002; Coskun et al. 2009; Fu et al. 2010].

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 [McGowen et al. 2006; Saen et al. 2007; Floyd et al. 2007].

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 [Skadron et al. 2004]. 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 [Khan and Kundu 2008], 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 [Nollet et al. 2004].

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.

3.5. Monitors for Fault Tolerance and Reliability

Ensuring reliable systems involves multiple abstraction layers from circuit and microarchitecture 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 [Sylvester et al. 2006]. However, centralized monitor schemes represent a dependability exposure due to their single point of failure vulnerability.

Online monitors intend to extract metrics for accurate determination of aging models or for complete reliability analysis [Atienza et al. 2008], 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 [Wells et al. 2009]. 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 submicron technologies for dynamic detection and correction of errors. A combination of *in situ* error-detecting circuits and microarchitectural recovery mechanisms enhance system robustness in the face of various errors [Fick et al. 2009]. Most adaptive techniques employ tracking circuits, such as tunable replica circuits [Tschanz et al. 2009], to compensate for manifestations of voltage droops, of process-voltage-temperature (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 [Murali et al. 2005], or by supporting multipath 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 [Prvulovic et al. 2002]. 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 [Smolens et al. 2004]. These methods along with robust on-chip monitor communication can help to ensure monitoring for reliability.

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 [Arora et al. 2005]. For example, custom signature verification units [Milenković et al. 2005] 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 [Fiorin et al. 2008]. Monitor coprocessor units have been proposed, such as a Reliability and Security Engine (RSE) [Nakka et al. 2004], 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 [Shi et al. 2006]. 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 [Lukovic and Christianos 2010].

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 online exploration of the best performance/energy parameter values that could be customized related to an application's memory usage characteristics. Runtime 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 [Bauer et al. 2008]. 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. [2005] describe a methodology for monitoring routines to check insecure operations through the addition of extra microinstructions 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 runtime monitoring methods employ sophisticated circuitry. Gordon-Ross et al. [2007] describe a hardware cache-tuning module that nonintrusively 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 [2006] 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. [2001] 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 [Dally and Towles 2001; Modarressi et al. 2010]. 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.

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, runtime monitoring strategies are evolving with an increased dynamism becoming a key part of systems design.

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 preceding 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 nonintrusive, whether the monitor behavior can perturb program execution or the data flow
—Organization: hierarchical, centralized and distributed monitors
—Reactivity: reactive and proactive 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.

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,

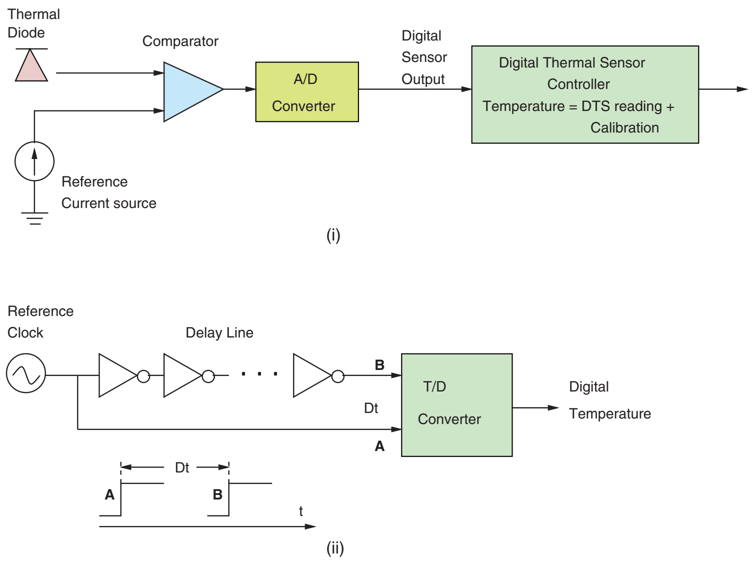

Fig. 8. Digital temperature sensors with (i) a thermal diode, analog-to-digital (A/D) converter and calibration, and (ii) a delay-locked loop using inverters and a time-to-digital (TD) converter.

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 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.

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. [2008] 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 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. [2012] propose compact sensors for devices undergoing NBTI and defect-induced oxide breakdown. Common practice using ring

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

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. [2011] 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) [Bowman et al. 2011] 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. [2010] showing an adapted multisignal 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. [2012] 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. [2010b] 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 [Kaxiras and Xekalakis 2004]. 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 blockwise 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 [Bhagavatula and Jung 2012]. 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. Multisensor 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 [Sharifi et al. 2010], 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 [Long et al. 2008]. To this end, Zhang et al. [2011] 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.

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 [Isci and Martonosi 2003], or for the GPU architecture and the GPGPU kernels [Hong and Kim 2010]. The following equation gives the fundamental power model as introduced by Isci and Martonosi [2003].

$$\begin{aligned} Power \; = \; & \sum_{i=0}^{n} (AccessRate(C_i) \times ArchitecturalScaling(C_i) \times MaxPower(C_i) \\ & + NonGatedClockPower(C_i)) + IdlePower. \end{aligned}$$

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. [2011] 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. [2006] 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).

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 [Luk et al. 2005; Nethercote and Seward 2007] 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. [2004] 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 [Sweeney et al. 2004] 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 a 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 online or postmortem exploration of program trace or output data with the objective to debug programs or to profile its runtime 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 online 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 online adaptation of functional application components for managing reliability versus performance trade-offs in parallel and real-time applications [Bihari and Schwan 1991], and to load-balancing or program configuration for enhanced reliability in distributed systems [Gheith and Schwan 1993].

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 offline monitoring events can be sorted. However, for online monitoring event reordering must be performed online 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.

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. Multiuse sensors and their hardware controllers that can monitor performance, degradation, temperature or power consumption as the circuit ages are becoming a fundamental subsystem of multicore SoCs [Sylvester et al. 2006].

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 [Ernst et al. 2003] and error-detection sequential (EDS) [Tschanz et al. 2010] 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 runtime monitoring of tasks. These efforts have resulted in hardware-based monitoring tools [Martínez et al. 2002; Suh et al. 2004] which are fast but they require specialized hardware support in the form of wholesale changes to the processor pipeline, memory management and the caches.

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 for instance, 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 nonintrusiveness 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 [Sastry et al. 2001], and for sampling and processing of profiling data [Mousa and Krintz 2005]. 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 runtime 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 [Nowroz et al. 2010] 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 [Chen et al. 2010a]. Towards a different direction Choi et al. [2009] describe a hybrid methodology and infrastructure to profile and optimize the performance and energy consumption of multithreaded 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.

4.6. Event Sampling

Time-based sampling term is used when system cycles, rather than events, are counted [Zhang et al. 1997; Chen et al. 2010a]. 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 [Anderson et al. 1997].

From a slightly different perspective, by bringing methods and terms from data networking to on-chip interconnects [Papadogiannakis et al. 2006], 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 [Boutros Saab and Bonnaire 2000]. However, dynamically ranging the awareness level usually entails latency and complexity overheads.

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 online processing of the captured data. In Wen et al. [2012] a debugging coprocessor is responsible to gather and organize collected information as monitored record in a histogram-table. When the coprocessor receive a command packet invokes the indicated debugging operations, such as stop, step, continue the execution. However, the repercussion of online 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 subclock cycle errors, such as *Transition Detector* in Razor-based processor [Bull et al. 2011] 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 follows.

—*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 Zhao et al. [2011]. To provide on-chip monitoring of temperature effects Zhao et al. [2011] 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 runtime, based on environmental limits, such as temperature, voltage, and faults, or changes in priorities. For instance, similar to the strategy proposed in Keung et al. [2004], 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 Munawar and Ward [2006]. This monitor attempts to reduce monitoring overhead by preselecting 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.

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 nonintrusive 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, multiclock 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

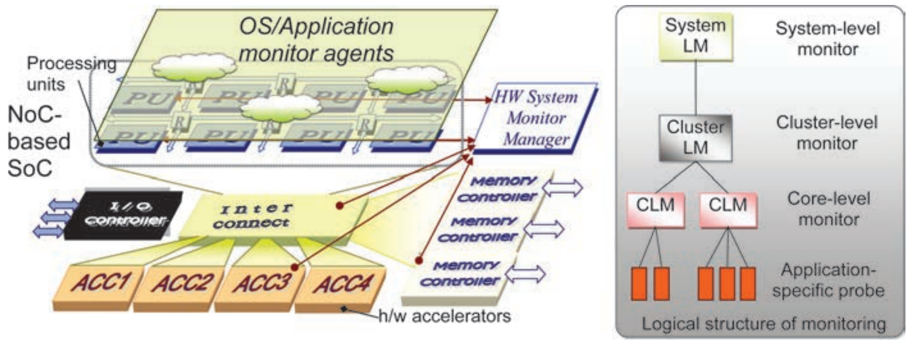

Fig. 9. Hierarchical monitoring in a multicore SoC at software and platform level. The logical organization can be mapped to either level. Software monitoring provides greater flexibility and scalability but often worse performance, while the hardware approach yields better performance but requires more design effort.

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 Bowman and Tschanz [2010], 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 Ciordas et al. [2008], 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 nonintrusive, as only the monitoring subnetwork consisting of dedicated links is used for transporting the monitoring data. However, in general, by sharing any system resource, nonintrusiveness is potentially not guaranteed and must be enforced.

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 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 multicore 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.

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 Tedesco et al. [2009] 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. [2008], 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.

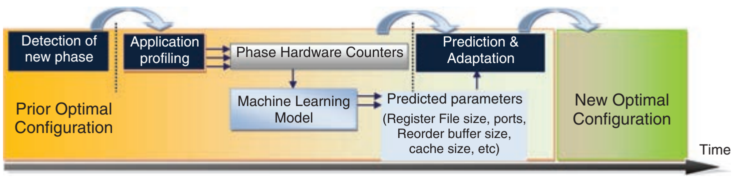
Detection of new phase
Application profiling
Phase Hardware Counters
Prediction & Adaptation
Prior Optimal Configuration
Machine Learning Model
Predicted parameters (Register File size, ports, Reorder buffer size, cache size, etc)
New Optimal Configuration
Time

Fig. 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]).

4.10. Monitoring for Reactive and Proactive 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 multicore embedded SoCs has opened the opportunity for adaptive real-time proactive management [Coskun et al. 2009; Dubach et al. 2010]. Runtime resource management techniques have emerged, as shown in Figure 10 that employ predicting of the best hardware configuration for any phase of a program to maximize energy efficiency. The target is to devise proactive 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) proactive 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

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 [Aisopos et al. 2011].

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 [Babaoglu et al. 2006]. Emergent intelligent technologies such as *evolvable hardware* [Stoica and Andrei 2007], 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 nonself but also the self, (ii) a distributed and adaptive system with autonomous components capable of mutual evaluation [Ishida 2004].

Adopting biological concepts, hierarchical and distributed monitoring infrastructures have been proposed for online state collecting and subsequent evaluation of information [Becker et al. 2006; Kramer et al. 2011; Schmeck et al. 2010; Massie et al. 2004]. 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 [Becker et al. 2006]. 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 [Furber et al. 2012; Ananthanarayanan et al. 2009], which set as ultimate goal to simulate the behavior of aggregates of billion neurons in real time. In the SpiNNaker system prototype chip [Furber et al. 2012] 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.

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 multimonitor and multiagent 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 [Fisch et al. 2012].

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 [Sylvester et al. 2006]. 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., Dilman and Raz [2006]) by exploring algorithms that combine polling and event reporting aiming at minimizing the overall communication cost, and in distributed systems (e.g., Saab et al. [2002]). 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 multicore 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 runtime 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 for NoC application data and for monitoring data can potentially provide suboptimal 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

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	Runtime 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	Nonintrusive debugging frame-work for many-core systems, operates in parallel to the original data interconnection, enabling "nonintrusive" 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. Runtime 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.	Runtime 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

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 [Borkar 2005]. 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.

REFERENCES

Abramovici, M., Bradley, P., Dwarakanath, K., Levin, P., Memmi, G., and Miller, D. 2006. A reconfigurable design-for-debug infrastructure for socs. In *Proceedings of the 43rd annual Design Automation Conference (DAC'06)*. ACM, New York, NY, 7–12.

Aisopos, K., Chen, C.-H. O., and Peh, L.-S. 2011. Enabling system-level modeling of variation-induced faults in networks-on-chip. In *Proceedings of the 48th Design Automation Conference (DAC'11)*.

Al Faruque, M. A., Ebi, T., and Henkel, J. 2007. Run-time adaptive on-chip communication scheme. In *Proceedings of the IEEE/ACM international conference on Computer-aided design (ICCAD'07)*. IEEE, 26–31.

Al Faruque, M. A., Krist, R., and Henkel, J. 2008. ADAM: run-time agent-based distributed application mapping for on-chip communication. In *Proceedings of the 45th Annual Design Automation Conference (DAC'08)*. ACM, New York, NY, 760–765.

ALIMONDA, A., CARTA, S., ACQUAVIVA, A., PISANO, A., AND BENINI, L. 2009. A feedback-based approach to dvfs in data-flow applications. *Trans. Comp.-Aided Des. Integ. Circ. Sys. 28*, 11, 1691–1704.

ANANTHANARAYANAN, R., ESSER, S. K., SIMON, H. D., AND MODHA, D. S. 2009. 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 (SC'09)*. ACM, New York, NY, 63:1–63:12.

ANDERSON, J. M., BERC, L. M., DEAN, J., GHEMAWAT, S., HENZINGER, M. R., LEUNG, S.-T. A., SITES, R. L., VANDEVOORDE, M. T., WALDSPURGER, C. A., AND WEIHL, W. E. 1997. Continuous profiling: where have all the cycles gone? In *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP'97)*. ACM, New York, NY, 1–14.

ARORA, D., RAVI, S., RAGHUNATHAN, A., AND JHA, N. K. 2005. Secure embedded processing through hardware-assisted run-time monitoring. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE'05)*. IEEE, 178–183.

ATIENZA, D., DE MICHELI, G., BENINI, L., AYALA, J. L., VALLE, P. G. D., DEBOLE, M., AND NARAYANAN, V. 2008. Reliability-aware design for nanometer-scale devices. In *Proceedings of the 2008 Asia and South Pacific Design Automation Conference (ASP-DAC'08)*. IEEE, 549–554.

BABAOGLU, O., CANRIGHT, G., ET AL. 2006. Design patterns from biology for distributed computing. *ACM Trans. Auton. Adapt. Syst. 1*.

BASU, A., BOBBA, J., AND HILL, M. D. 2011. Karma: scalable deterministic record-replay. *In Proceedings of the International Conference on Supercomputing (ICS'11)*. ACM, New York, NY, 359–368.

BAUER, L., SHAFIQUE, M., AND HENKEL, J. 2008. Efficient resource utilization for an extensible processor through dynamic instruction set adaptation. *IEEE Trans. VLSI Syst. 16*, 10, 1295–1308.

BECKER, J., BRÄNDLE, K., BRINKSCHULTE, U., HENKEL, J., KARL, W., KÖSTER, T., WENZ, M., AND WÖRN, H. 2006. Digital on-demand computing organism for real-time systems. In *Proceedings of the 19th International Conference on Architecture of Computing Systems Workshops*. 230–245.

BENINI, L., BOGLIOLO, A., AND MICHELI, G. D. 2000. A survey of design techniques for system-level dynamic power management. *IEEE Trans. VLSI Syst. 8*, 3, 299–316.

BHAGAVATULA, S. AND JUNG, B. 2012. A low power real-time on-chip power sensor in 45-nm SOI. *IEEE Trans. Circuits Syst. 59-I*, 7, 1577–1587.

BIHARI, T. E. AND SCHWAN, K. 1991. Dynamic adaptation of real-time software. *ACM Trans. Comput. Syst. 9*, 143–174.

BORKAR, S. 2005. Designing reliable systems from unreliable components: The challenges of transistor variability and degradation. *IEEE Micro 25*, 6, 10–16.

BOUTROS SAAB, C. AND BONNAIRE, X. 2000. Cluster monitoring platform based on self adaptable probes. In *Proceedings of the IFIP Symposium on Computer Architecture and High Performance Computing*.

BOWMAN, K. AND TSCHANZ, J. 2010. Resilient microprocessor design for improving performance and energy efficiency. In *Proceedings of the International Conference on Computer-Aided Design (CCAD'10)*. 85–88.

BOWMAN, K. A., TOKUNAGA, C., ET AL. 2011. All-digital circuit-level dynamic variation monitor for silicon debug and adaptive clock control. *IEEE Trans. Circuits Syst 58-I*, 9, 2017–2025.

BULL, D., DAS, S., SHIVASHANKAR, K., DASIKA, G., FLAUTNER, K., AND BLAAUW, D. 2011. A power-efficient 32 bit ARM processor using timing-error detection and correction for transient-error tolerance and adaptation to PVT variation. *IEEE J. Solid-State Circuits 46*, 1, 18 –31.

CHABLOZ, J.-M. AND HEMANI, A. 2010. 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 (ISLPED'10)*. ACM, New York, NY, 247–252.

CHAMPAC, V., AVENDAÑO, V., AND FIGUERAS, J. 2010. Built-in sensor for signal integrity faults in digital interconnect signals. *IEEE Trans. VLSI Syst. 18*, 2, 256–269.

CHANDY, M. K. 2006. Event-driven applications: Costs, benefits and design approaches. Gartner Application Integration and Web Services Summit, San Diego, CA,

CHEN, D., VACHHARAJANI, N., HUNDT, R., LIAO, S.-W., RAMASAMY, V., YUAN, P., CHEN, W., AND ZHENG, W. 2010a. Taming hardware event samples for FDO compilation. In *Proceedings of the 8th Annual IEEE/ACM International Symposium on Code Generation and Optimization (CGO'10)*. ACM, New York, NY, 42–52.

CHEN, P., CHEN, C.-C., PENG, Y.-H., WANG, K.-M., AND WANG, Y.-S. 2010b. 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 45*, 3, 600–609.

CHOI, K., SOMA, R., AND PEDRAM, M. 2005. Fine-grained dynamic voltage and frequency scaling for precise energy and performance trade-off based on the ratio of off-chip access to on-chip computation times. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 24*, 18–28.

Choi, W., Kim, H., Song, W., Song, J., and Kim, J. 2009. ePRO-MP: A tool for profiling and optimizing energy and performance of mobile multiprocessor applications. *Sci. Program. 17*, 285–294.

Ciordas, C., Basten, T., Rădulescu, A., Goossens, K., and Meerbergen, J. V. 2005. An event-based monitoring service for networks on chip. *ACM Trans. Des. Autom. Electron. Syst. 10*, 4, 702–723.

Ciordas, C., Hansson, A., Goossens, K., and Basten, T. 2008. A monitoring-aware network-on-chip design flow. *J. Syst. Archit. 54*, 3–4, 397–410.

Coresight, A. www.arm.com/products/solutions/CoreSight.html.

Coskun, A. K., Rosing, T. V., and Gross, K. C. 2009. Utilizing predictors for efficient thermal management in multiprocessor SoCs. *IEEE Trans. Comp.-Aided Des. Integ. Cir. Sys. 28*, 10, 1503–1516.

Cota, E., Carro, L., and Lubaszewski, M. 2004. Reusing an on-chip network for the test of core-based systems. *ACM Trans. Des. Autom. Electron. Syst. 9*, 4, 471–499.

Dally, W. J. and Towles, B. 2001. Route packets, not wires: on-chip inteconnection networks. In *Proceedings of the 38th annual Design Automation Conference (DAC'01)*. ACM, New York, NY, 684–689.

Daoud, E. and Nicolici, N. 2011. Embedded debug architecture for bypassing blocking bugs during post-silicon validation. *IEEE Trans. VLSI Syst. 19*, 4, 559 –570.

de Paula, F. M., Nahir, A., Nevo, Z., Orni, A., and Hu, A. J. 2011. TAB-backspace: Unlimited-length trace buffers with zero additional on-chip overhead. In *Proceedings of the 48th Design Automation Conference (DAC'11)*. ACM, New York, NY, 411–416.

Dean, J., Hicks, J. E., Waldspurger, C. A., Weihl, W. E., and Chrysos, G. 1997. ProfileMe: hardware support for instruction-level profiling on out-of-order processors. In *Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture*. IEEE, 292–302.

Demme, J. and Sethumadhavan, S. 2011. Rapid identification of architectural bottlenecks via precise event counting. In *Proceedings of the 38th Annual International Symposium on Computer architecture (ISCA '11)*. ACM, New York, NY, 353–364.

Dilman, M. and Raz, D. 2006. Efficient reactive monitoring. *IEEE J. Sel. Areas Comm. 20*, 4, 668–676.

Dubach, C., Jones, T. M., Bonilla, E. V., and O'Boyle, M. F. P. 2010. A predictive model for dynamic microarchitectural adaptivity control. In *Proceedings of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 485–496.

El Shobaki, M. and Lindh, L. 2001. A hardware and software monitor for high-level system-on-chip verification. In *Proceedings of the International Symposium on Quality Electronic Design*. 56–61.

Ernst, D., Kim, N. S., Das, S., Pant, S., Rao, R., Pham, T., Ziesler, C., Blaauw, D., Austin, T., Flautner, K., and Mudge, T. 2003. Razor: A low-power pipeline based on circuit-level timing speculation. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 7.

Fick, D., DeOrio, A., Hu, J., Bertacco, V., Blaauw, D., and Sylvester, D. 2009. Vicis: A reliable network for unreliable silicon. In *Proceedings of the 46th Annual Design Automation Conference (DAC'09)*. ACM, New York, NY, 812–817.

Fiorin, L., Palermo, G., Lukovic, S., Catalano, V., and Silvano, C. 2008. Secure memory accesses on networks-on-chip. *IEEE Trans. Comput. 57*, 1216–1229.

Fisch, D., Fisch, D., J̈anicke, M., Kalkowski, E., and Sick, B. 2012. Techniques for knowledge acquisition in dynamically changing environments. *ACM Trans. Auton. Adapt. Syst. 7*, 1, 16:1–16:25.

Floyd, M., Allen-Ware, M., Rajamani, K., Brock, B., Lefurgy, C., Drake, A. J., Pesantez, L., Gloekler, T., Tierno, J. A., Bose, P., and Buyuktosunoglu, A. 2011. Introducing the adaptive energy management features of the Power7 chip. *IEEE Micro 31*, 60–75.

Floyd, M. S., Ghiasi, S., Keller, T. W., Rajamani, K., Rawson, F. L., Rubio, J. C., and Ware, M. S. 2007. System power management support in the IBM Power6 microprocessor. *IBM J. Res. Dev. 51*, 6, 733 –746.

Flynn, M. J. and Hung, P. 2005. Microprocessor design issues: Thoughts on the road ahead. *IEEE Micro 25*, 16–31.

Fu, Y., Kottenstette, N., Chen, Y., Lu, C., Koutsoukos, X. D., and Wang, H. 2010. Feedback thermal control for real-time systems. In *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'10)*. IEEE, 111–120.

Furber, S. B., Lester, D. R., Plana, L. A., Garside, J. D., Painkras, E., Temple, S., and Brown, A. 2012. Overview of the SpiNNaker system architecture. *IEEE Trans. Comput. PP*, 1.

Gheith, A. and Schwan, K. 1993. CHAOSarc: kernel support for multiweight objects, invocations, and atomicity in real-time multiprocessor applications. *ACM Trans. Comput. Syst. 11*, 33–72.

Gordon-Ross, A., Viana, P., Vahid, F., Najjar, W., and Barros, E. 2007. A one-shot configurable-cache tuner for improved energy and performance. In *Proceedings of the Conference on Design, Automation and TEST in Europe (DATE'07)*. EDA Consortium, San Jose, CA, 755–760.

GRATZ, P., GROT, B., AND KECKLER, S. 2008. Regional congestion awareness for load balance in networks-on-chip. In *Proceedings of the IEEE 14th International Symposium on High Performance Computer Architecture (HPCA'08)*. 203 –214.

HA, D., WOO, K., MENINGER, S., XANTHOPOULOS, T., CRAIN, E., AND HAM, D. 2012. Timedomain cmos temperature sensors with dual delay-locked loops for microprocessor thermal monitoring. *IEEE Trans. VLSI Syst. 20*, 9, 1590–1601.

HAUSWIRTH, M., SWEENEY, P. F., DIWAN, A., AND HIND, M. 2004. Vertical profiling: understanding the behavior of object-priented applications. *SIGPLAN Not. 39*, 251–269.

HEIDER, K. G. 1988. The Rashomon effect: When ethnographers disagree. *Amer. Anthropologist 90*, 1, 73–81.

HONG, S. AND KIM, H. 2010. An integrated GPU power and performance model. In *Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA'10)*. ACM, New York, NY, 280–289.

HOPKINS, A. B. T. AND MCDONALD-MAIER, K. D. 2006. Debug support strategy for systems-on-chips with multiple processor cores. *IEEE Trans. Comput. 55*, 2, 174–184.

HOWER, D. R., MONTESINOS, P., CEZE, L., HILL, M. D., AND TORRELLAS, J. 2009. Two hardware-based approaches for deterministic multiprocessor replay. *Comm. ACM 52*, 93–100.

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

INTEL TERAFLOPS. 2010. http://www.intel.com/content/www/us/en/research/intel-labsteraflops-research-chip.html.

INTEL XEON. 2012. Processor E5-2600 Product family uncore performance monitoring guide. www.intel.com.

ISCI, C. AND MARTONOSI, M. 2003. Runtime power monitoring in high-end processors: Methodology and empirical data. In *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*. 93.

ISHIDA, Y. 2004. *Immunity-Based Systems: A Design Perspective. Advanced Information Processing*. Springer.

KAXIRAS, S. AND XEKALAKIS, P. 2004. 4T-decay sensors: a new class of small, fast, robust, and low-power, temperature/leakage sensors. In *Proceedings of the International Symposium on Low Power Electronics and design (ISLPED'04)*. ACM, New York, NY, 108–113.

KEUNG, H. N. L. C., DYSON, J. R. D., JARVIS, S. A., AND NUDD, G. R. 2004. Self-adaptive and self-optimising resource monitoring for dynamic grid environments. In *Proceedings of the International Workshop on Database and Expert Systems Applications*. 689–693.

KHAN, O. AND KUNDU, S. 2008. A framework for predictive dynamic temperature management of microprocessor systems. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'08)*. IEEE, 258–263.

KIM, T.-H., PERSAUD, R., AND KIM, C. H. 2008. Silicon odometer: An on-chip reliability monitor for measuring frequency degradation ofdigital circuits. *IEEE J. Solid-State Circuits 43*, 4, 874–880.

KONG, J., CHUNG, S. W., AND SKADRON, K. 2012. Recent thermal management techniques for microprocessors. *ACM Comput. Surv. 44*, 3 13:1–13:42.

KRAMER, D., BUCHTY, R., AND KARL, W. 2011. A light-weight approach for online state classification of self-organizing parallel systems. In *Proceedings of the International Conference on Architecture of Computing Systems Workshops*. 183–194.

KYUNG, H.-M., PARK, G.-H., KWAK, J. W., JEONG, W., KIM, T.-J., AND PARK, S.-B. 2007. Performance monitor unit design for an axi-based multi-core soc platform. In *Proceedings of the ACM Symposium on Applied Computing (SAC'07)*. ACM, New York, NY, 1565–1572.

LEATHERMAN, R. 1997. *On-Chip Instrumentation Approach to System-on-Chip Development*. First Silicon Solutions.

LEATHERMAN, R. AND STOLLON, N. 2005. An embedded debugging architecture for socs. *IEEE Potentials 24*, 1, 12–16.

LONG, J., MEMIK, S. O., MEMIK, G., AND MUKHERJEE, R. 2008. Thermal monitoring mechanisms for chip multiprocessors. *ACM Trans. Archit. Code Optim. 5*, 9:1–9:33.

LU, C., WANG, X., AND KOUTSOUKOS, X. 2005. Feedback utilization control in distributed realtime systems with end-to-end tasks. *IEEE Trans. Parallel Distrib. Syst. 16*, 550–561.

LUK, C.-K., COHN, R., MUTH, R., PATIL, H., KLAUSER, A., LOWNEY, G., WALLACE, S., REDDI, V. J., AND HAZELWOOD, K. 2005. Pin: building customized program analysis tools with dynamic instrumentation. *SIGPLAN Not. 40*, 190–200.

LUKOVIC, S. AND CHRISTIANOS, N. 2010. Hierarchical multi-agent protection system for noc based mpsocs. In *Proceedings of the International Workshop on Security and Dependability for Resource Constrained Embedded Systems (S&D4RCES'10)*. ACM, New York, NY, 6:1–6:7.

MANGANO, D. AND STRANO, G. 2010. Enabling dynamic and programmable qos in socs. In *Proceedings of the 3rd International Workshop on Network on Chip Architectures (NoCArc'10)*. ACM, New York, NY, 17–22.

MARESCAUX, T. AND CORPORAAL, H. 2007. Introducing the supergt network-on-chip: Supergt qos: more than just gt. In *Proceedings of the 44th Annual Design Automation Conference (DAC'07)*. ACM, New York, NY, 116–121.

MARTÍNEZ, J. F., RENAU, J., HUANG, M. C., PRVULOVIC, M., AND TORRELLAS, J. 2002. Cherry: Checkpointed early resource recycling in out-of-order microprocessors. In *Proceedings of the 35th Annual ACM/IEEE International Symposium on Microarchitecture*. IEEE, 3–14.

MASSIE, M. L., CHUN, B. N., AND CULLER, D. E. 2004. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Comput. 30*, 5–6, 817–840.

MCGOWEN, R., POIRIER, C. A., BOSTAK, C., IGNOWSKI, J., MILLICAN, M., PARKS, W. H., AND NAFFZIGER, S. 2006. Power and temperature control on a 90-nm itanium family processor. *IEEE J. Solid-State Circuits 41*, 1, 229–237.

MENG, K., JOSEPH, R., DICK, R. P., AND SHANG, L. 2008. Multi-optimization power management for chip multiprocessors. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques (PACT'08)*. ACM, New York, NY, 177–186.

MERKEL, A. AND BELLOSA, F. 2008. Task activity vectors: a new metric for temperature-aware scheduling. In *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems*. 1–12.

MILENKOVIĆ, M., MILENKOVIĆ, A., AND JOVANOV, E. 2005. Hardware support for code integrity in embedded processors. In *Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES'05)*. ACM, New York, NY, 55–65.

MOCHOCKI, B. C., LAHIRI, K., CADAMBI, S., AND HU, X. S. 2006. Signature-based workload estimation for mobile 3d graphics. In *Proceedings of the 43rd Annual Design Automation Conference (DAC'06)*. ACM, New York, NY, 592–597.

MODARRESSI, M., SARBAZI-AZAD, H., AND TAVAKKOL, A. 2010. An efficient dynamically reconfigurable on-chip network architecture. In *Proceedings of the 47th Design Automation Conference (DAC'10)*. ACM, New York, NY, 166–169.

MOSHOVOS, A., MEMIK, G., CHOUDHARY, A., AND FALSAFI, B. 2001. JETTY: Filtering snoops for reduced energy consumption in SMP servers. In *Proceedings of the 7th International Symposium on High-Performance Computer Architecture (HPCA'01)*. IEEE, 85.

MOUSA, H. AND KRINTZ, C. 2005. HPS: Hybrid profiling support. In *Proceedings of the 14th International Conference on Parallel Architectures and Compilation Techniques (PACT'05)*. IEEE, 38–50.

MUNAWAR, M. AND WARD, P. 2006. Adaptive monitoring in enterprise software systems. In *Proceedings of the SIGMETRICS Workshop on Tackling Computer Systems Problems with Machine Learning Techniques*.

MURALI, S., THEOCHARIDES, T., VIJAYKRISHNAN, N., IRWIN, M. J., BENINI, L., AND MICHELI, G. D. 2005. Analysis of error recovery schemes for networks on chips. *IEEE Des. Test 22*, 434–442.

NAKKA, N., KALBARCZYK, Z., IYER, R. K., AND XU, J. 2004. An architectural framework for providing reliability and security support. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'04)*. IEEE, 585.

NETHERCOTE, N. AND SEWARD, J. 2007. Valgrind: A framework for heavyweight dynamic binary instrumentation. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'07)*. 89–100.

NOLLET, V., MARESCAUX, T., VERKEST, D., MIGNOLET, J.-Y., AND VERNALDE, S. 2004. Operating-system controlled network on chip. In *Proceedings of the 41st Annual Design Automation Conference (DAC'04)*. ACM, New York, NY, 256–259.

NOWROZ, A. N., COCHRAN, R., AND REDA, S. 2010. Thermal monitoring of real processors: techniques for sensor allocation and full characterization. In *Proceedings of the 47th Design Automation Conference (DAC'10)*. ACM, New York, NY, 56–61.

NVIDIA TEGRA 3. 2012. www.nvidia.com/object/tegra-superchip.html.

PAPADOGIANNAKIS, A., KAPRAVELOS, A., POLYCHRONAKIS, M., MARKATOS, E. P., AND CIUFFOLETTI, A. 2006. Passive end-to-end packet loss estimation for grid traffic monitoring. In *Proceedings of the CoreGRID Integration Workshop*.

PARK, S.-B. AND MITRA, S. 2010. Post-silicon bug localization for processors using IFRA. *Comm. ACM 53*, 2, 106–113.

PASTRNAK, M., H. N. DE WITH, P., AND MEERBERGEN, J. V. 2006. Qos concept for scalable MPEG-4 video object decoding on multimedia (NoC) chips. *IEEE Trans. Consum. Electron. 52*, 1418–1426.

POUCHARD, L., POOLE, S., LOTHIAN, J., AND GROER, C. 2009. Open standards for sensor information processing. Tech. rep. ORNL-TM-2009-145, Oak Ridge National Laboratury.

POURSHAGHAGHI, H. R. AND DE GYVEZ, J. P. 2009. Dynamic voltage scaling based on supply current tracking using fuzzy logic controller. In *Proceedings of the 16th IEEE International Conference on Electronics, Circuits, and Systems*. 779–782.

PRVULOVIC, M., ZHANG, Z., AND TORRELLAS, J. 2002. Revive: cost-effective architectural support for rollback recovery in shared-memory multiprocessors. In *Proceedings of the 29th Annual International Symposium on Computer Architecture (ISCA'02)*. IEEE, 111–122.

QURESHI, M. K. AND PATT, Y. N. 2006. 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*. IEEE, 423–432.

RADETZKI, M., FENG, C., ZHAO, X., AND JANTSCH, A. 2012. Methods for fault tolerance in networks on chip. *ACM Comput. Surv. 44*, 1–35.

RAGEL, R. G., PARAMESWARAN, S., AND KIA, S. M. 2005. Micro embedded monitoring for security in application specific instruction-set processors. In *Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES'05)*. ACM, New York, NY, 304–314.

SAAB, C. B., BONNAIRE, X., AND FOLLIOT, B. 2002. Phoenix: A self adaptable monitoring platform for cluster management. *Cluster Comput. 5*, 1, 75–85.

SAEN, M., OSADA, K., ET AL. 2007. Embedded SoC resource manager to control temperature and data bandwidth. In *Proceedings of the IEEE International Solid-State Circuits Conference (Digest of Technical Papers)*. 296–604.

SALAPURA, V., GANESAN, K., GARA, A., GSCHWIND, M., SEXTON, J. C., AND WALKUP, R. E. 2008. Next-generation performance counters: Towards monitoring over thousand concurrent events. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'08)*. IEEE, 139–146.

SALFNER, F., LENK, M., AND MALEK, M. 2010. A survey of online failure prediction methods. *ACM Comput. Surv. 42*, 3, 10:1–10:42.

SASTRY, S. S., BODÍK, R., AND SMITH, J. E. 2001. Rapid profiling via stratified sampling. *SIGARCH Comput. Archit. News 29*, 278–289.

SCHMECK, H., M̈ULLER-SCHLOER, C., ÇAKAR, E., MNIF, M., AND RICHTER, U. 2010. Adaptivity and self-organization in organic computing systems. *ACM Trans. Auton. Adapt. Syst. 5*, 3, 10:1–10:32.

SHARIFI, A., ZHAO, H., AND KANDEMIR, M. 2010. Feedback control for providing QoS in NoC-based multicores. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE'10)*. European Design and Automation Association, 1384–1389.

SHI, W., LEE, H.-H. S., 'FALK, L., AND GHOSH, M. 2006. An integrated framework for dependable and revivable architectures using multicore processors. *SIGARCH Comput. Archit. News 34*, 2, 102–113.

SHU, L. AND LI, X. 2010. Temperature-aware energy minimization technique through dynamic voltage frequency scaling for embedded systems. In *Proceedings of the 2nd International Conference on Education Technology and Comput.* Vol. 2. V2–515 –V2–519.

SINGH, P., KARL, E., BLAAUW, D., AND SYLVESTER, D. 2012. Compact degradation sensors for monitoring nbti and oxide degradation. *IEEE Trans. VLSI Syst. 20*, 9, 1645–1655.

SKADRON, K., ABDELZAHER, T., AND STAN, M. R. 2002. Control-theoretic techniques and thermal-RC modeling for accurate and localized dynamic thermal management. In *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA'02)*. 17–28.

SKADRON, K., STAN, M. R., SANKARANARAYANAN, K., HUANG, W., VELUSAMY, S., AND TARJAN, D. 2004. Temperature-aware microarchitecture: Modeling and implementation. *ACM Trans. Architect. Code Optim. 1*, 1, 94–125.

SMOLENS, J. C., GOLD, B.T., KIM, J., FALSAFI, B., HOE, J. C., AND NOWATZYK, A. G. 2004. Fingerprinting: bounding soft-error detection latency and bandwidth. In *Proceedings of the 11th International Conference of Architectural Support for Programming languages and opereating Systems.* ACM New York, NY, 224–234.

SPRUNT, B. 2002. The basics of performance-monitoring hardware. *IEEE Micro 22*, 64–71.

STOICA, A. AND ANDREI, R. 2007. Adaptive and evolvable hardware - a multifaceted analysis. In *Proceedings of the 2nd NASA/ESA Conference on Adaptive Hardware and Systems (AHS'07)*. IEEE, 486–498.

SUH, G. E., LEE, J. W., ZHANG, D., AND DEVADAS, S. 2004. Secure program execution via dynamic information flow tracking. *SIGARCH Comput. Archit. News 32*, 85–96.

SWEENEY, P. F., HAUSWIRTH, M., CAHOON, B., CHENG, P., DIWAN, A., GROVE, D., AND HIND, M. 2004. Using hardware performance monitors to understand the behavior of java applications. In *Proceedings of the 3rd*

Conference on Virtual Machine Research and Technology Symposium, USENIX Association, Berkeley, CA, 5.

Sylvester, D., Blaauw, D., and Karl, E. 2006. ElastIC: An adaptive self-healing architecture for unpredictable silicon. *IEEE Des. Test 23*, 484–490.

Tedesco, L., Clermidy, F., and Moraes, F. 2009. 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 (CODES+ISSS'09.)* ACM, New York, NY, 109–118.

Tilera Corporation. 2009. The TILE64 chip. www.tilera.com.

Tschanz, J., Bowman, K., Lu, S.-L., Aseron, P., Khellah, M., Raychowdhury, A., Geuskens, B., Tokunaga, C., Wilkerson, C., Karnik, T., and De, V. 2010. A 45nm resilient and adaptive microprocessor core for dynamic variation tolerance. In *Proceedings of the IEEE International Solid-State Circuits Conference (Digest of Technical Papers) (ISSCC)*. 282–283.

Tschanz, J., Bowman, K., Wilkerson, C., Lu, S.-L., and Karnik, T. 2009. Resilient circuits: enabling energy-efficient performance and reliability. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD'09)*. ACM, New York, NY, 71–73.

van den Brand, J.W., Ciordas, C., Goossens, K., and Basten, T. 2007. Congestion-controlled best-effort communication for networks-on-chip. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE'07)*. EDA Consortium, San Jose, CA, 948–953.

Vermeulen, B. and Goossens, K. 2010. Debugging multi-core systems on chip. In *Multi-Core Embedded Systems*, G. Kornaros, Ed., CRC Press, Taylor & Francis Group, 153–198.

Wang, P.-H., Yang, C.-L., Chen, Y.-M., and Cheng, Y.-J. 2011. Power gating strategies on GPUs. *ACM Trans. Archit. Code Optim. 8*, 3, 13:1–13:25.

Wang, Y., Ma, K., and Wang, X. 2009. Temperature-constrained power control for chip multiprocessors with online model estimation. In *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA'09)*. ACM, New York, NY, 314–324.

Weissel, A. and Bellosa, F. 2004. Dynamic thermal management for distributed systems. In *Proceedings of the 1st Workshop on Temperature-Aware Computer Systems (TACS'04)*.

Wells, P. M., Chakraborty, K., and Sohi, G. S. 2009. Mixed-mode multicore reliability. In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'09)*. ACM, New York, NY, 169–180.

Wen, C.-N., Chou, S.-H., Chen, C.-C., and Chen, T.-F. 2012. NUDA: A non-uniform debugging architecture and nonintrusive race detection for many-core systems. *IEEE Trans. Comput. 61,* 2, 199–212.

Xilinx Chipscope. www.xilinx.com/tools/cspro.htm.

Xu, M., Bodik, R., and Hill, M. D. 2003. A flight data recorder for enabling full-system multiprocessor deterministic replay. In *Proceedings of the 30th Annual International Symposium on Computer Architecture (ISCA'03)*. ACM, New York, NY, 122–135.

Yao, J., Liu, X., Yuan, M., and Gu, Z. 2008. 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. (CODES+ISSS'08)*. ACM, New York, NY, 85–90.

Zhang, X., Wang, Z., Gloy, N., Chen, J. B., and Smith, M. D. 1997. System support for automatic profiling and optimization. *SIGOPS Operating Syst. Rev. 31*, 5, 15–26.

Zhang, Y. and Srivastava, A. 2011. Accurate temperature estimation using noisy thermal sensors for Gaussian and non-Gaussian cases. *IEEE Trans. VLSI Syst. 19*, 9, 1617–1626.

Zhao, J., Madduri, S., Vadlamani, R., Burleson, W., and Tessier, R. 2011. A dedicated monitoring infrastructure for multicore processors. *IEEE Trans VLSI Syst. 19*, 6, 1011–1022.

Zhou, P., Teodorescu, R., and Zhou, Y. 2007. Hard: Hardware-assisted lockset-based race detection. In *Proceedings of the IEEE 13th International Symposium on High Performance Computer Architecture*. IEEE, 121–132.

Received May 2012; revised October 2012; accepted December 2012