POLITECNICO DI MILANO
Dipartimento di Elettronica, Informazione e Bioingegneria
DOTTORATO DI RICERCA IN INGEGNERIA DELL'INFORMAZIONE

THERMAL AND ENERGY MANAGEMENT TECHNIQUES FOR MULTI-CORE AND MANY-CORE SYSTEMS

Doctoral Dissertation of:
Federico Terraneo

Advisor:
Prof. William Fornaciari

Tutor:
Prof. Francesco Amigoni

The Chair of the Doctoral Program:
Prof. Carlo Fiorini

2014 – XXVII

Abstract

THE current trend to increase the performance of computing systems is to shift from instruction level parallelism to task level parallelism, with multi core architectures being mainstream, and many-core systems already used for specialized tasks and expected in the near future to become widely deployed. This, coupled with the continuous technology scaling has exacerbated a reliability issue caused by the power consumption of heavy loaded functional units which cause severe temperature gradients in the silicon die, the so called "hot spots", as well as thermal cycles caused by highly varying processor workloads. These problems will all lead to the issue of dark silicon, where the limited ability to dissipate heat reduces the portion of a chip that can be turned on as technology scaling progresses, significantly reducing the achievable performance despite the progress achieved thanks to Moore's Law scaling continues.

It is thus expected that in the near future, with the introduction of new techniques such as 3D stacking to further increase system performance, these issues will become increasingly significant, and will require much more elaborate thermal control techniques than the ones currently employed.

The aim of this thesis is twofold. First, novel thermal and power control strategies have been developed to face the needs of future multicore architectures, based on the application of control theory. Second, a state of the art simulation framework has been developed that is able to perform the assessment of dynamic thermal management and power-performance policies comprising a cycle accurate simulator and accurate sensor and actuator models, being thus able to test policies in a realistic setting.

Experimental results using the proposed simulation framework and standard benchmark suites such as MiBench [42] have been used to evaluate the proposed policies, and assessing the resulted performance improvements beyond the state of the art.

Sommario

La strada attualmente perseguita per incrementare la potenza di calcolo dei sistemi di computazione consiste nel passare dall'incremento del parallelismo a livello di istruzione all'introduzione di più flussi di esecuzione paralleli, con architetture multiprocessore già affermate e diffuse, mentre architetture con un numero elevato di processori, dette *many-core* già utilizzate come acceleratori per applicazioni specifiche, quali il calcolo scientifico. Ci si aspetta che queste architetture diventeranno sempre più affermate anche per applicazioni generali nel breve futuro. Questo fatto, unito alla progressiva riduzione delle dimensioni dei transistor sta causando problemi di affidabilità in quanto la densità di potenza delle architetture attuali, unita alla disuniforme distribuzione del consumo tra i vari processori e le loro unità funzionali genera dei punti caldi e cicli termici causati da dei carichi di CPU variabili. Queste ed altre considerazioni portano al cosiddetto problema del *dark silicon*, dove la ridotta capacità di dissipare il calore generato forza a ridurre il numero di transistor che possono essere tenuti operativi alla massima frequenza ad una frazione che si riduce con il progredire dell *scaling* tecnologico. Questo porta ad una riduzione dell'incremento di prestazioni nel passaggio da una generazione alla successiva, nonstante il suddetto scaling secondo la legge di Moore continui.

Inoltre, ci si aspetta che l'introduzione di tecnologie innovative per incrementare la capacità di calcolo quali il *3D stacking*, che consiste nel costruire un circuito integrato con più strati di silicio dotati di componenti attivi, causino un ulteriore incremento della densità di potenza, richiedendo

quindi tecniche molto più elaborate di quelle utilizzate attualente in produzione per controllare la dissiapzione di potenza all'interno dei futuri sistemi multiprocessore.

Lo scopo della presente tesi di dottorato è duplice. Innanzitutto, sono state sviluppate delle innovative tecniche per controllare la temperatura dei futuri sistemi multiprocessore, nonchè la potenza dissipata, basandosi sulla teoria del controllo. Inoltre, un flusso di simulazione è stato sviluppato, in grado di verificare le prestazioni delle suddette politiche di controllo. Il simulatore può simulare una architettura multiprocessore a livello del singolo ciclo macchina ed è dotato di accurati modelli per quanto riguarda sensori e attuatori, essendo quindi in grado di simulare le politiche in un ambiente realistico, tenendo conto delle non idealità degli attuatori e dei sensori, nonchè dei costi di attuazione.

Risultati sperimentali ottenuti grazie al flusso di simulazione e *benchmark* standard sono stati utilizzati per validare le politiche presentate, validando gli incrementi di prestazioni ottenuti rispetto allo stato dell'arte.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

TODAY the semiconductor industry is facing increasing problems to continue delivering performance improvements at a pace that is now expected by its user base, as well as the general public. This is not yet caused by problems in achieving feature size reductions, but by the side effects and nonidealities caused by said scaling.

Although the diminishing returns of further optimizing uniprocessor architectures has been successfully overcome through the move to multi-core designs – at least for parallelizable workloads –, the failure of Dennard scaling [28] in deep nanometer architectures results in an ever worsening power density increase, eventually leading to the dark silicon problem [37, 84], where power and thermal constraints limit the number of transistors that can be switched at the maximum clock speed to an ever decreasing fraction. This problem, if not solved, could be a major roadblock in the evolutionary path from multi-core to many-core architectures.

For this reason, high-performance multi-core designs are increasingly power limited. Thus, microarchitectural improvements and run-time policies that can increase the power efficiency of an MPSoC are particularly sought after.

However, power efficiency alone is not enough to mitigate the dark silicon issue, as one of the major problems caused the power density increase is the need to effectively dissipate the generated heat away from the silicon die to prevent immediate failures as well as reliability issues caused by high operating temperatures.

In such a *scenario*, thermal management is a fundamental design challenge that needs to be explicitly taken into account, as there can be cases where power-performance and thermal optimizations are in conflict. For example, a power optimization strategy that concentrates computation in a small area of the chip in order to use power gating could well generate an hot spot [53].

Thus, to effectively turn the increasing transistor count made possible by Moore scaling into a performance increase both thermal and power management techniques are required.

Another important consideration is the high variability in the load experienced by multiprocessor system on chips (MPSoCs), caused by the potentially very different activities performed by the various cores.

In this perspective, effective *dynamic* thermal management solutions that can push the cores to their maximum performance subject to the constraint imposed by the need to remain within safe operating temperatures is a key aspect to achieve the best utilization of the computational capabilities of current, and especially future MPSoCs.

The work presented in this doctoral thesis is a contribution to this large and not yet fully explored research topic.

1.1 Contributions

The main contributions of this thesis can be divided in two main areas: the development of an extensible simulation flow able to produce an accurate view of the transient thermal behavior of current and future MPSoCs, including 3D die stacked ones, and the development of dynamic thermal-management and power-performance policies that can counteract the negative effects of the increasing power densities. In detail, the contributions can be summarized as follows:

- The development of accurate models for multiple DVFS actuators in a cycle-accurate instruction-set simulator (GEM5 [16]), including accounting for the resynchronization overhead between different clock domains. This allows to simulate the DVFS actuator used by dynamic thermal management policies without losing the cycle-accuracy of the simulation, and taking into account their overhead and nonidealities.

- The development of a novel flexible thermal simulator based on object-oriented and component-based modeling, in the Modelica language. Such a thermal simulator is more open to extensibility than the current state of the art, such as HotSpot [46], as it allows to easily make changes in the chip structure and thermal dissipation path from the chip to the heatsink, like for example simulating a 3D stacked MPSoC.

- The design of an innovative thermal control policy for 3D MPSoCs using event-based control theory. Such a policy is centered on a hardware/software partitioning of the controller implementation, and can provide the fast reaction time needed to counteract abrupt temperature increases found in 3D die stacked chips and the low overhead of a policy implemented entirely in hardware coupled with the flexibility of a policy implemented in software.

- The design of an effective power-performance policy for reducing the power consumption of NoC routers by using DFS to reduce their clock frequency when the network traffic is low. This policy is based on a controller tuned starting from a model of the frequency to contention relation of a NoC router, and provides a formal proof for what concerns the stability of the closed loop system.

1.2 Dissertation organization

This PhD thesis is organized as follows

- *Chapter 2* provides a background on simulation frameworks for MPSoCs, as well as thermal management and power-performance problems, with a focus on the future challenges.

- *Chapter 3* introduces the cycle-accurate simulation flow developed as part of this PhD thesis to evaluate the thermal behavior of modern MPSoCs, and test dynamic thermal management as well as power-performance policies.

- *Chapter 4* presents the developed component-based thermal simulator that is proposed as a solution to model the thermal dynamics of current and future generation MPSoCs.

- *Chapter 5* details the proposed thermal management policy. This policy, based on event-based control theory allows to handle the increased

power density of future 3D MPSoCs coupled with the high workload variability of a many-core platform.

- *Chapter 6* outlines the proposed power-performance policy to optimize the power consumption of NoC routers using DFS to reduce their frequency when the network load is low.
- *Chapter 7* concludes the thesis, and outlines future research directions.

CHAPTER 2

The thermal and power management problem in MPSoCs

Digital? Every idiot can count to one.

Bob Widlar

THIS chapter presents an overview of the problems related to the design of MPSoCs, with a focus on thermal management and control, power-performance optimizations and on the development of simulation environments to assess the performance of novel solutions prior to implementing them in silicon. This motivates the research in these large and still not fully explored fields that have the potential to further improve the performance and power efficiency of computing architectures, finding solutions to the various roadblocks that are being found as technology scaling progresses.

2.1 Thermal management

High performance computing systems have historically always been limited by thermal considerations [48], and on the thermal management problem

in computing systems a large research *corpus* has been developed, as for example testified by [48, 53, 91, 101].

However, the semiconductor industry is currently facing increasing problems related to the power density increase at every technology node, causing the well known dark silicon problem [37, 84]. As such, solutions that help maximizing the performance of an MPSoC subject to temperature and power constraints, as well as explore the power-performance and thermal-performance tradeoffs are particularly sought after. One recent example is the sprint computing [75] approach, which consists in relying on the slow thermal dynamics to transiently exceed a chip thermal design power (TDP) without reaching unsafe temperatures. This has already found adoption in commercial applications, in the form of the Turbo boost 2.0 [77] which is a hardware controller in the recent Intel processors capable of operating the cores above their TDP if the temperature state of the chip allows it.

There are two main reasons why the temperature of an MPSoC, or in general an integrated circuit has to be kept under control. The first and most obvious reason is to prevent immediate failures such as thermal runaway conditions [91]. This already sets an upper limit to the temperature that an integrated circuit can safely reach. However, many phenomena affecting the reliability of an integrated circuit, such as hot carrier injection (HCI) [43], electromigration [32], negative bias temperature instability (NBTI) [59], as well as thermal fatigue on the solder joints [90] depend on temperature. It is thus often desirable, or even necessary, to further lower the chip temperature at a value computed from reliability considerations.

A first solution to the current trend of increasing power densities in MPSoCs could be the development of better heat dissipation methods to keep the chip temperature at safe levels despite the increasing power and power density. This is essentially what has been done in the past, considering that early single chip microprocessors, such as the Intel 8086 operated without special heat dissipation considerations. Later on, heatsinks to improve thermal dissipation through natural convection became necessary, and finally forced air cooling through fans became mandatory. Further improvements to withstand greater heat fluxes have been proposed in the literature, such as liquid cooling, both of the chip package or directly at the chip level through microchannels on the silicon die [101], the use of active devices such as thermoelectric coolers (TECs) over the entire chip surface [79] or micro TECs directly placed atop of an hot spot [8], or the use of phase changing materials (PCM) [93]. Although this is a promising research direction, these solutions are currently not used for mainstream applications due to issues in implementing them in a cost effective way. Moreover, active so-

lutions such as the one based on TECs require a non negligible amount of power to operate, thereby reducing energy efficiency.

However, TDP is a very conservative constraint, as an MPSoC power consumption is not constant at the maximum value. Even considering a situation where all the cores in the MPSoC are fully loaded, the power consumption depends on the CPI [10] and thus depends on factors such as the executed code and cache misses. Moreover, the cores are often put in a low power state by the operating system during idle periods or when waiting for I/O operations or user input.

The highly variable power consumption of MPSoCs motivates the introduction of dynamic thermal management (DTM) policies. DTM policies work by sensing the chip temperature at run-time, and rely on the availability of actuators such as increasing fan speeds to improve the heat dissipation, or reducing core frequencies to reduce the power consumption if required. Their strength lies in their capability to maximize the performance of a given architecture limited by a heat dissipation budget, across varying loads.

One of the first DTM policy proposed is the stop and go [21], which halts the processor clock if the operating temperature is higher than a given threshold, and restores normal operation only after it has cooled down. Although this policy yields a considerable performance penalty compared to other techniques [29], it is often used as a secondary policy that acts only in critical conditions.

An improved policy [80] uses a control loop to modulate the instruction fetching of a processor, reducing on purpose the fetch rate if temperature is too high. Contrary to the stop and go policy that uses a binary actuator, this policy can adapt the performance impact to the required control action to keep the temperature under control.

Dynamic voltage and frequency scaling (DVFS) [78], although first introduced to explore the power-performance tradeoff is an effective actuator also for DTM due to its quadratic effect on power consumption, and is being employed as the knob for many DTM policies.

One of the first works making use of DVFS in a DTM policy in multicore processors is [29], which proposes a proportional-integral (PI) controller sensing the core temperature and actuating on its frequency. The scheme is distributed, meaning that each core in the MPSoC has its own controller, and requires per-core DVFS. While the proposed solution achieves remarkable performance, the overhead of the proposed controller is not taken into account in the quoted work. This scheme is the most similar in nature to the DTM policy presented in Chapter 5 of this thesis, however the scheme

here presented makes use of event-based control theory thereby presenting a significantly lower overhead, to the point of allowing a software implementation of the policy thereby allowing for greater flexibility.

The heat and run approach [73] proposes a DTM policy based on task migration for MPSoCs. The policy works by moving applications that perform "heavy" computation away from "hot" cores. The policy yields a performance improvement compared to stop and go, but suffers from two major drawbacks. First, task migration cannot be performed at a sub-millisecond timescale without incurring in an excessive overhead due to cache lines invalidation. In addition, this solution exacerbates both temporal and spatial thermal gradients, which degrade the overall system reliability [33].

In [50], the authors propose a solution based on convex optimization, that assumes the power to be constant and optimizes the workload under steady state temperature constraints. The authors however admit that the solution needs to be complemented with a DTM technique, to account for varying power consumption and ambient temperature.

A Linear Quadratic Regulator (LQR) is presented in [99] to address thermal balancing of a multicore chip based on a model of the system, workload requirements – to be obtained from the OS scheduler – and temperature measurements. The proposed policy is meant to be operated every 100ms, thus being aimed at controlling only the slow thermal dynamics, and its centralized nature, together with the need to interact with the OS scheduler, make it difficult to operate at finer timescales, a matter that will be further discussed in Chapter 5.

Other proposals focus on predictive approaches; a notable example is [98], where a Model Predictive Control (MPC) solution is presented to address thermal management in multi-cores with the objective of smoothing the control actions. The proposed scheme is centralized, and thus its scalability is limited as the number of cores increases. Furthermore, it relies on the strong assumption of knowing the future chip workload. Analogously, [10] describes a decentralized MPC solution to the thermal management problem, thus solving scalability issues of centralized solutions. The proposal still requires for an accurate workload information. The control rule in the quoted work is particularly lightweight for an approach based on MPC, but it has to be applied at a fixed rate, and is therefore limited by the tradeoff between limiting the number of events and achieving good thermal control.

Thermal management in MPSoCs is thus a well-studied research topic, where many solutions have already been proposed. Having presented an overview of the literature related to the thermal management problem in

MPSoCs, it is worth having a look at the future challenges posed by the expected directions in which the semiconductor industry will move.

One of this breakthrough changes is surely the move to 3D die-stacked architectures [19] which consists in building integrated circuits with multiple layers of active components, connected through high-speed methods such as through-silicon vias. This construction method yields the possibility to have more dense interconnections, reducing wire delays and power consumption, but will result in even greater power densities. As such, the increase in the amplitude of the temperature transients induced by fast thermal dynamics, although already present in 2D chips, will impose more and more stringent constraints in the reaction time required by thermal policies, to limit the temperature of hot spots to safe levels. As Chapter 5 of this thesis will explain, DTM policies will need to be operated at the millisecond, or even sub-millisecond timescale to effectively control the die temperature, thereby imposing severe limitations to the policy design and implementation.

2.2 Power-performance tradeoffs

The exploration and optimization of the power-performance tradeoff in MPSoC design is a vast research topic that spans from core microarchitectural design, interconnection design, whether bus-based or NoC-based, memory hierarchy and cache optimizations, as well as the design of run-time policies. It would thus not be possible to give a full overview of all the power-performance techniques that have been proposed in computing systems. Instead, this section will focus on the power-performance tradeoff in NoC design, with an emphasis on run-time policies making use of dynamic frequency scaling (DFS) to take advantage of the varying NoC load to improve power efficiency.

While NoCs are considered the future solution for the interconnection problems in the transition from multi-core to many-cores, the resource constrained nature of an on-chip network imposes the need to optimize designs mainly in two directions: performance increase and power reduction.

Many proposals rely on microarchitectural modifications to improve performance and reduce power. In this perspective, the use of virtual channels and dynamic traffic distribution [41] allows to reduce contention, improve throughput and provide fault-tolerance. These solutions are definitely useful, but techniques that dynamically adapt the NoC frequency to the actual load allow to further improve power efficiency.

[65] proposes a buffer-less routing approach that leads to lower power

consumption without significantly sacrificing the NoC performance. However, such buffer-less schemes delivers reasonable performance only when the injection rate into the network is low. Being a design-time optimization, this solution is very good for power optimizations, but is limited to solutions where performance is less significant than power. In this perspective, DFS based methodologies adapt to the load in the network allowing for more flexibility for the power-performance optimization.

[64] discussed a fine-grained frequency tuning scheme for NoC routers to optimally manage the power-performance trade-off. In particular the methodology exploits signaling between routers to collect critical information to steer frequency. Moreover, the work allows for a run-time VC reconfiguration to aggressively save power. However, the proposed solution does not model the relations between routers' frequencies and real performance and power measures. To this extent the proposed solution cannot be easily improved, since it represents a fixed heuristic.

The work in [15] leverages the traffic unbalancing within a specific NoC topology to exploit the classical technique of DVFS to minimize the power consumption coupled with *ad-hoc* routing algorithms. A power minimization linear programming model has been proposed to find a routing that minimizes the power consumption while satisfying the traffic demands and meeting the link capacity constraint. The solution relies on a mathematical formulation that must be solved at design time considering a static even in average behavior for the system as a whole.

Different proposals focus on the queuing theoretical framework to model on-chip networks. [67] presented analytical model that focuses on QoS assurance. However, it assumes that the NoC has an underlying synchronous behavior with constant service time routers, thus it is not suitable for optimization using DFS actuators. Moreover, it assumes infinite buffers, and taking into account the finite nature of NoC buffers would greatly complicate the model. An analytical queuing theoretical approach to model NoCs accounting for finite buffers has been presented in [68]. The solution exploits the classic queuing theory, while the router serving time model has been derived from real data. However, the methodology relies on exponential distribution for flit arriving times that cannot in general be guaranteed in NoCs [67], and does not account for run-time frequency variations.

The work in [27] proposes an heuristic approach focused on DVFS actuators to mitigate power consumption on the real Intel SCC multi-core. Even if this solution has been tested on a real multi-core, it does not provide an accurate model of the relation between frequency and performance thus it does not allow to exploit the solution for further improvements.

[69] proposes a design methodology for partitioning an NoC architecture into multiple voltage and frequency islands (VFIs) and assigning supply and threshold voltage levels to each VFI. The employed resynchronization scheme is based on FIFO buffers.

The work in [12] presented a complete DVFS scheme for IP unit integration to be employed for NoC-based design. However, this work does not easily allow to model different policies or different topologies as in a cycle accurate simulation framework, since it has a prototyping focus.

2.3 Simulation frameworks

Simulators are powerful research tools for thermal management and power-performance policies. Their advantage is their inherent ability to provide a greater detail of introspection compared to a real hardware architecture. When designing innovative policies it is often required to sense quantities, either microarchitectural, such as the NoC load, or physical, such as the chip temperature or the power consumption of a functional unit. These sensors can be required either by the policy itself to close a control loop, or need to be logged in order to assess the policy effectiveness. In addition, actuators are also required to steer the system behavior. Hardware architectures often lack the required sensors and actuators needed for the assessment of innovative policies, and this motivates the research in simulation frameworks. In addition, the introduction for such hardware sensors and actuators requires accurate pre-silicon analysis, thus simulation frameworks can be a design aid also to explore the design space for such actuators.

Wattch [22] constitutes the first cycle-accurate single-core power-performance simulator. However, the advent of multi-core architectures required simulation toolchains that allow to accurately mimic the behavior of multi-core systems.

The SESC simulator [76] provides cycle-accurate simulation of bus-based multi-core processors, based on the MIPS architecture. However, it does not support Network-on-Chip architectures and does not support for DFS.

The Polaris framework [82] allows for power and area design space exploration for Network-on-Chip architectures without considering a detailed power estimation for both processors and memory hierarchy. Moreover, it does not implement an heterogeneous NoC model to allow for dynamic frequency changes during simulation.

The simulator presented in [57] is meant to simulate large-scale architectures, and exploits parallel simulation on physical hardware with partic-

ular emphasis on the on-chip network. While the framework enables the possibility for power-performance trade-off analysis, it lacks a complete asynchronous on-chip network model, thus it is not possible to explore different GALS configuration for the interconnect as well as the simulation considering dynamic frequency scaling based on high level policies.

The work in [24] presents Sniper, a framework that can simulate multi-cores underpinned by an on-chip network interconnect, supporting per-core DVFS. However frequency scaling support is not present for the NoC model.

The HANDS [26, 103] framework sits on GEM5 and allows to simulate multi-core architectures collecting power-performance thermal and reliability estimates at the same time. Even if this is quite accurate it lacks a complete asynchronous NoC model, thus it is not possible to test different DFS schemes to trade-off power vs. performance. Moreover, the thermal simulation capability is limited to a steady state analysis.

The virtual platform presented in [11] is instead a simulator targeted at the development of thermal policies. However, it does not support NoC-based architectures. Moreover, even if they support the DVFS, the framework lacks for an accurate evaluation of the PLL model as well as the implementation of the DVFS for the on-chip network.

2.4 Summary

This chapter presented a summary of the challenges related to the thermal and power-performance optimizations in MPSoCs. For space limitations, it was decided to limit the scope of this overview to the literature that was felt more relevant to the topics presented in this dissertation. When considered appropriate, an additional background of the literature complementing this overview is inserted in the chapters related to the individual methodologies developed.

CHAPTER 3

A cycle-accurate MPSoC simulator for thermal and power-performance explorations

> One is equal to two for sufficiently large values of one.
>
> Unknown, on the IEEE floating point representation

MULTI-CORE processors emerged as a commonplace solution to deliver increasing processing power to demanding applications ranging from the embedded to the supercomputing market and are currently the mainstream solution for general-purpose computation. In addition, many-core platforms are already employed for specialized tasks in the form of GPGPU or dedicated accelerators, where simpler cores lacking hardware support for modern operating systems such as an MMU are acceptable, although thanks to Moore scaling it is expected that general-purpose many-core platforms will be available in the near future. In this *scenario*, Network-on-Chip (NoC) are widely recognized as the most promising solution to solve the interconnection problem as the number of cores

increases, being more flexible and scalable than the traditional bus architecture. However, NoCs, although very promising for future MPSoCs are not yet mainstream unlike multi-cores. Owing to the limited core count of current general-purpose processors, bus-based architectures are still preferred.

Performing the strategic decision making required to design a successful MPSoC platform in general, and in detail the development of thermal and power-performance policies, requires a rapid prototyping platform to test their performance. This chapter thus introduces the developed simulation framework [86], which is focused on allowing rapid prototyping of run-time thermal and power-performance policies for multi- and many-core architectures underpinned by a NoC interconnection. After a motivation section to underline the importance of simulators as enablers for research in the design of effective policies, this chapter gives an overview of the entire simulation flow presenting the components upon which it is built, and then details the improvements that have been made to the GEM5 [16] cycle-accurate simulator to add the sensors and actuators models required for policy design. The modeling and simulation of a chip thermal dynamics is instead treated in chapter 4.

3.1 Motivation

The need to develop effective thermal and power-performance policies relies on the availability of sensors for physical quantities, such as temperature, or microarchitectural ones, such as the number of flits stored in the input buffer of a NoC router. In addition, actuators also need to be introduced providing the knobs on which to act to close the loop and operate the policy. This highlights three different issues. First, if the selection of the kind of sensors as well as actuators and their placement in the considered platform is decoupled from the policy design, this results in suboptimal solutions. In particular, the possibility to accurately place sensors and actuators in a way that is tightly coupled with the designed policy provides the best performance. Second, the possibility to propose novel sensors and actuators, that are more effective to face the considered issues, whether thermal or power, requires for a great flexibility of the multi-core hardware. Last, thermal policy evaluations impose to account both actuators and sensors overheads, that can severely affect the benefit of the methodology.

Often, the required sensor and actuators are not yet present in current mainstream architectures, which limits the range of policies that can be tested due to the partial observability and reachability of the system. More-

over, for what concerns NoC-related policies, the fact that most current multi-core designs are still bus-based makes it outright impossible to use them for policy assessment.

However, designing a real hardware architecture in order to test a new policy would be prohibitive due to the time and cost required to build a prototype MPSoC. With this possibility out of question for apparent reasons, two solutions exist to assess run-time policies in MPSoCs: FPGA-based prototyping or cycle-accurate simulation.

FPGA-based prototyping incurs several limitations, as sensors for physical quantities like temperature may be difficult to design in an FPGA and require external hardware [5,6], and for what concerns actuators, partitioning the computational logic (as opposed to the I/O ring) into multiple independent voltage islands is not supported in current FPGAs, which would make implementing fine granularity DVFS impossible. In addition the limited amount of hardware resources in current FPGAs may make it hard to implement an entire many-core architecture.

To this extent, considering the complexity of the actual MPSoCs, the need to quickly and flexibly modify the hardware platform eventually with novel sensors and actuators, as well the requirement for a complete observability and controllability of the systems motivates the research in simulation frameworks. In particular, architectural simulators are widely accepted to support strategic decision making on the allocation and management of the hardware monitors and knobs and to test policies to orchestrate thermal-performance and power-performance tradeoffs.

3.2 Design goals

As anticipated, assessing a thermal or power-performance policy in a realistic settings requires models for sensors and actuators. However, real-world sensors and actuators have limitations and nonidealities. To give an example of those nonidealities consider a PLL that is commonly used to dynamically change the frequency of a CPU core or NoC router. PLLs are made by a Voltage Controlled Oscillator (VCO) disciplined using a feedback loop to a lower frequency but more stable reference signal, such as a crystal oscillator. However, the frequency produced by a PLL cannot be changed instantaneously, rather, when the frequency set point is changed the output frequency evolves depending on the oscillator and feedback loop dynamics. However, state of the art simulators neglect this and simulate a frequency change as an instantaneous one. This simplification neglects the overhead of a frequency change, a matter that may be significant while testing a pol-

icy that makes frequent changes to the frequency of a component.

Summarizing, the simulator proposed in this chapter has the following main characteristics

- *Account for Multiple Simultaneous Metrics* - Policy design requires accurate estimates from the underlying microarchitecture to be extracted simultaneously. The proposed flow provides power, thermal and performance estimates at different granularity levels for both cores and the NoC, thus allowing to develop accurate thermal-performance and power-performance policies. The simulation environment tackles all the restrictions on the observability of the metrics, while it is possible to set constraints on the metric extraction to mimic real hardware limitations. For example, the simulator extracts a temperature value per each functional unit of the CPU, while it is possible to assume a single thermal sensor per core located in a specific functional unit to model the actual, limited number of hardware sensors integrated in the real platform.

- *Accurate Model for the Actuators* - The actuators represent the knobs to steer the system to the desired operating point. However, the possibility to employ custom knobs and the availability of accurate models including their overheads represent two critical features. In this perspective, the flow exposes a complete analytical model for DVFS for both cores and NoC routers. The net results is a framework where the simulation of the microarchitecture and the actuators run in a seamless fashion.

- *Dynamics of the System Components* - This simulation flow is designed from the start with the aim of assessing policies that operate at run-time, and thus is not limited to steady-state analysis. It can simulate the system transient behavior, producing traces of the desired metrics, such as power and temperature as a function of simulation time. This allows the policies to operate from sensor data to control the system behavior closing the loop between the policies and simulator, in order to verify their ability to limit overshoots and, in general, undesired transient behaviors.

This is not the first simulation flow proposed for MPSoCs. Different simulation solutions have been proposed to address power-performance and thermal-performance optimizations for single-core and multi-cores. Nevertheless, few of them focused on a comprehensive approach to jointly estimate multiple design dimensions. Moreover, the possibility to accurately

Table 3.1: *State-of-the-art multi-core simulators: features, advantages and drawbacks with focus on fine granularity DVFS.*

		Proposed flow	SST [44]	VirPlat [11]	HANDS [103]	Sniper [24]	Ocin tsim [74]
Tradeoff	Power	✓	✓	✓	✓	✓	✓
	Thermal	✓	✓*	✓	✓*		
DVFS CPU	Func	✓		✓	✓	✓	
	PLL	✓					
	Vreg	✓					
DVFS NoC	Func	✓		✓			✓
	PLL	✓					
	Vreg	✓					
GALS	Func	✓		✓		✓	✓
	Resync	✓					

* = Limited to steady state thermal simulations.

simulate the model of the actuators coupled with the platform microarchitecture as well as the possibility to track the transient behaviors represent two distinguishing features of the proposed simulation flow. The proposed simulation flow is here compared with other state of the art solutions focused on thermal and power-performance tradeoffs to show its benefits and limitations. Each row in Table 3.1 represents a feature to support power-performance and thermal-performance optimizations. The table also shows the level of support for each feature, detailing if models of the actuators have been implemented, or the simulator is limited to a functional simulator without considering the overhead introduced by the actuator nonidealities. For example, the row *DVFS CPU* considers whether the simulator provides only the possibility of changing the core frequency, or supports a detailed model of the PLL and voltage regulator.

Table 3.1 highlights a great effort in the development of power-performance simulators since all the considered frameworks account for power estimates. Moreover, support for DVFS – at least in its functional declination – is stronger for CPU cores than for NoC routers. Last, the state of the art generally lacks accurate simulation frameworks which allows to model both actuators and microarchitectures in a detailed way with support for transient analysis, i.e. Table 3.1 reports that most of the specific features for each main feature are most of the time not supported by the simulators.

3.3 Simulation flow design

From a high-level perspective, the proposed simulation flow is shown in Figure 3.1. The flow can be divided in five different components, the instruction-set simulator, power models, thermal models, policy module

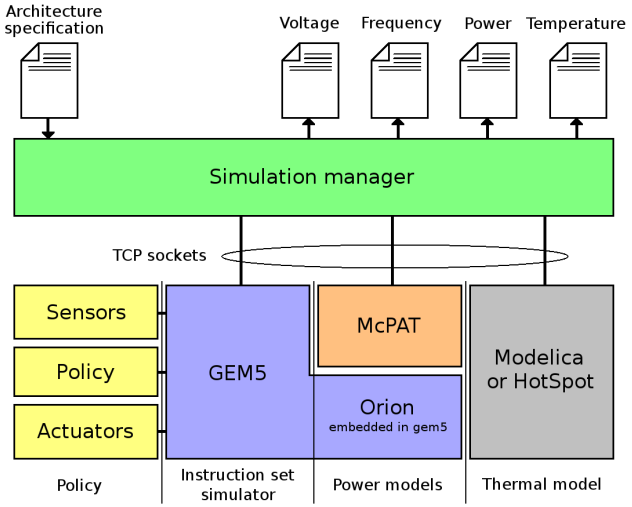

Figure 3.1: *Overview of the simulation flow.*

and simulation manager.

The first component is the GEM5 [16] instruction set simulator, that simulates code execution on the CPU cores as well as the memory accesses through the memory hierarchy, including NoC accesses. It produces statistics on the utilization of the functional units of the cores, as well as the NoC and caches. The simulator is periodically stopped and detailed statistics are dumped and fed to the power models to compute the power consumption.

The GEM5 simulator was extended with support for DVFS for cores and NoC routers, frequency islands and resynchronizers for signals that cross clock domains, as well as sensors to provide introspection capabilities. In addition, the garnet [2] NoC embedded in GEM5 was enhanced to support 3D MPSoCs. To speed up the simulation, GEM5 is used in *syscall emulation* mode, running the applications without simulating the operating system, although a model of the delays incurred due to disk access latencies has been introduced, as this has a non-negligible impact on the power consumption trace and thus the thermal behavior of an MPSoC.

The detailed statistics produced by the instruction set simulator are fed to the power models to compute the power consumption in each of the functional units of the simulated cores, as well as the NoC routers.

McPAT [56] is used to obtain the power consumption of each functional unit of the cores as well as for the caches. The McPAT operation can be

divided in two phases. First a model of the target architecture is built based on a given set of parameters, and then the power consumption is computed based on a given set of statistics on the usage of the individual functional units. In order to adapt McPAT for an on-line simulation framework capable of computing power as a function of time as the simulation progresses, the two phases have been separated. The model building for the target architecture is performed only once at the beginning of the simulation, using the maximum target frequency for consistency. The power computation is instead performed periodically with the statistics and clock frequency of each core coming from GEM5. The power dissipation of the NoC routers is instead computed using the Orion [51] model embedded in GEM5.

The next component of the simulation flow is the thermal simulator, whose purpose is to produce transient thermal traces of the chip starting from a floorplan of each layer in the chip and power consumption data. The simulation flow supports multiple thermal simulators, the default one is is the thermal simulator described in chapter 4 that focuses on a component-oriented modeling approach, being thus flexible to extensions and modifications to the thermal dissipation path, as well as supporting 3D MPSoCs. It is also possible to use the HotSpot thermal simulator [46], configured in transient simulation mode.

The floorplan for the multi-core chip is produced using HotFloorgen [46] starting from area information obtained from McPAT, Cacti [66] and Orion, extended with a tool to combine the core and router floorplan in a flexible way to produce tiled muticore architectures floorplans.

The fourth component is the policy module which can be further divided in three sub-components: the sensors, policy and actuators, which are described in more detail in the next section. For convenience, this module is not a separate process, rather, the policies are implemented as C++ classes in the GEM5 simulator, thereby having full access to the simulation state, a matter that eases the introduction of sensors for microarchitectural metrics, such as the buffer filling in the routers of the NoC. Power and thermal sensors are also available, as the simulation manager communicates these values to GEM5. In addition, to simplify policy design, it is also possible to write policies using the GNU Octave mathematical-oriented programming language, as the Octave interpreter has been embedded in GEM5.

The last component is the simulation manager, which is a custom program developed in C++ to orchestrate the various simulation tools by connecting them in a closed loop pipeline as well as to adapt the format produced by a tool into the one expected by the next one. All the tools have been modified to connect to the manager through TCP sockets, in order

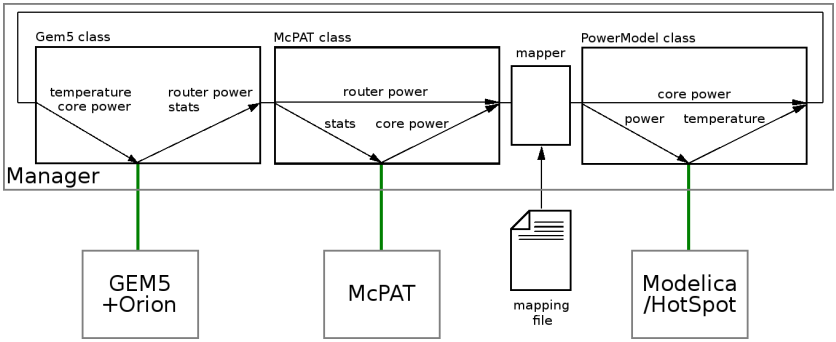

Figure 3.2: *Operation of the simulation manager in interconnecting the individual tools that compose the simulation flow.*

to efficiently exchange information. The simulation manager periodically stops the instruction set simulator, at a configurable rate, to update the power and thermal information, as well as to run the policies. It is also possible to execute the various tools in parallel on different cores, to speed up the simulation. Additionally, the simulation manager produces as output traces of the voltage, frequency, power and temperature as a function of time for the cores, NoC routers and caches.

Figure 3.2 shows the internal architecture of the simulation manager, giving a high-level overview of its operation. The manager is composed of three main C++ classes that handle the interaction with the individual tools through a TCP socket, the first one is the Gem5 class which handles the connection to the GEM5 simulator. The simulator takes as input the temperature and power data to run the policies, and produces statistics for the cores of the MPSoC, while the Orion power model embedded in GEM5 directly produces the power consumption data for the NoC routers. The data is then passed to the McPAT class, which directs the core stats to the McPAT power model, obtaining as a result the power consumption of cores and caches, while the router power is instead simply forwarded. The next component is the mapper which aggregates the power information produced by Orion and McPAT. This mapping is made necessary due to slight differences in the functional unit names between the power models and the thermal simulator. For example, the McPAT power model produces two separate power consumption values for the integer addition/subtraction and multiplication parts of the integer ALU, while in the floorplan of the thermal model there is only one ALU component. The aggregated power consumption data is fed to the PowerModel class that manages data

transfer to the power model, which can be either the Modelica component-based model or HotSpot, and obtains from it the temperature values that are passed back together with the core power to GEM5, closing the simulation loop.

3.4 Implementing DVFS in a cycle-accurate simulator

This section describes the modifications that have been introduced to the GEM5 software architecture to support simulating multiple components such as cores and NoC routers operating at different clock frequencies with no phase relation between them, as well as allowing to modify the frequency of components at run-time.

As GEM5 is an event-driven simulator, individual components are simulated by scheduling events at the active edge of their clocks. In addition, the representation of time in the simulator is discrete with a 1 picosecond resolution, thus the events corresponding to the active edges of all clocks in the system are aligned to the 1ps timescale. Considering that the frequencies of cores used in the simulations have a maximum of 2GHz, or 500ps, the resolution of the timescale allows to simulate frequency changes with a worst-case resolution of 4MHz, which is sufficient even considering the need to model the frequency changes that happen during a PLL lock.

To support dynamic frequency scaling the first modification performed is adding support for changing the period at which events are scheduled on a component –i.e, cores and NoC routers– basis. Components in GEM5 are synchronous logic blocks that can be as small as a circuit made of a limited amount of gates, or as large as a CPU core. All components have an associated `ClockedObject` class that is used to schedule events aligned to the component's active clock edge. Said events are used to simulate the operations that happen inside the component during one clock cycle. It should be noted that since GEM5 is an an event driven simulator, as part of the processing of one event, additional events may be scheduled at future clock cycles. On the other hand, if a component has no operation to perform at a given clock cycle, no event will be scheduled for it, thus improving the simulation performance. The `ClockedObject` class was extended with the possibility to dynamically change the period of the clocked component throughout the simulation.

Although this modification ensures that all events that are scheduled after the frequency change are placed at their correct point in time, this does not solve the problem of already scheduled events. Components can in fact schedule events also multiple clock cycles in the future, and the transla-

tion between the clock cycle and the event time point happens upfront in GEM5, before inserting the event in the global event queue. The event queue does not even have any knowledge of the clocked object associated with the event, nor the clock cycle. It merely scheduled events in the global time (in picoseconds). Thus, if a frequency change is made to a component that has at least one event already scheduled in the future, that event will be serviced at the wrong time, introducing a semantic misbehavior.

To solve this issue a framework was implemented to identify all the events already scheduled for a given component in order to move them forward or backwards to the correct time after the frequency change. To do so, the event management system of the simulator has been extended to support the possibility to move already scheduled events between different simulation times.

Changing the frequency of a component thus entails moving already scheduled events to the time they will need to be serviced considering the frequency change and updating the `ClockedObject` frequency so that subsequently generated events are scheduled at the appropriate time.

The last problem that was experienced is caused by moving events. Moving already scheduled events introduces the problem of the order of execution of events scheduled at the same time, as this could introduce additional semantic misbehavior. To solve this, GEM5 supports prioritization of events to enable ordering the execution of events scheduled at the same time. This feature was used by forcing all events belonging to the pipeline of a component to be executed in backward order, as described in [23], thereby guaranteeing that the simulator would not use as input data for a pipeline stage the data computed in the same clock cycle by the previous stage, which is not physically possible.

In addition, support for frequency islands was implemented, by grouping a set of components together, in order to allow the simulator to be used to explore frequency scaling strategies at different granularity levels. Figure 3.3 shows five different frequency islands for which all the events owned by the logic in a specific island must be moved together in case of a frequency change.

Although these modifications allow to safely change the clock frequency of an *isolated* component without introducing semantic misbehavior, two open problems remain.

The first one is how to manage signals that cross a clock domain, as this, if not handled correctly, would lead to metastability issues in a real hardware implementation, or semantic misbehavior in a simulator. This is the subject of Section 3.5.

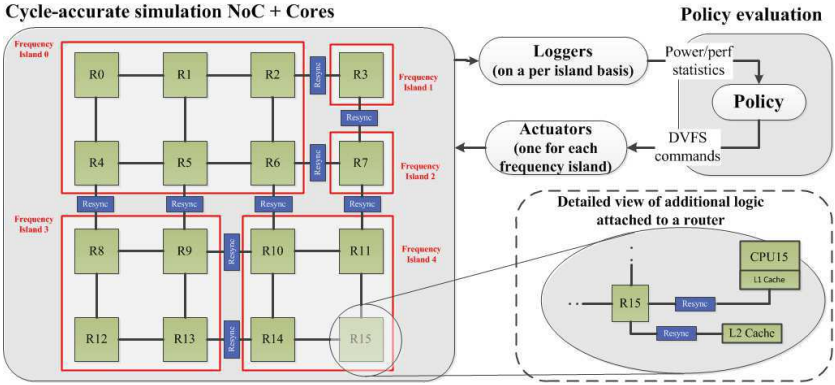

Figure 3.3: *Logic view of the simulated multicore detailing frequency islands and clock resynchronizers.*

The second one is how to model the overheads and limitations of the frequency scaling module. In this respect, the simulation framework supports two different DFS implementations: one that employs a single PLL for the whole chip and derives the clock for each frequency island through frequency dividers, and another using a dedicated PLL for each island.

In the first case, a frequency change is simulated as an abrupt change from the previous to the new frequency. Frequency change requests not aligned to a clock edge boundary are properly delayed till the next clock edge to avoid the insertion of clock glitches, as real world clock switch implementations do. This frequency scaling option is restricted to an integer sub-multiple of the base frequency.

Conversely, a clocking scheme employing a PLL for each frequency island allows to choose a wider range of clock frequencies without being restricted to an integer sub-multiple of the maximum frequency. In this case, frequency changes are implemented by modeling the PLL dynamics and changing the PLL set point. This is the subject of Section 3.6.

3.5 Accounting for the resynchronization overhead in a GALS architecture

Frequency islands are introduced in the simulator to allow grouping individual components in a common voltage and frequency domain. The granularity of the islands is configurable, ranging from each core and router having its own frequency to a single island encompassing the entire chip.

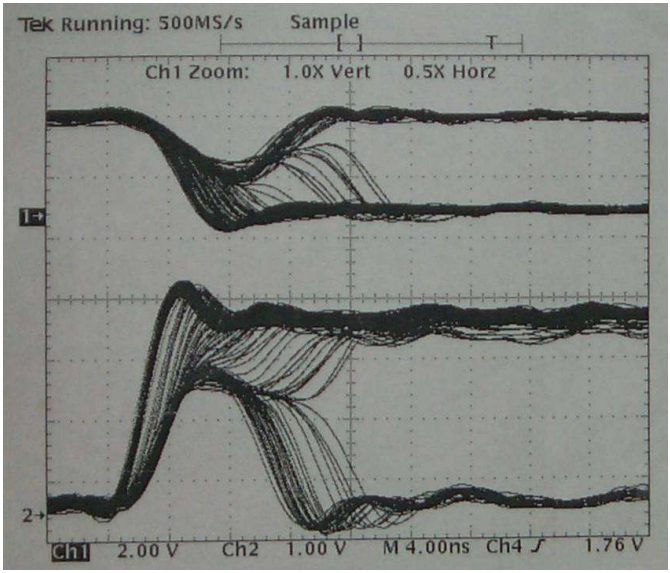

Figure 3.4: *Superimposed image showing various transients of the output of a flip-flop whose timings are violated, showing metastability issues. Photo taken from www.fpga-faq.com.*

Thus, the proposed simulator is able to model a GALS MPSoC. The GALS design methodology consists in building frequency islands with components, such as cores and NoC routers, that are synchronous inside, but run asynchronously relative to each other. This has the advantage of not requiring a power hungry clock distribution network to distribute a skew-free clock across an entire MPSoC, as well as opening up the possibility to perform dynamic frequency scaling at a frequency island granularity [12, 69].

Thus, the possibility to have different frequency islands is vital to test policies applicable to MPSoCs, but introduces the problem of handling signals that cross the boundaries of a clock domain.

In a digital system, a signal that crosses two logic blocks running at different frequencies or at the same frequency but different phases may incur in timing violations. In detail, the logic block receiving the signal will at some point in the circuit attempt to latch it using a flip-flop. However, due to the lack of phase relation between the clock in which the signal is generated and the clock of the flip-flop latching it, setup or hold timing may be violated. When this happens, a phenomenon known as metastability occurs, where the output of the flip-flop whose timings are violated may

exhibit an anomalous behavior, such as a temporary oscillation that eventually resolves to a random logic level, the output temporally being stuck at an analog level that is neither a logic 0 nor 1, or a transition to a logic level shortly followed – without a clock pulse – by another transition to the opposite one.

The most common solution to the problem is to resynchronize the signal using multiple flip-flops. For example, connecting two flip-flops in a chain without any combinatorial logic between them – that would introduce a propagation delay – leaves the output of the first filp-flop an entire clock cycle to resolve the metastable state before the signal is sampled again by the next flip-flop.

Coming back to the simulation of an MPSoC, instead of its actual hardware implementation, it was observed that completely neglecting the resynchronization between frequency islands would introduce semantic misbehavior in the simulation. To preserve the correct operation of the simulated MPSoC events related to signals that cross clock domains had to be delayed till the next active clock edge in the target clock domain.

However, even if the simulation would work under these conditions, the lack of a true resynchronizer model in the simulator taking account of its overhead means that the clock domain crossing is not simulated in a cycle-accurate way.

To maintain the cycle accuracy of the simulation, a resynchronizer model is implemented that can be inserted in the link between two NoC routers [62], or between a router and a network interface, which is a component that interfaces cores L2 cache controllers and memory controller to the NoC. In order to simulate an asynchronous multi-core an adequate number of resynchronizer components must be inserted on each frequency domain border. The resynchronization logic is a two way handshaking protocol that adds a minimal amount of logic and only two wires, i.e. request and acknowledge, to a network link.

The model of the resynchronizer is depicted in Figure 3.5. The left part of the figure reports the output port interface on the upstream router, while the right side provides the input port interface of the downstream router. When a new flit is ready to be sent out, the upstream router triggers the *new_flit* signal for 1 clock cycle. This result in the *req* signal being toggled one cycle later. Moreover, the back path in the upstream router switches the *busy* signal high. After the propagation delay across the network link the *req* signal enters to a two flip-flop chain in the downstream router, that is used to avoid metastability issues. The third flip-flip in the chain is used in couple with the *req_stable* signal as an edge detector, since our resyn-

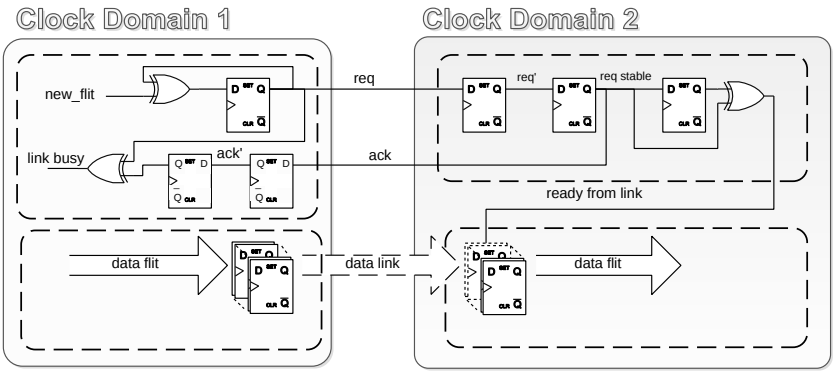

Figure 3.5: *The implemented resynchronization scheme to avoid metastability when crossing clock domains.*

chronizer works on edges to increase throughput [3]. The edge detector triggers the *data_valid* line, signaling that there is a valid flit on the link. The *req_stable* signal is also sent back as an acknowledge to the upstream router to signal the data transfer completion. Also the upstream router manages the *ack* signal using a two flip-flip chain to avoid metastability issues. The *busy* signal is used to prevent the transmission of new flits until the reception of the acknowledge signal. The implementation connects it to the switch allocator stage to enable or disable the allocation on the specific output port.

It is worth noticing that, even if it is fully functional and guaranteeing cycle accuracy, the proposed resynchronization scheme is thought of as a customization point for the simulator, and is extensible to allow implementing different resynchronization schemes to test their performance, such as asynchronous FIFOs [61].

3.6 Modeling the PLL dynamics

PLLs are commonly used to set the frequency of an MPSoC. Since PLLs are software-configurable, they are one of the options that allow implementing dynamic frequency scaling and, when coupled with programmable voltage regulators, also DVFS. Their main advantage is allowing a large range of output frequencies, usually integer multiples of the reference frequency, even though more advanced implementations such as fractional-N PLLs [34] can relax also this constraint. For this reason, they are more flexible than frequency dividers that can only produce frequencies that are

an integer sub-multiple of the input frequency, and more suitable for fine frequency control in the dark silicon age.

A PLL is made of a controllable oscillator, and a feedback loop that takes as input the phase difference between the reference clock and the produced one, suitably divided in order to have the same frequency of the reference clock. The feedback loop is used to control the frequency produced by the oscillator. Both the phase detector, loop filter and oscillator can be either analog or digital. For example, a ring oscillator is a common way to implement an analog voltage controlled oscillator (VCO), whose frequency can be set by varying the voltage used to power the oscillator. Another option is a LC-tuned oscillator whose frequency can be set by means of varactors, although due to the cost in terms of area of integrating inductors on-chip, this design is limited to applications requiring low phase noise, such as RF transceivers. A digitally controlled oscillator can instead be implemented with the same LC-tuned design by using a switched capacitor bank instead of varactors.

However, regardless of how it is implemented, the PLL is a dynamic system and thus a change in the frequency set point does not result in an immediate change of the output frequency. Thus, to accurately simulate a PLL in an instruction-set simulator its dynamics need to be simulated as well. This allows to correctly take into account the overhead introduced by a frequency change.

The goal is thus to integrate into the simulation framework a flexible model for the PLL dynamics, easily configurable to match the settling time of a given PLL. The ease of configuration design constraint, as well as the need for a computationally lightweight model call for the selection of a low order transfer function with as few parameters as possible. Considering however that most PLL exhibit an overshoot behavior during a frequency change [1, 71], a first order (one pole) transfer function is not sufficient, as it would not allow to simulate the overshoot behavior. It was thus decided to use a generic two pole transfer function, equation (3.1), as the underlying model to simulate he PLL dynamics.

$$G(s) = \frac{1}{1 + 2\frac{\xi}{\omega}s + \frac{s^2}{\omega^2}} \tag{3.1}$$

This transfer function has only two parameters, ξ and ω. The first one is limited in the $[0..1]$ range, and determines the overshoot of a frequency change transient, while the second affects the settling time. More in detail, the settling time can be computed as $4/(\omega\xi)$, while the overshoot (in per-

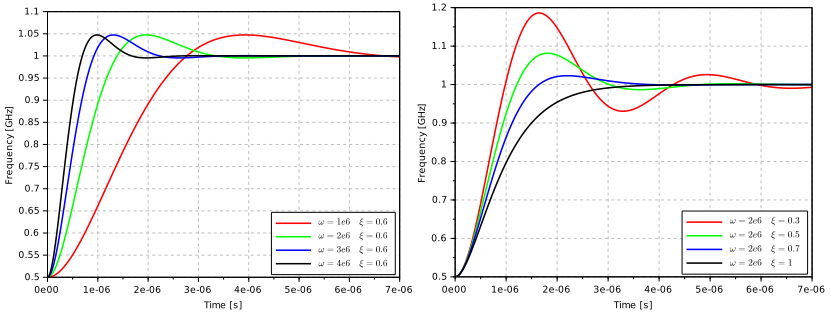

Figure 3.6: *Impact of the ω and ξ parameters of equation* (3.1) *on the simulated PLL step response.*

centage of the input step amplitude) is $100e^{-\xi\pi/\sqrt{1-\xi^2}}$. Figure 3.6 shows the simulated PLL response to a step change in the set point from 0.5 to 1GHz with varying parameters to better evidence their impact on the simulated dynamics. The left plot keeps ξ constant and sweeps ω from $1e6$ to $4e6$. As a result, the settling time decreases from around 6.7μs to 1.7μs, while the overshoot does not change. The right plot instead keeps ω constant, and sweeps sweeps ξ from 0.3 to 1, resulting in a reduction of the overshoot from 37% of the transient amplitude to zero. Notice how an increase in overshoot also causes a settling time increase, due to the introduced oscillatory behavior.

A step change in the frequency set point is then implemented by instructing the simulator to perform multiple individual frequency changes to the frequency island controlled by the PLL to track the two-pole step response, until the response has settled. This in turn results in multiple individual frequency changes to the components that belong to the frequency island. The process of performing an individual frequency change to one component happens as described in Section 3.4.

Two PLL models have been introduced in GEM5 as part of this thesis, that differ in how the differential equation describing the PLL closed loop response is integrated. The first one has the advantage of allowing a very flexible tradeoff between the precision of the frequency change tracking and simulation overhead. However, it has the disadvantage of allowing a new frequency change only after the frequency output has settled. The second one correctly simulates changes in the PLL set point also while the output frequency is still changing. This model is more suitable for aggressive policies that perform frequent frequency changes.

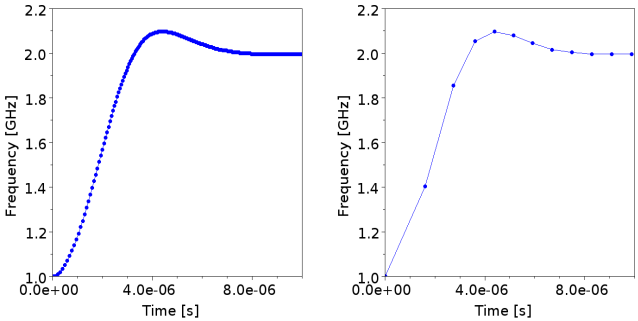

Figure 3.7: *Simulation of a frequency change from 1 to 2GHz with two PLL simulation granularities.*

3.6.1 First PLL model

The first PLL model is based on the fact that there exists a closed-form solution (3.2) for the step response of a two pole transfer function. This equation gives the frequency as a function of time for a step change from f_o, the old frequency, to f_n, the new desired one. Thus it is possible to sample this function at every clock edge to compute the next clock frequency, that will also determine the point in in time of the next clock edge.

$$f(t) = f_o + (f_n - f_o)(1 + \frac{1}{\sqrt{1-\xi^2}} e^{-\xi\omega t} sin(\omega t\sqrt{1-\xi^2} + acos(\xi))) \tag{3.2}$$

This results in continuously changing the clock period on a cycle-by-cycle basis until the step response reaches its steady state, allowing an accurate simulation of the frequency change during this transition phase.

As this process entails a large number of individual frequency changes, it introduces an overhead in the simulation. To allow the user to trade off simulation accuracy for speed, a configuration option k has been introduced, to sample the step response (and therefore cause a frequency change) not every clock period, but every k clock periods, thereby reducing the number of frequency changes.

Figure 3.7 shows the simulation of the PLL model when changing its frequency set point from 1 to 2GHz, where individual frequency changes are marked with a dot. The left plot shows the results with $k = 1$. The frequency transition smoothly follows the two pole step response, but to achieve this result 184 individual frequency changes to the clocked frequency island are required. The right plot shows the results with $k = 16$.

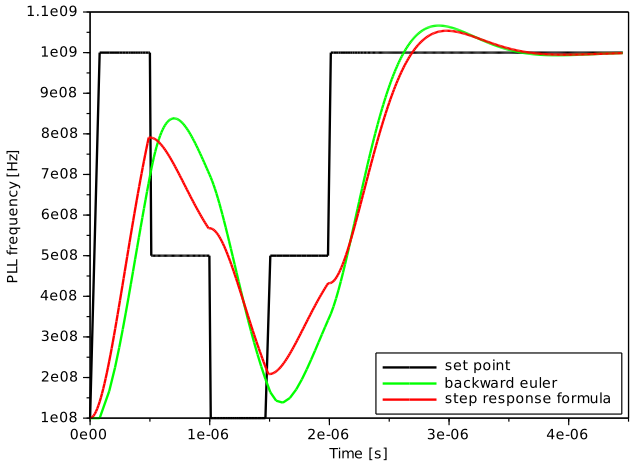

Figure 3.8: *Comparison between the two PLL models when frequency changes are faster than the PLL settling time.*

In this case the frequency change is approximated with only 12 frequency changes.

3.6.2 Second PLL model

As anticipated, the issue of the first PLL model is that it cannot correctly track the PLL output if changes to the frequency set point occur faster than the PLL settling time. This is because the first model does not actually integrate the PLL differential equations, but rather uses a closed-form solution that is only valid for a step response. Figure 3.8 shows in red the frequency output under fast set point changes of the first PLL model, and in green the same response of the second PLL model here presented. As can be seen, the step response formula results in discontinuities in the output frequency, and does not take into account that frequency may continue to increase or decrease before changing direction due to the dynamic nature of the system.

To correctly produce the output frequency value under arbitrary inputs, the only solution is to integrate the PLL differential equation on-line within the simulator. This is performed by turning the transfer function into a state space model, which in this case is a system of two differential equations, that is then integrated using the backward Euler method.

The translation from a transfer function to a state space model allows multiple solutions. A convenient state space model for (3.2) is (3.3). The two forms represent the same dynamic system if $c_1 a_{12} b_2 = -a_{12} a_{21} = \omega^2$

and $-a_{22} = 2\xi\omega$, and the conversion from the parameters of the transfer function model to the state space one can be performed using the `tf2ss()` library function in the Scilab or Octave mathematically oriented programming languages.

$$\begin{cases} \begin{bmatrix} \dot{x_1} \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ b_2 \end{bmatrix} u \\ y = \begin{bmatrix} c_1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \end{cases} \tag{3.3}$$

The backward Euler method integrates the differential equation $\dot{y} = f(y)$ approximating the value at the next time step as $y(t+1) = y(t) + hf(y(t+1))$, where h is the integration step. Applying the definition to (3.3) results in (3.4).

$$\begin{cases} \begin{bmatrix} x_1(t+1) \\ x_2(t+1) \end{bmatrix} = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + h \begin{bmatrix} 0 & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_1(t+1) \\ x_2(t+1) \end{bmatrix} + h \begin{bmatrix} 0 \\ b_2 \end{bmatrix} u(t+1) \\ y(t+1) = \begin{bmatrix} c_1 & 0 \end{bmatrix} \begin{bmatrix} x_1(t+1) \\ x_2(t+1) \end{bmatrix} \end{cases} \tag{3.4}$$

The system of equation can then be solved to obtain the equations of the integration algorithm. Omitting for brevity intermediate computations, the resulting algorithm is (3.5).

$$\begin{cases} x_2(t+1) = \frac{x_2(t) + ha_{21}x_1(t) + hb_2u(t+1)}{1 - h^2a_{21}a_{12} - ha_{22}} \\ x_1(t+1) = x_1(t) + ha_{12}x_2(t+1) \\ y(t+1) = c_1x_1(t+1) \end{cases} \tag{3.5}$$

The last step to complete the algorithm is how to initialize the state variable in order to have the PLL model output the default frequency when the simulator is started. This can be done by solving (3.5) assuming an æquilibrium condition. Again, omitting trivial computations, the state initialization in order to have an initial output frequency of $y(0)$ is (3.6)

$$\begin{cases} x_1(0) = y(0)/c_1 \\ x_2(0) = \frac{ha_{21}x_1(0) + hb_2u(0)}{-h^2a_{21}a_{12} - ha_{22}} \end{cases} \tag{3.6}$$

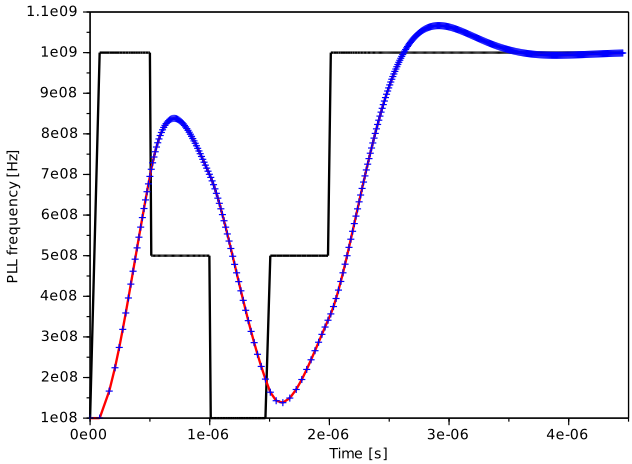

Figure 3.9: *Simulation of the PLL dynamics using the backward Euler integration method. Blue crosses represent the point when the integration algorithm is applied*

The implementation of this integration method in GEM5 uses a variable integration step, selected as a multiple of the current PLL period, resulting in continuously changing the clock period on a cycle-by-cycle basis, allowing an accurate simulation of the frequency changes. This allows to achieve the same tradeoff between simulation performance and accuracy as in the first PLL model, although in this case the integration step cannot be arbitrarily increased, or numerical instability of the integration algorithm would occur.

Figure 3.9 shows the second PLL model. As can be seen, the simulated frequency output smoothly follows the two pole step response even for fast set point changes. The variable integration step occurring every 8 clock edges, and can be seen from the fact that the blue crosses are less frequent when the frequency is low.

3.7 Frequency change simulation overhead

As already explained, simulating a frequency change in a detailed way is a complex process. The PLL model generates a certain number of individual frequency change events in the cycle accurate simulator to track the PLL dynamics, and each frequency change event requires to find and move in time all the events scheduled by the components of the frequency island controlled by the PLL. This introduces an overhead in the time required to

Table 3.2: *Timing overhead (in microseconds) for performing a frequency change depending on frequency island sizes and change model.*

Island size	16	8	4	2	1
Frequency divider (single change)	749	395	201	89	43
PLL model ($k = 16$)	5969	3666	1918	997	470
PLL model ($k = 1$)	71529	42916	24626	12186	5249

perform the cycle-accurate simulation, slowing down the simulator. This section is dedicated to assessing said overhead, to make sure it is negligible compared to the typical simulation time of a cycle-accurate simulator for MPSoCs.

In particular several experiments were conducted considering different frequency island sizes with 1, 2, 4, 8, 16 NoC routers. Results are reported in Table 3.2, that reports the overhead of a single frequency change (frequency divider model), and that of the PLL model with two different simulation accuracies. It is interesting to note that the absolute time required to perform one single frequency change is below one millisecond, thus is negligible. The PLL model, as expected, requires more time as it implies multiple individual frequency changes. Also, the time required to move events decreases with the size of the frequency island. This was expected, since the number of events is bound to the number of components in the frequency island. Moreover, the reported data highlights the quasi-linearity of the time required to move the events of a frequency island in response of a frequency change.

3.8 Voltage scaling

Although the modifications shown so far are sufficient to model a DFS actuator, to extend the proposed model for DVFS it is required to model also the voltage scaling and interlocking part.

Voltage scaling is simulated accounting for the switched voltage regulator dynamics, which are approximated using the same two pole transfer function of the PLL model, with different parameters to account for the different settling time and overshoot of the voltage regulator. Also, the same integration methods used in the PLL model are used to track the voltage as a function of time produced by the switching regulator.

However, it is also necessary to consider the overhead introduced by the interlocking between the PLL and voltage regulator. In fact, during a DVFS transition to a higher performance state it should be avoided to

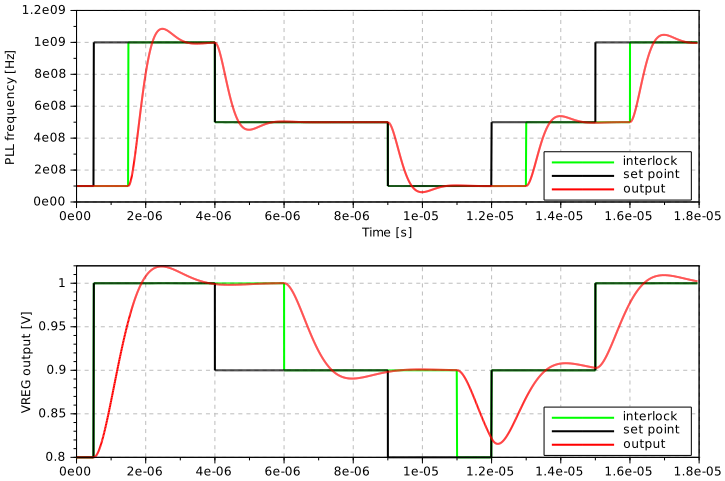

Figure 3.10: *DVFS transitions considering the PLL and voltage regulator dynamics and interlocking. Voltage and frequency set point (black), set point after interlock (green), and PLL/voltage regulator outputs (red).*

increase the frequency before the operating voltage, as this may lead to timing violations as the frequency island is operated at a too low voltage for the imposed frequency.

A typical solution to this problem is to introduce an interlocking mechanism that ramps up the voltage first, and only when it has reached a safe level the frequency can start increasing. The interlock should also do the opposite the opposite during a frequency decrease.

Figure 3.10 gives an overview of DVFS module implemented in GEM5 evidencing how multiple frequency transitions are simulated using the proposed detailed model. As can be seen, when the frequency set point is increased, the actual frequency change is delayed to let the voltage reach a safe level. On the contrary, a frequency set point decrease causes an immediate decrease of the frequency output by the PLL, but the voltage decrease is delayed.

The implementation of the interlocking mechanism is designed as a customization point for the simulator, and although the presented interlocking policy is fully functional, it is possible to extend the simulator to implement different policies, such as coupled DVFS transitioning schemes [4] to assess their relative performance compared to the baseline model.

Table 3.3: *Experimental setup: processor and router micro-architectures and technology parameters.*

Processor core	2 GHz, out-of-order Alpha core
Int-ALU	4 integer ALU functional units
Int-Mult/Div	4 integer multiply/divide functional units
FP-Mult/Div	4 floating-point multiply/divide functional units
L1 cache	64kB 2-way set assoc. split I/D, 2 cycles latency
L2 cache	512KB per bank, 8-way associative
Coherence Prot.	MESI token (for real traffic) [2]
Router	3-stage wormhole switched with 64b link width, 4vcs per vnet
	Frequency variable from 500 MHz to 2GHz
Topology	4x4 2D-mesh, based on Tilera iMesh network [95]

This accurate DVFS scheme allows to assess run-time policies taking into account the non-ideal behavior of the actuator that could dwarf their benefits, and are thus a key component of the proposed simulation flow.

3.9 Results

This section is dedicated to showing the flexibility and scalability of the proposed simulation framework to estimate and evaluate different microarchitectural solutions. The results shown in this chapter are focused on the implemented DFS scheme when applied to the NoC, while results regarding the DVFS module for cores, which shares many design concepts such as the resynchronizer and PLL model, is discussed in the next section where the thermal model is introduced.

Four different results are discussed. First, Section 3.9.1 presents a simple yet effective test to assess the correctness of the frequency scaling implementation. Section 3.9.2 shows how the proposed flow can be used to obtain both energy and performance metrics while changing the frequency of NoC routers at run-time, throughout the simulation. Section 3.9.3 shows an example of design space exploration of NoC routers frequencies, while Section 3.9.4 demonstrates the capability to support run-time optimization policies.

All the presented results are obtained using the microarchitectural configuration reported in Table 6.2. In particular the simulated architecture is a 16-core Alpha 21364 architecture, as it allows to assess the proposed simulation framework on a reasonable multi-core employing a NoC-based interconnect.

3.9.1 Implementation correctness

The proposed framework implements an asynchronous NoC model inside a cycle accurate simulator, also allowing to implement DFS. However, the

implementation impacts the core of the simulator, thus a proof of the correctness of this work should be provided, since the final semantics cannot be guaranteed *a priori*.

While the best solution to assess the validity of our asynchronous NoC should be a comparison between our model and an implemented HDL version of the simulated multi-core, there are multiple issues that prevent this. First, it is quite difficult to synthesize a complete multi-core implementation inside an FPGA consisting of NoC, cores and caches with cache controllers due to the limits of current FPGAs. In this perspective many solutions have been proposed providing synthesizable NoCs, but usually relying on synthetic traffic generators to provide traffic load [72]. Moreover, it was found that a simulation using synthetic traffic only is not enough to prove the validity of the solution, since a missed packet can often go unnoticed while performing simulations with synthetic traffic, while when cores and cache coherence protocols get involved even one lost packet can completely change the outcome of the simulation.

To this extent this section provides a weaker yet effective proof of the DFS implementation correctness focusing on the formal expected behavior of the simulation. First of all a black-box test was performed by simulating the full architecture. The code chosen to run on the cores is a subset of 9 tests from the MiBench [42] suite. The tests were performed with a simple policy that changes the frequency of the NoC every 100ns, with the sole aim of stressing the added DFS and resynchronization functionality. No discrepancies were found between the expected and obtained output. This shows that the modifications to the GEM5 simulator did not introduce errors that affect code execution.

Second, the timing accuracy of the introduced components were checked. To this extent multiple simulations starting from the same multi-core and using the same benchmark set were considered, but changing the frequency of the NoC for each simulation. These tests were performed using a synthetic traffic generator, to be able to control the rate of packets in the NoC. As the focus of this second test is assessing the resynchronization scheme, a resynchronizer module has been added between each router pair and between each router-L2 and router-L1 pair.

Figure 3.11 reports the number of routed flits as a function of the router frequency, in two different network load *scenarii*. The router frequencies range from 500 MHz to 2GHz with a step of 20 MHz, i.e. there are $\frac{2GHz-500MHz}{20MHz}$ different simulations with fixed frequency. The left part of Figure 3.11 reports the simulations using 0.50 flit/port/cycle while the right part of the same figure reports the same simulations using an injection rate

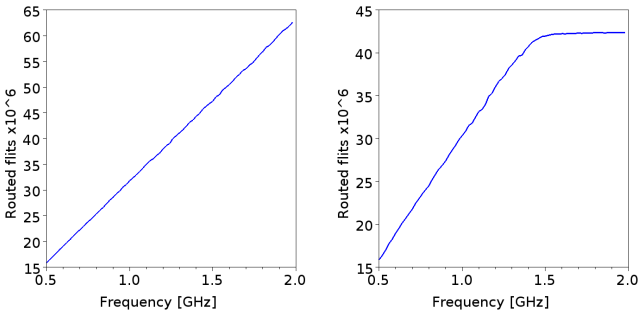

Figure 3.11: *Number of routed flits as a function of router* ***frequency****, with two network load* scenarii.

of 0.07 flit/port/cycle. The first case shows how, with a sufficiently high network load, the increase in frequency produce a linear increase in the processed flits. This means that the resynchronizers, as expected, do not affect the linearity of the frequency/processed flits relation. Moreover, the second case shows saturation at 1.5GHz. This is not a microarchitectural saturation, but rather shows that the NoC does not benefit from high frequencies when the injection rate is low.

The third test addresses the timing correctness of the DFS implementation. The architecture is the same 16-core MPSoC and also the traffic load is as in the second test, but the change in frequency between simulations was emulated using a PWM-like scheme alternating between only the two boundary frequencies: 500MHz and 2GHz. For each simulation a fixed number of frequency changes is performed between the two frequencies considering all the routers as a single frequency island. Each simulation is different from the others by the duty-cycle, i.e. the percentage of the simulation time spent in each one of the two frequencies. Throughout each of these simulations there were two frequency changes (one high-to-low and the other low-to-high) per 800ns, for a total of 40000 frequency changes per simulation.

In particular, Figure 3.12 reports the received packets as a function of the duty-cycle ranging from 2.5%, where most of the time of the NoC is spent at 500MHz, up to 97.5%. The left and right graphs of Figure 3.12 are two set of simulations that differ in the flit injection ratio exactly as in the previous test. As can be seen, the same relation between average frequency and routed flits can be found. Comparing this figure with Figure 3.11, it

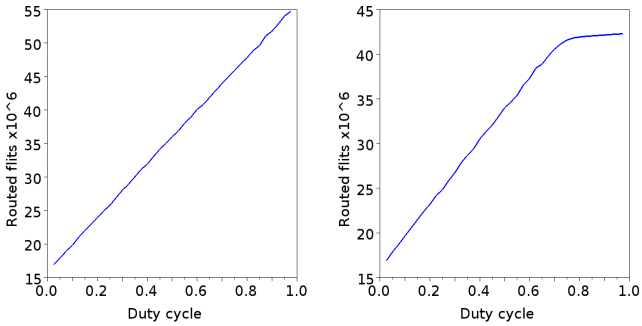

Figure 3.12: *Number of routed flits as a function of* ***duty cycle****, with two network load* scenarii.

can be noticed that the range of changes is lower, but this is to be expected as the duty-cycle did not reach 0% nor 100%.

3.9.2 Metrics extraction

This section shows the capability of the proposed framework to report at a fine granularity both performance and power metrics of the simulated architecture. Figure 3.13 reports both performance and power metrics for router $R7$ (see the 16-core simulation architecture details as reported in Table 3.3, as well as Figure 3.3 for the R7 placement in the NoC), considering the *fft* benchmark of the MiBench suite replicated on each one of the 16-cores. Power and performance samples are collected every $100ns$, while the simulated DFS actuator is configured to alternate the frequency of one NoC router between 100MHz and 1GHz with a period of $10\mu s$, and a duty cycle of 25%. The clock divider model was used in this test instead of the PLL one, so the frequency changes are instantaneous.

The contention information is obtained as the number of flits in the buffers of the selected router. The power information includes both the dynamic and clocking power, while the static power has been omitted as its value is constant throughout the simulation. Figure 3.13 show how the frequency of the router impacts the contention and consequently its performance. For example, when the router frequency is low also the dynamic power is low while the contention show an increasing trend. On the other hand, when the router frequency is high also the dynamic power is high, while the contention remains low. However, even if both contention and power are strictly bound to the frequency, the contention information also

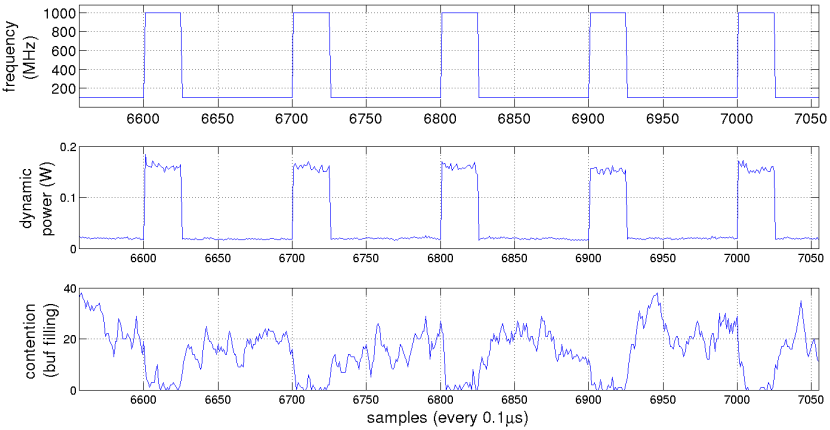

Figure 3.13: *Frequency, power and contention of router 7 in the simulated 16 core architecture while executing the fft benchmark.*

strongly depends on the network load which in turn depends on the cache misses dynamics of the executed code. For example, around sample 6950 the contention tends to slow down even if the router frequency is low. The power consumption, instead, due to the clocking power contribution, depends on the network load to a much lesser extent.

3.9.3 Design space exploration

Since the proposed framework allows for an accurate estimate for both power and performance metrics considering frequency scaling modules, it enables the possibility to explore different power-performance trade-off acting on frequency. In this perspective this section discusses how the proposed framework helps to select the most suitable set of frequencies to be assigned, even on a per router basis, in order to optimize the power-performance figure of merit. It is worth noticing that these experiments do not directly exploit the DFS module, while the possibility to set a different frequency for each router in the NoC stresses the asynchronous NoC design as well as the possibility to assign different static frequencies to different components expanding the GEM5 simulator capabilities. Figure 3.14 reports collected data for 9 different MiBench, run up to completion. The reported data are for router $R7$, which was placed in a frequency island on its own. All the other routers as well as the cores run at the same frequency of 1GHz. A test was performed per each considered MiBench, where for

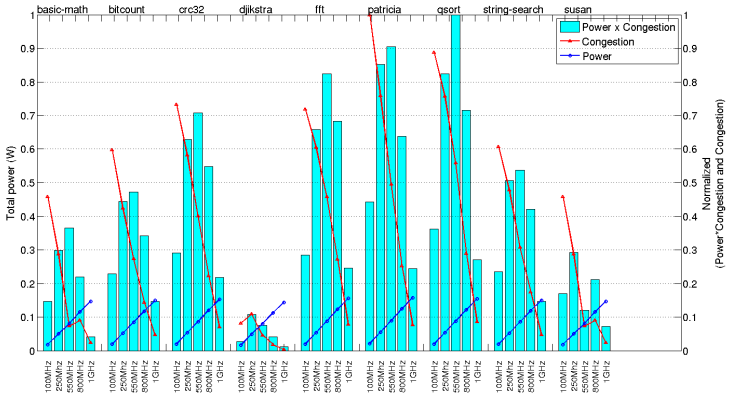

Figure 3.14: *Power, contention and Power-Contention-Product as a function of router frequency, for selected MiBench benchmarks.*

each simulation $R7$ is set at a fixed frequency. In total, five simulations per benchmarks were performed with different frequencies covering the range between 100MHz and 1GHz. The results reported in Figure 3.14 show the router average contention (red lines), the dynamic power (blue lines) and the power times contention or Power Contention Product (PCP) as an aggregate power-performance metric as bar graphs. Both contention an power times contention data are reported as normalized data on the right y axis, while power is reported in Watt on the left y axis. The contention decreases with the frequency as expected, thus all the red lines have a decreasing slope. On the other side increasing the frequency increases the power consumption, and this aspect is confirmed for each simulated benchmark.

The power times contention metric represents a comprehensive figure of metric for the power-performance optimization. Two main considerations can be discussed starting from the reported data. First, when the frequency is low, i.e. 100 MHz, a small increase, i.e. 250MHz does not significantly face the contention in the router. This aspects means that the power consumption increases while the level of the contention remains almost the same. To this extent the PCP metric tend to increase its value as confirmed for all the simulated benchmarks. For example *crc32* reports a normalized PCP equal to 0.29 @100MHz while the PCP is 0.64 @250MHz. However, if the frequency increases there is a point when the PCP gets a decreasing slope providing also lower values. This means that the power consumption increase due to higher frequency is compensated by the capability of

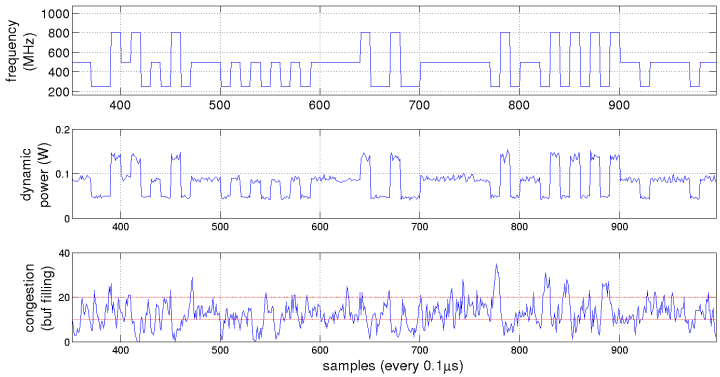

Figure 3.15: *Frequency, power and contention traces sampled while running the described DFS policy.*

the router to serve all the incoming flits quickly, thus maintaining the contention at a significantly lower level. In this perspective, increasing the frequency above a certain level will provide higher PCP values, since there are no more flits to be served, thus the increase in frequency introduces an increase in power consumption only, without any benefit on the contention metric.

3.9.4 Run-time optimization policies

While choosing a single frequency for the NoC, based on design space exploration, can be an effective solution while running applications with known memory access requirements, for general-purpose computing platforms a run-time change of the frequency of different islands of NoC routers can further optimize the power-performance trade-off. One of the possible uses of the proposed simulation framework is to evaluate the quality of different DFS policies operating on NoC routers.

In this perspective, this section discusses a simple policy used only as an example to highlight the flexibility of the simulation framework, while a more thorough discussion on power-performance policies for NoC routers is delayed to Chapter 6.

A very simple policy is here presented that can switch between three frequencies, a high one, a medium and a low one, being respectively 800MHz, 500MHz and 250MHz. The switch is managed using threshold values on the local router contention, that is the number of flits stored in the input

ports of the considered router. In particular, the policy compares the sampled contention with two thresholds deciding which frequency to assign. As the focus of this simulation is to assess the possibility to implement run-time policies rather than to assess a specific policy, the frequency divider model was used instead of the PLL one, thus again showing instantaneous frequency changes. At the beginning of the simulation the frequency is set at 500MHz. Then, the contention values are sampled every 100ns. However, the policy can change the router frequency on a multiple of the sampling period, to limit the number of frequency changes. Figure 3.15 reports a timing diagram showing frequencies, dynamic power and contention levels for $R5$ on a 16-cores running the FFT MiBench benchmark. The high and low thresholds are set at 20 and 10 flits respectively, as highlighted by the red lines in the contention graph. The frequency change limit is set at 10 times the sampling period, i.e. once a frequency change has been made, it has to be kept constant for at least 1us.

The results highlight how the simulation framework is capable of simulating microarchitectural details such as the NoC router operation and expose the metrics required to implement run-time policies and assess their performance.

3.10 Conclusions

This chapter proposes a novel simulation framework with a focus on assessing thermal and power-performance policies for MPSoCs. The proposed solution allows for different exploitations during architecture and policy design space exploration. The proposed simulator allows to set different frequency islands, down to the granularity of a single core or router, and change the frequency of each island dynamically during the simulation to test different power-performance policies. This is made easy thanks to the provided actuation system, thus exposing a simple yet scalable framework to write and test different policies. Moreover, the asynchronous design model represents an improvement with respect to the state of the art cycle accurate simulators, allowing to evaluate the power reduction due to better clock tree organization, as well as to test different resynchronization schemes. Third, the provided PLL model allows to evaluate the transient behavior during every frequency change on a per island basis, with the possibility to trade-off accuracy for simulation speed. Last, the proposed results strengthen the semantic of the modified NoC allowing GEM5 to support both synchronous as well asynchronous NoC for both synthetic as well as real traffic *scenarii*.

It should be stressed that the simulator described in this chapter is an enabler for the assessment of the work described in the subsequent ones. This is because the developed thermal control and power-performance methodologies have been implemented as policies in the presented simulator. This made possible to verify the performance of the proposed policies in a realistic setting, also considering the sensor and actuator nonidealities, and to compare their performance with respect to other state-of-the-art policies.

CHAPTER 4

Modeling and simulation of an MPSoC thermal dynamics

All models are wrong, some are useful.

George E. P. Box

THE thermal management problem in MPSoCs is becoming more significant [53] as chip feature size scaling progresses. This is caused by the worsening power density due to non ideal Dennard scaling [28], where the increasing number of transistor per chip is no longer compensated by a corresponding reduction in the per-transistor power dissipation. As such, this increase in power density is causing the so called dark silicon problem [37, 84], where power and thermal constraints limit the portion of a chip that can be turned on at full speed. It is therefore undeniable that modeling the thermal dynamics of a chip is more important than ever, to be able to design effective cooling methods and run-time policies to maximize performance subject to thermal constraints, despite the variable workloads and operating conditions that modern MPSoCs face.

This chapter presents a novel, extensible thermal model for MPSoCs. This model is based on compact RC networks, as this was proven to pro-

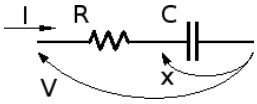

Figure 4.1: *Simple RC circuit used to demonstrate the difference between ODE and DAE.*

vide good performance and sufficient adherence with respect to more elaborate simulation methods, such as the finite element method (FEM) and measurements taken on actual chips [46].

The proposed thermal simulator is built upon the recent advances in modeling languages, being thus designed from the ground up using an object-oriented and component-based approach. This results in significant advantages both in terms of flexibility, where any component of the thermal dissipation stack can be substituted with another with a compatible interface, and in terms of testability, where each component can be tested in isolation prior to being integrated in the final thermal model.

4.1 Object-oriented modeling

This section briefly describes the core concept of object-oriented modeling, and how it differs from ordinary causal modeling, using an electrical example for simplicity. Consider this simple example of an RC circuit, as shown in Figure 4.1.

A straightforward way to model its behavior is through an ODE or ordinary differential equation. However, this requires to decide *a priori* which are the input and outputs of the system, leading to a lack of reusability of the model. For example, if the input is the voltage (i.e. the circuit is connected to a voltage generator), then a model that allows to compute the current flowing through the circuit is Equation (4.1).

$$\begin{cases} \dot{x} = \frac{V-x}{RC} \\ I = \frac{V-x}{R} \end{cases} \tag{4.1}$$

However, the same model is no longer adequate if the same circuit is used in a context where the input is the current, while the output is the voltage. In such a case, a different model is required, as shown in Equation (4.2).

$$\begin{cases} \dot{x} = \frac{I}{C} \\ V = x + RI \end{cases} \tag{4.2}$$

The main limitation of these models is that they are *causal*, i.e, they are written assuming that some of the boundary variables that connect a model component to the external world are either inputs or outputs.

This forces to have multiple models of the same physical system with different combinations of inputs and outputs, and it is easy to understand how this could become unmanageable.

In contrast, consider Equation (4.3). It is still a model for the same circuit of Figure 4.1, but it is no longer an ODE equation, it is a differential algebraic equation (DAE).

$$\begin{cases} C\dot{x} - I = 0 \\ x + RI - V = 0 \end{cases} \tag{4.3}$$

As can be seen, the model is composed of an implicit differential equation, and an implicit equation. The model is thus not oriented, *a-causal* or *declarative*. Contrary to oriented models, that are described in a form that is close to their solution algorithm, DAE-based models are implicit, and thus require symbolic manipulation to be rewritten in a form that can be solved. If however, this step is performed automatically by a solver, one can write a single model that can be used regardless of the boundary equations, becoming thus a reusable *component*.

An object-oriented modeling environment is characterized by allowing to write a-causal models based on DAE equations, allowing therefore to model the physical phenomenon of interest without the need to decide at modeling time which are the inputs and outputs.

Connecting multiple components in an object-oriented modeling environment requires to write additional equations that bind the variables of the components being connected. For example, consider the electrical circuit of Figure 4.2, built using two components, an ideal voltage source of 1 Volt, and the RC circuit previously modeled.

$$\begin{cases} C\dot{x} - I = 0 \\ x + RI - V = 0 \\ V_0 = 1 \\ V_0 = V \\ I_0 + I = 0 \end{cases} \tag{4.4}$$

The full DAE model of the circuit is shown in Equation (4.4). The first two equations come from the RC model, the third one is that of the voltage

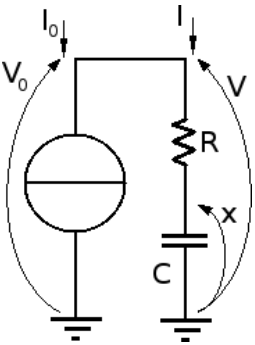

Figure 4.2: *Simple RC circuit with generator used to demonstrate the difference between ODE and DAE.*

source, while the last two are connection equations that arise as the two components are connected. In detail, for an electrical connection they are the Kirchhoff's voltage and current law.

This shows the component-based approach that is made possible by object-oriented modeling. It allows to describe individual components independently, without specifying *a priori* the inputs and outputs, and connect them to build an entire system. To solve this model, however, a symbolic manipulation pass is needed before the system can be numerically integrated. Thus, even though the object-oriented approach simplifies model development, it requires specialized environments to solve the so obtained models in an automated way.

4.1.1 Why the Modelica Language

The thermal simulator proposed in this paper is written in Modelica [38,60], that is an object-oriented modeling language. It differs from an ordinary programming language as its main purpose is the simulation of dynamic systems. The language has built-in support for writing differential algebraic equations just like an ordinary programming language has support for assignment of an expression to a variable, and the runtime takes care of integrating them. The required numerical solver is embedded in the runtime, and has proven effective and accurate over decades of application; the interested reader can find the underlying theory e.g. in [25].

In addition, it supports component-oriented modeling where each individual component, such as a layer of the 3D chip, or the heat sink is modeled separately with its own equations, and can interact with other com-

ponents through special types called *connectors*, which when instantiated introduce the necessary connection equations to bind the variables of the connected components. Components can be parametric, thereby having the same equations but different parameters, like the different heat capacity and thermal resistivity that differentiates a layer of silicon from one of copper. Other than enhancing code structuring and reuse, components ease validation as they can be individually tested prior to instantiating them in a complex system.

A complete system such as a 3D chip including the heat dissipation stack can thus be modeled by first designing and then connecting individual components, while the Modelica compiler uses symbolic equation manipulation techniques to combine the individual differential equations into a single system of equations that can then be integrated by the Modelica runtime. It is worth noticing that, although Modelica is a textual language, models can be composed graphically. For example, components can be connected graphically through a simple user interface. Also, individual components that are just a combination of lower level components can be designed graphically. This further enhances the ease of use of the Modelica language.

The choice of the Modelica language thus stems from its flexibility, coming from a higher level language that understands linear and nonlinear differential equations natively, relieving the programmer from the burden of integrating them, as well as from the object-oriented and component-based nature that eases modifications to the chip structure (2D vs 3D), and the heat dissipation stack. This allows to make the proposed thermal model easily extensible, for example to explore different thermal dissipation stacks introducing advanced solutions such as TECs and heat pipes although these extensions are deferred to future works.

Free software implementations of the Modelica compiler and runtime are available. The OpenModelica environment [39] has been here selected due to its maturity and stability.

4.2 Modelica Thermal Model

The proposed thermal simulator [85] is composed of a collection of elementary equation-based components and connectors (in the Modelica jargon, a *model library*) together with a native library written in C++ that interfaces the thermal simulator with the rest of the simulation flow as described in Chapter 3. At the lowest level of said library stand the *volume* and the *conductor* models. The former represents a finite, parallelepiped volume

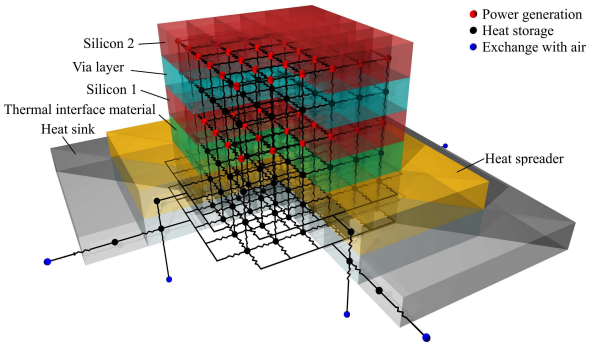

Figure 4.3: *Synthetic geometric representation of the 3D chip thermal model (not in scale, and some volumes were removed to better show the thermal network).*

of material with known physical properties. It is characterized by one differential equation for its energy balance, and has six connectors (one per face) to exchange heat with neighboring elements. The latter has two connectors, and takes care of modeling thermal exchanges due to temperature differences, like conduction and convection. Quite intuitively, assembling such components gives rise to a capacity/conductance (or equivalently, resistance) network, that comes to be the RC equivalent of the modeled thermal one.

The two models above are then used to build the *layer* component, which is a planar structure composed of an uniform grid of volumes with uniform physical properties. The layer component is used to model all the layers of the 3D chip, as well as the thermal interface material, by simply matching the parameters with the physical properties of the layer material. The heat spreader and heat sink components are instead modeled using a non-uniform grid that is finer at the center in order to match with the chip layers, surrounded by four and eight trapezoid volumes respectively for the heat spreader and heat sink to take into account the increased side length with respect to the silicon die. This technique is also adopted in HotSpot [46].

The full stack of components used to model a 3D chip is depicted in Figure 4.3 although obviously with a smaller number of volumes than the real design, to enhance the readability of the figure. Starting from the bottom, we can find the heat sink, which exchanges heat convectively with air. Then there is the heat spreader, which is made of a good heat conductor such as copper, to spread the heat produced by the small chip die area laterally, in order to more uniformly heat the heat sink. Then we have the first layer in the stack, which is the thermal interface material. It is used to provide good

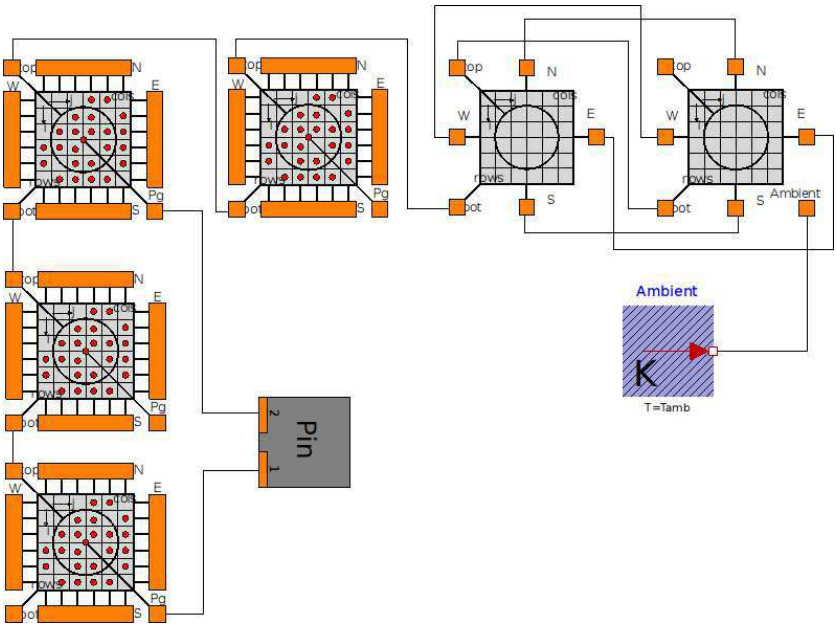

Figure 4.4: *The 3D chip Modelica diagram, showing the individual components.*

and uniform thermal contact between the silicon bulk and heat spreader, avoiding surface nonuniformities introducing voids under certain parts of the chip, a matter that would lead to severe hot spots. The next layer is the first silicon layer of the 3D chip, then there is a layer used to model the thermal conductivity of the through-silicon vias, and finally the second silicon layer.

Figure 4.4 conversely shows the graphical Modelica view of the 3D chip. As can be seen, the model is composed of four layers: the first silicon layer, the via layer that interconnects it with the second silicon layer, and finally the thermal interface material layer. Then there are the heat spreader and heat sink, which exchanges heat with the ambient. Last, a power generation block injects the power into the two active silicon layers. This block is implemented in a native language (C++), and receives the power values through a TCP socket from the simulation manager described in Chapter 3. It also performs the power and temperature mapping using the chip floorplan information. This is because the power data produced by the simulation flow is at a per functional unit basis, whereas the thermal model is based on an uniform grid. It becomes thus necessary to map the power

Table 4.1: *Microarchitectural parameters.*

Processor core	2GHz, out-of-order Alpha core
Functional Units	4 Int-ALU, 2 Int-Mult/Div, 2 FP-Mult/Div
L1 cache	64kB 2-way set assoc. split I/D, 2 cycles latency
L2 cache	512KB per bank, 8-way associative
Coherence Prot.	MESI
Router	3-stage wormhole switched with 64b link width, 4vcs per vnet Frequency variable from 100 MHz to 2GHz
Topology	2D-mesh 4 tiles 2x2 (3 CPU per tile)
Technology	32nm at 1.1V

from the functional units to the grid, as well as do the opposite with the temperature. For the power to temperature mapping, it is done as the sum of the power of the functional units that intersect the grid cell, multiplied by their intersection area and divided by the functional unit area. The mapping of the temperature is instead done through a weighted average of the temperatures of the grid cells that intersect the functional unit, proportional to their area.

To be able to simulate the model it is necessary to fill in the various parameters of the components, such as specific heat, thermal conductances and thickness of the various layers. For the 3D chip, these values are taken from [19].

It is worth noticing that to simulate a 2D chip instead of a 3D one it is sufficient to remove the additional components that are present only in a 3D chip, namely the through-silicon via layer and the second silicon layer, as well as to change the layer parameters such as the bulk thickness. This shows the flexibility of the component-based modeling approach.

4.3 Experimental results and validation

This section shows the operation of the proposed thermal simulator when embedded in the full simulation flow of Chapter 3. Note that the goal of the section is to show the flexibility of the entire simulation flow, and its ability to easily design and validate thermal policies exploiting DVFS as actuator. All the considered *scenarii* are evaluated using a tiled multi-core architecture simulating real applications from the MiBench suite [42]. Each tile has a three stages virtual channeled wormhole router that connects it to the rest of the multi-core as well as an L2 bank. The L2 is shared between all the tiles of the multi-core but it is physically split, i.e. one bank per tile. Last, we considered three processors per tile. In particular, we focus on an Alpha out-of-order processor model. Table 6.2 reports the main architectural

parameters.

4.3.1 Validation with HotSpot

As the interconnection between the components that build the thermal model is trivial (see Figure 4.4), the validation of the thermal simulator ultimately amounts to that of the differential equations in its individual components. Thus, for the purpose of validation, the components were combined to obtain a 2D chip model equivalent to the HotSpot model. The Modelica and the HotSpot models were then run with an identical power trace and chip floorplan. HotSpot was configured in transient mode, with the grid model and a grid size identical to the Modelica one. Figure 4.5 shows the difference between the state variables (i.e., the temperatures of all the volumes) values in the proposed Modelica simulator and HotSpot. As can be seen, the temperature difference is limited to 0.02°C due to numerical errors, and decreases as the simulation progresses. HotSpot was validated [48] against a FEM simulator, resulting in a 6% steady state error, and a thermal test chip showing a 5% steady state error, and a 7% error for what concerns simulating the transient behavior. Thus, the discrepancies between the proposed model and HotSpot are within the simulation errors of HotSpot, so the library components are validated, and can be used safely for new chip geometries.

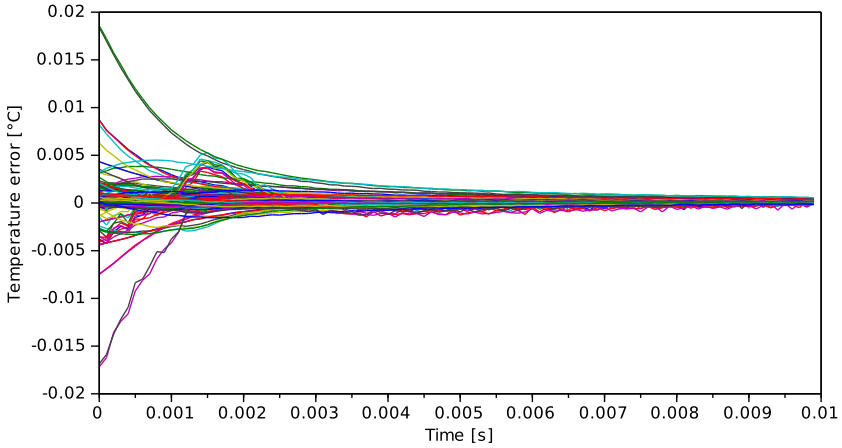

Figure 4.5: *Temperature difference for a 2D chip simulated with both HotSpot and the proposed Modelica simulator.*

4.3.2 Thermal transient analysis

The analysis of the dynamic behavior of the system from the thermal viewpoint represents a critical aspect to be considered during the design of runtime thermal policies. Several previous methodologies focused on cycle-accurate simulation coupled with steady-state temperature evaluation [44, 103] for policy design and tuning, while the thermal dynamic of the system can greatly influence the final obtained results as detailed below. Figure 4.6 shows a transient thermal analysis for a 12-core architecture evaluating the impact of a frequency step down has on the chip thermal profile. The frequency change, from 2GHz to 1GHz around at 0.3s of the simulation, determines a temperature step down in the order of 18°C. Moreover, two different snapshots of the thermal transient state chip are reported at 0.1s and 0.5s of the simulation.

The thermal transient analysis allows for an accurate estimation of the chip thermal profile, which is impossible using averaged temperature methods or steady state approximations. For example, Figure 4.6 reports a peak temperature around 80°C, while after the frequency step down the temperature for the cores is lower than 64°C. It is worth to note that a steady state analysis can not capture the sharp temperature decrease at the frequency step down.

4.3.3 Thermal-performance policy assessment

This section shows how the proposed simulation flow, thanks to its thermal simulator, can be used for the implementation and assessment of thermal control policies. The aim of this section is not to propose a novel thermal control policy, as this is the purpose of Chapter 5, but rather to show the ability of the simulation flow to support thermal policies thanks to a DVFS capable cycle-accurate simulator providing the necessary actuator and a thermal model allowing to have the chip temperature as a function of simulation time and thus model temperature sensors. For this test the thermal policy implemented is the one proposed in [30], which proposed a PI-based control scheme exploiting a DVFS module for the actuation.

Figure 4.7 reports the thermal evolution of the cores when the PI control scheme is employed on a per core basis. In particular, three temperature set points, i.e. 78°C, 70°C and 75°C, have been selected to test the implementation and the efficacy of the proposed control scheme. Again, it is worth to note that we are not dealing with the effectiveness of the proposed thermal policy, while we are interested in the integration of it inside the proposed simulation flow focusing on two aspects. First, we explored a 100us sam-

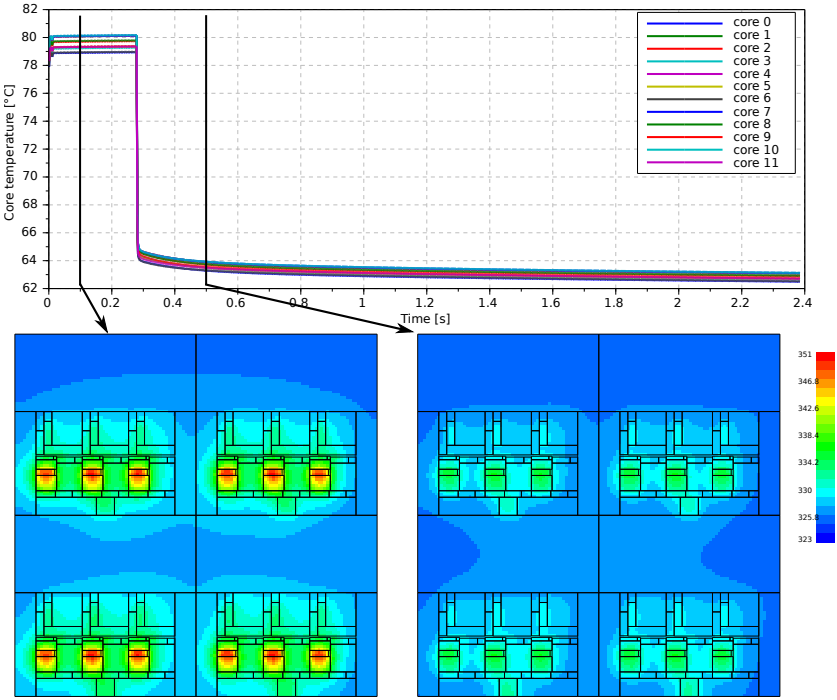

Figure 4.6: *Thermal transient behavior of a 12-core multi-core considering a frequency step-down from 2GHz to 1GHz at 0.3s of simulation. Two thermal snapshots are reported to highlight the flexibility of the proposed flow to compute transient temperature analysis.*

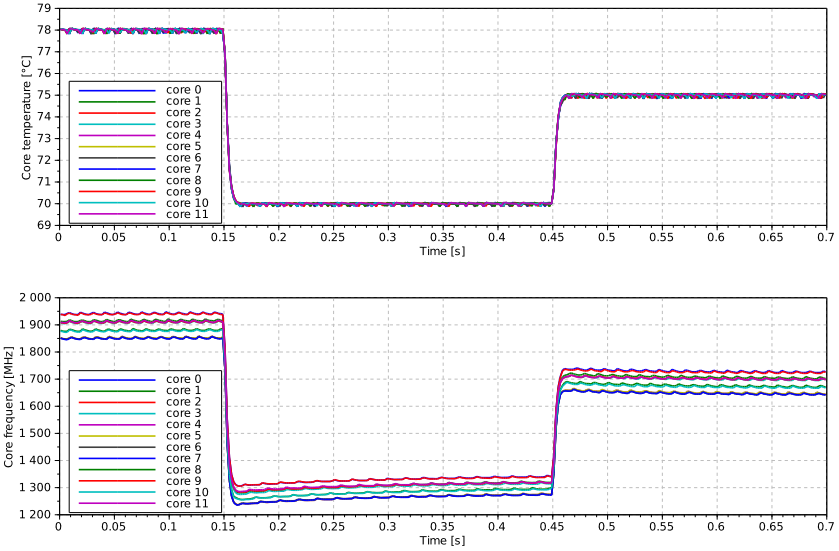

Figure 4.7: *Evolution of the temperature on the cores using the control-based scheme proposed in [30] for each core.*

pling/actuation rate for the policy, while the original work proposed it at 28us. In particular, such exploration allows to evaluate that the policy can efficiently track the temperature set point even if a longer interval between two actuations is used, thus relaxing implementability constraints. On the other hand, the original work does not accurately model the DVFS module, i.e. they accounted for a 28us timing overhead for each frequency/voltage transition, while we can ensure an accurate behavior of the DVFS module from both power and timing viewpoints. Moreover, the transient analysis exploited by the presented simulation flow allows to track the undershoot in the temperature response. This evaluation enables the possibility to better tune the regulator. For example considering core 2 in Figure 4.7, at 0.15s the temperature set point changes from 78°C to 70°C, with a frequency variation from 1.95GHz to 1.30GHz due to the PI controller. However, the controller continues to modify the frequency depending on the evolution of the temperature in the chip.

4.4 Conclusions

This chapter presented a novel thermal simulator, whose accuracy has been demonstrated to be the same of HotSpot, built with an object-oriented, component-based formalism. This allows for a component-level validation, and permits to construct models of new architectures straightforwardly. This thermal simulator is integrated in a cycle accurate simulation flow, being thus able to simulate in a detailed way current and future generation MPSoCs. The inherent flexibility of the component-based approach allows to simulate 2D as well as 3D die-stacked chips. It thus results in a step forward in terms of usability end extensibility with respect to the state of the art.

CHAPTER 5

Addressing the thermal control needs of future MPSoCs

Power is nothing without control.

THE increasing need for computational power is driving the semiconductor industry towards aggressive technology scaling, resulting in a continuous increase in the number of transistors per chip. However, even though technology scaling reduces gate delays, wiring and interconnect delays do not scale equally well [18]. In this *scenario*, 3D-stacking is emerging as a viable solution for some of the major limitations of 2D chips. 3D-stacking partitions the design in multiple blocks, each implemented in a separate silicon layer. Said layers are interconnected by means of on-chip vias. This increases compactness, thus reducing power and latency [94]. As a result, 3D die stacking is a promising solution to increase MPSoC performance, thanks to the new design possibility offered by the increased circuit density achievable.

3D stacking allows splitting the design of CPU cores between multiple layers, to reduce global wiring delays within the core [19]. An effective

use of this techniques allows architectural changes of great impact such as the reduction in the number of pipeline stages without sacrificing clock frequency. Another use is the design of a 3D MPSoC with multiple layers of cores stacked atop of each other. In this case, the performance improvement comes from the reduction of inter-core interconnection delays. As an example, 3D NoC designs [13, 97] show improved performance due to the reduced network diameter and thus reduced average number of hops per packet. Stacking multiple cores can be combined with other improvements, such as increasing cache sizes [35] to further boost performance. At the other end of the spectrum, it is possible to make use of 3D stacking by dedicating an entire layer to memory or caches [89], leaving all the cores on the other layer.

On a different but related front, 3D stacking dramatically exacerbates thermal issues with respect to planar designs. This is because, although the reduction in global wire length and the associated need for repeaters reduces the overall power consumption, power is being dissipated in a smaller area compared to a planar design, thus increasing power density. Moreover, all but the last silicon layer are no longer in direct contact with the heat spreader and the thermal dissipation stack, so it is more difficult to effectively dissipate the heat produced in the additional layers. This worsens the formation of localized, sub-millimeter hot spots, which can degrade long-term reliability and, in extreme cases, cause immediate failures.

In such a perspective, innovative and possibly cost effective thermal management techniques are required, with a synergistic action of the hardware and the software layers [9] to fully exploit the benefits of 3D stacking. Thus, thermal management is nowadays a first-class concern for current and future 3D multi-cores. For example, several proposals aim to optimal task placement to reduce thermal gradients [87], while other techniques rely on reactive [31] as well as on predictive strategies [10, 100] to manage the thermal chip profile.

However, the chip thermal behavior strongly depends on the activity of the cores, which in turn is affected by the executed code, the interaction with other tasks and even user inputs. In this respect, on-chip power management and increasing performance demands increase the non-uniformity of power dissipation across the chip. This results in so abrupt and large power changes to produce significant – and potentially dangerous – temperature variations on a millisecond timescale in 3D chips, as the following experiments will confirm. It is nonetheless worth noticing that the problem of fast thermal dynamics is not limited to 3D chips, as even 2D chips exhibit quite fast dynamics on a tens of millisecond timescale or less [73],

thus the need for effective thermal control policies arises also in this case. Another important fact is that any thermal control scheme would need to operate faster than the thermal dynamics of the controlled system to be effective, five to ten times faster being reasonable values. Thus it follows that any controller for 2D chips has to be able of reacting in a few milliseconds at most, while for 3D chips the requirement is even stricter, with reaction times in the order of hundreds of microseconds. This fact requires careful consideration for both 2D and 3D chips, first of all because reliable predictions of the core activities at such a time scale are hardly possible, which rules out predictive approaches and calls for reactive policies.

In addition, the mere fact that the controller reaction times needs to be in the order of a few milliseconds or less, raises overhead concerns for the thermal control policy. To see why operating thermal control policies at a timescale of a few milliseconds can be a concern for 2D chips, consider that the current trend is to avoid as much as possible periodic interruption of tasks, as the impact of interrupts on deep pipelines is significant [49]. An example of this trend is the effort that the Linux kernel development community is undergoing to move towards a fully tickless kernel, thus getting rid of a periodic 10ms interrupt used for timekeeping within the kernel [45]. Such effort would be spoiled by a thermal control scheme requiring a periodic execution of a software control policy at a timescale of a few milliseconds. As a result, thermal control policies are often moved from a software implementation to a hardware one, a recent example being the scheme used by Intel for turbo boost 2.0 [77]. For a 3D chip the problem is even worse, a traditional control scheme operating at a fixed rate can practically *only* be implemented entirely in hardware to avoid imposing an excessive overhead on the cores it has to control.

However, implementing thermal control in hardware severely affects the flexibility of the policy, hampering the tweaks that may be required to adapt it to the different products the MPSoC is to be integrated into, such as a desktop or laptop computer.

To preserve flexibility, it would thus be beneficial to devise an *adaptive* thermal management policy, combining fast reaction to critical temperature changes with low overhead, achieved by reducing the intervention rate when temperature is either low or relatively constant. If the controller is no longer forced to be periodically activated at the maximum required rate, its implementation can be moved back to the software layer. In the *scenario* just envisaged, thus, the hardware layer has only to provide event generation functionality, which do not depend on the specific system, while the policy can be implemented in software, allowing purpose-specific tuning

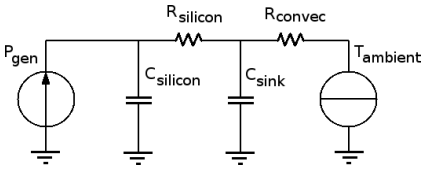

Figure 5.1: *Simplified model of a chip thermal dissipation stack to evidence the two thermal dynamics. This model is used only to give an intuitive explanation of the existence of two separate thermal dynamics, the model identification has instead been performed on the full model of Section 4, detailed in Figure 4.3*

even after its deployment.

This chapter proposes a novel reactive, decentralized thermal management methodology [55, 85] exploiting the theory of event-based control [7, 92]. The self-adaptation of the time between two control events, that is based on the actual thermal profile and the goal of the regulation, represents a key aspects of the proposed methodology. As will be shown in the following, the proposed control scheme provides a temperature control quality comparable to other fixed rate control-based strategies, that however must be implemented in hardware due to overhead constraints.

The proposal has been validated through detailed simulations on a 3D-stacked multi-core which represents the most critical *scenario* from the thermal viewpoint, where accurate control of the fast thermal dynamics becomes of vital importance for the operation and long-term reliability of the MPSoC.

However, the proposed scheme can also be used for 2D-chips when there is the need to control its fast thermal dynamics, since it is not tailored to a specific die-stacking topology.

5.1 The thermal dynamics in an MPSoC

As anticipated, the existence of fast thermal dynamics in an MPSoC is a key motivation for the adoption of event-based control to achieve an effective thermal control with a low overhead while retaining the flexibility of a software implementation. This section is intended to further support this claim showing how in commercial MPSoCs the thermal dynamics are already fast enough that an event based controller would be beneficial, while in future 3D MPSoCs it would become an enabler for the implementation of thermal policies in software.

The thermal dynamics of an MPSoC show at least two separate time

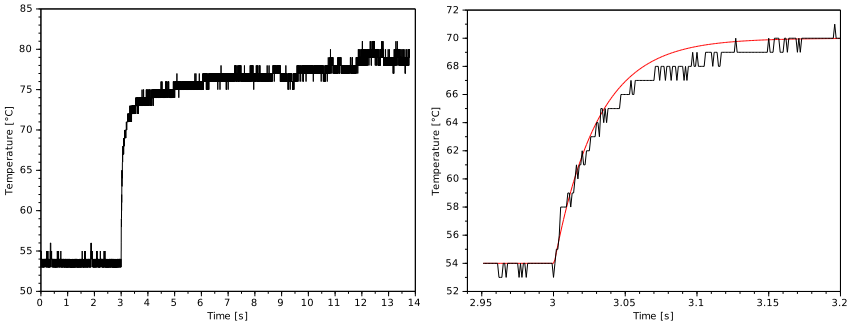

Figure 5.2: *Temperature transient of one of the cores in a Core i7 3630QM processor caused by a step power increase. Left: full transient, right: zoom showing the fast thermal dynamics and single pole approximation.*

scales: a slow one, in the order of seconds to a few minutes, and a fast one, in the sub millisecond to tens of milliseconds range [47, 73]. Figure 5.1 shows an oversimplified RC model of a chip thermal dissipation stack with the only aim of giving an intuitive explanation for the existence of two separate thermal dynamics. Starting from the left, we find the generator modeling the power dissipation within the chip. The silicon layer has a thermal capacity $C_{silicon}$ which is small due to its limited mass. Moreover, the chip is connected to the thermal dissipation stack through a non-negligible thermal resistance $R_{silicon}$. This resistance is in most part that of the silicon bulk, as the heat produced by the active layers has to traverse the bulk to reach the heat spreader, and silicon is not as good a thermal conductor as copper, which is the material usually employed for the heat spreader. Then there is the heat sink thermal capacity C_{sink} which is orders of magnitude larger than that of the chip, and finally the heat sink exchanges heat to the ambient through convection.

Due to the coupled effect of the small thermal capacity of the silicon and of the non-negligible thermal resistance connecting it to the heat sink, the chip temperature, can "swing" very rapidly – as the load dictates – with respect to the temperature of the heat sink, thus producing the aforementioned fast thermal dynamics. The slow thermal dynamics, quite intuitively, is conversely that of the heat sink.

To further evidence the presence of the two thermal dynamics, as well as measure their thermal time constants, an experiment on real hardware and a thermal simulation are here reported.

5.1.1 Thermal dynamics in a commercial 2D multicore processor

This experiment consists in logging the temperature of a multicore processor during a load step, such as the one seen as a machine transitions from the idle state to a 100% CPU utilization. The experiment has been performed on an Intel Core i7 3630QM quad core processor running Linux. This processor has a per-core temperature sensor with a 1°C resolution, which allows to monitor the temperature of each core with sufficient detail. A modified version of the `coretemp` Linux kernel module was used, coupled to an userspace C++ program to log the temperature of the cores every millisecond. The need for a modified kernel module arises from the default one having a software capping on the polling rate for the temperature sensor of 10ms, too coarse to accurately see the fast thermal dynamics. The `stress` tool was used to load the cores at the same time at $t = 3$s starting from the temperature trace.

Figure 5.2 shows the resulting transient for core 0. As can be seen in the left plot that reports the full transient, temperature rises from 54°C to 70°C nearly instantly. At this timescale the fast thermal dynamics are so fast that look like a step increase in temperature. After that, temperature continues to rise till 80°C as the heatsink heats up. The left part of the figure conversely shows a zoom on the first instants of the thermal transient evidencing the fast thermal dynamics in greater detail. The black line is the sampled temperature, while the red line is a single pole approximation with a 30ms time constant and 16°C gain. This means that the transition of one core from deep sleep to active will cause a short-term temperature increase of 16°C. Recall that a controller would need to be operated at least five to ten times faster to prove effective in reacting to thermal transients, thus a fixed-rate controller would need to be operated at a 3 to 6ms period, which is both faster the typical period of state of the art thermal control policies (10ms, see [98, 100]) and the default Linux kernel tick [45]. Notice how this processor has the turbo boost 2.0 feature, thus thermal management is done in hardware. This shows how even current MPSoCs would benefit from event-based thermal control, as it would allow to bring back software-based thermal control and thus the flexibility it entails.

5.1.2 Thermal dynamics in a simulated 3D MPSoC

While such considerations hold for a 2D chip such as the Intel multicore, a 3D chip provide higher power densities, causing the fast thermal dynamics to be of considerably higher amplitude compared to a 2D MPSoC.

To assess the thermal dynamics of a 3D MPSoC, a 24 core chip with 12

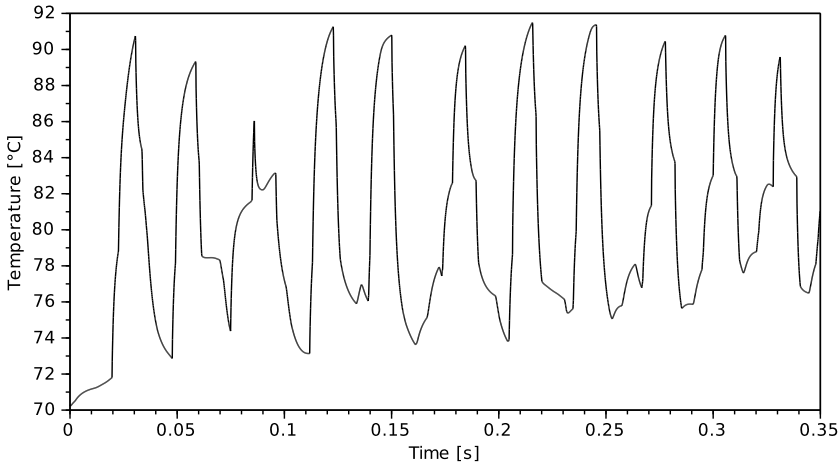

Figure 5.3: *Temperature of the hot spot of one core executing the* sha1 *MiBench on a 24-core 3D-chip.*

cores per layer has been simulated using the simulation flow proposed in Chapter 3. The architectural details of the MPSoC are postponed to Section 5.4, as this MPSoC is the same that was used to validate the proposed event-based thermal controller.

Figure 5.3 shows the thermal profile of the hotspot of a core in the 24-core 3D MPSoC running the *sha1* MiBench [42]. High temperature variations over time can be observed, strictly correlated with the application activity. In particular, temperature decreases occur when the CPU is in a low power state waiting for a DMA transfer from the disk, while high temperature peaks correspond of intense CPU activity due to data processing. Furthermore, Figure 5.3 highlights the two separate time constants that compose the thermal dynamics. In particular, the fast one causes abrupt temperature variations each time the CPU enters/exits a low power state. For example, between 0.112ms and 0.125ms the temperature changes from 73°C to 91°C, which means a gradient of 9°C/ms. The slow one conversely appears as a slow increasing drift, totaling approximately 6°C over one second — thus, a rate 1500 times lower than that of the fast dynamics. Using model identification techniques allowed to obtain a single pole approximation of the fast thermal dynamics, having a 2.5ms time constant. Again, note that to control a thermal process with a 2.5ms timescale the controller would need to be operated at less than 500μs. For what concerns the gain,

due to the thermal coupling between two vertically placed cores, temperature rises by 25°C within 10ms if only one core is at 100% load, and nearly 50°C if both transition from idle to full load at the same time.

It should be noted that the dynamic power consumption of a core depends on the executed code, and in particular on the CPI [10], thus if the executed code has a high CPI the temperature increase will be lower. The static power consumption is instead dependent on the core operating voltage, and on its temperature as well. It is worth mentioning that the identified single pole model for the 3D chip is the one that will later be used for control synthesis and is described in the Laplace domain (s being the corresponding complex variable) as

$$\Theta(s) = \frac{25}{1 + 0.0025s}(U(s) + D(s)) \tag{5.1}$$

where $U(s)$ is the Laplace transform of the control action (the DVFS command), $D(s)$ that of the load-originated disturbance, and $\Theta(s)$ that of the temperature. This model will later be used as the basis for control synthesis.

5.1.3 The need to control the fast thermal dynamics

As can be seen, the thermal time constant in a 3D chip can be lower than the one in a 2D chip. To explain why, it must be considered that the added silicon layer has to be thinned to allow for the addition of through silicon vias to the manufacturing process. Thus, it adds a limited amount of thermal capacitance to the design. Moreover, the bulk thickness of the last silicon layer has to be selected based on a tradeoff between thermal dynamics and steady state temperature. This is because increasing the bulk thickness increases the thermal capacitance of the chip, thus slowing down the thermal dynamics. However, it also increases the thermal resistance towards the heatsink, thus resulting in a higher steady state temperature, an important matter when this resistance has to be traversed by multiple layers of active components, not just one as in a 2D chip design. It is because of this tradeoff that it is not surprising that a 3D chip can have faster thermal dynamics than a 2D one.

This experiment highlights an important fact that thermal policies aimed at future generation MPSoC need to take into account: that more than half of the steady state temperature increase during a CPU load step happens in the first instants following the power increase. Thus, the fast thermal dynamics may cause a highly loaded processor to exceed safe operating temperatures in just a few milliseconds. Hence thermal policies that act on a timescale of tens of milliseconds or more, such as [40], can only control

the *slow* thermal dynamics, and are not suitable for 3D MPSoCs, where controlling the fast thermal dynamics is vital for the reliability of the device.

As the presented experiments have shown, thermal policies aimed at future generations of MPSoCs will need to sense temperatures and act on the control knob (the core frequencies and voltages) at a sub-millisecond timescale for 3D chips, and a a timescale of a few milliseconds in 2D ones. Thus, if a traditional fixed-rate policy is to be executed in software, even as a kernel ISR, the *necessary* intervention rate results in an unacceptable overhead, especially considering the effect of interrupts on deep pipelines [49]. Summarizing, the need for an extremely prompt control action in the face of highly unpredictable, abruptly varying loads capable of producing thermal stresses like that of Figure 5.3, provides a practical motivation for the proposed approach.

5.2 The proposed control scheme

As anticipated, the time constant of the thermal dynamics dictates the appropriate sensing and actuation rate at which the thermal controller needs to be run. This rate needs to be at least five to ten times smaller than said time constant [96], resulting in a values from 200μs to 500μs for the 3D chip that will be used for the validation phase. It is important to stress that while this constraint is due to the physical process, it remains the same *regardless of the control scheme being used*. Having this in mind, briefly discussing two representative *scenarii* straightforwardly evidences the limitations of fixed rate control solutions.

The first *scenario* considers a control scheme running at a fixed time step of 10ms, such as [98, 100], thus violating the constraint above. After the controller has computed the DVFS command at a certain time t_0, this is kept constant until $t_1 = t_0 + 10$ms. If the power consumption suddenly increases say at $t_0 + 1\mu$s, since the controller *structurally* cannot react before t_1, at that time temperature may have increased up to about 50°C (an occurrence of this is shown in Figure 5.10 later on, around 0.6s). The second *scenario* considers a control design that conversely matches the sample step requirement. In this case, trading temperature quality control versus overhead, becomes critical. In any case, given the 500μs sample rate requirement, at least 2000 interrupts per second have to be accepted, which can be a problem *per se*.

Also, with *any* fixed sample rate – abiding by the constraint or not – the obtained controller has a limited flexibility with respect to the application behavior. For example, consider an application with a highly variable ac-

tivity in certain moments, and a more uniform one in others. With a step close to 200μs, when the application has a uniform activity, the controller will uselessly compute the same actuation value at each step, while with a a more relaxed rate, reaction to some abrupt activity changes may not be timely enough. Incidentally, apart from the difficulty of obtaining reliable activity forecasts at the necessary time scale, predictive schemes also exacerbate the flexibility limitation above owing to their computational burden.

Summing up, delivering good control quality at the required time scale, and in the presence of abrupt and unpredictable activity variations, requires reactive policies with self-adaptation of the time between two subsequent control actions; exploiting the event based control theory, as done in the following, is thus a natural solution.

5.2.1 Event-based control

The main idea behind event based control is to design a controller that is not meant to be operated periodically, rather only when an "event" happens. This allows to spend the overhead of running the control algorithm *only when needed*, instead of having a fixed and periodic activation time. By applying the event based theory, one therefore obtains an adaptive scheme that dynamically changes the controller parameters to compensate for a non constant calling interval. Moreover, the same theory allows to rigorously guarantee important properties [54] such as closed loop stability, as well as to keep the performance close to that of a fixed rate one.

Coming to the application of the theory to our case, events are generated according to the so called *send on delta* policy, based on two conditions:

- *timeout event*: an event is generated when a timeout occurs;
- *fast temperature change*: an event is generated when a temperature change between the actual temperature collected by the sensor and the one at the last control action, is greater than a specified threshold value.

The simplicity of the event generation mechanism naturally suggests a hardware-software partition where the event generation policy is implemented in hardware, as a state machine that generates interrupts based on the temperature sensor readings, while the control policy itself is implemented at the OS level as an interrupt routine, or at the EFI/BIOS level through an SMI interrupt [81]. Such a partition allows to introduce the proposed scheme with minimal hardware modifications during the design of

the 3D MPSoC, leaving the flexibility to implement and fine-tune the control algorithm typical of a software-based run time thermal management policy.

5.2.2 The choice of a distributed approach

Given that a core in a generic 3D MPSoC can have other cores on its side, as well as additional cores above or below it, these can act as disturbance sources heating up the core due to thermal conduction. It is thus a natural question whether a distributed, decentralized (thus with limited information from the neighbor cores) or centralized approach is best for controlling the fast thermal dynamics of an MPSoC.

It should be stressed that from the need for a fast response time needed to react to abrupt thermal transients, it follows that the most lightweight solution that gives adequate results is surely the one to be preferred.

For this reason, an investigation was performed to see whether a fully decentralized solution would be sufficient, as it results in no communication overhead between controllers running on different cores *at all*, backing up towards more complex solutions only if the decentralized approach was proven not sufficient.

For this reason, thermal simulations with a two core MPSoC were performed considering two topologies: one with cores side by side in the same layer, and one with a core above the other, thus in two different layers. This test was aimed at computing the thermal coupling between cores at the steady state. To do so, one core was kept fully loaded while the other was kept idle. Measuring the temperature in the idle core was used as the mean to compute the thermal coupling between the cores. For what concerns the first topology, it was shown that the thermal coupling was negligible, due to the small facing areas making thermal conduction negligible. The thermal coupling between two vertically placed cores was however significant.

To see if a decentralized controller could be sufficient to face the thermal coupling in the latter case, a second experiment was performed, that has the main difference of having both cores under closed loop thermal control. In this experiment, both cores are fully loaded, but their temperature are controlled using two independent PI controllers, thus using a decentralized approach. To induce a disturbance in one of the cores, the temperature set point is modified first in the first core, and then in the second. The result is shown in Figure 5.4. As can be seen, when the temperature in one core due to a set point variation, the closed loop PI controller is capable of keeping the temperature of the other core nearly constant, thus counteracting the

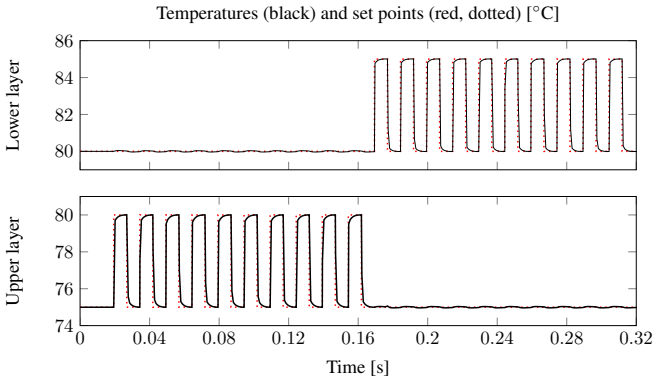

Figure 5.4: *Temperature of two vertically placed cores under PI control, with alternately varying set point, to assess the possibility to adopt a decentralized approach.*

thermal coupling without the need for information exchange between the two controllers.

This is because, despite the high steady state thermal coupling, power is generated in the active sections of silicon layers, that are very thin with respect to the overall chip, and interleaved by much thicker zones of bulk silicon. Thus the capacitance of the intermediate bulk makes the dynamics of the two sections practically decoupled in the band of interest.

It was thus chosen to design the event-based controller following a decentralized approach, as it was proven with these preliminary experiments to be a promising solution that minimizes the communication overhead between controllers. As the validation of the proposed controller in Section 5.4 will show, the proposed decentralized event-based controller is capable of effectively controlling the temperature of a 24 core 3D MPSoC, thus confirming the results obtained through the preliminary experiments here described.

5.2.3 Control synthesis

As the previous section has explained, it was chosen to adopt a control scheme that is entirely distributed, with a per-core feedback loop that do not need to exchange any information with the adjacent ones. This was shown to provide reasonable performance even considering the high thermal coupling between two cores that are one above the other in the MPSoC floorplan, so it was selected as it minimizes the communication overhead.

The control synthesis is here performed along the guidelines proposed

in [54], starting from the thermal model of equation (5.1) whose pole is the one of the fast thermal dynamic, as this is the dynamic that the controller is aimed at regulating. The exact discretization rule is then applied to the process model, in order to obtain a discrete-time representation of it. The resulting discretized process is

$$P(z) = \Gamma \frac{1 - e^{-T_s/\tau}}{z - e^{-T_s/\tau}} \tag{5.2}$$

where z^{-1} is the unity delay operator of the $\mathcal{Z}$ transform, τ and Γ are respectively the pole of the system and the gain, 2.5ms and 25°C in the simulated 3D MPSoC, while T_s is the sampling time. Note that this is obviously not a constant in an event-based controller, so the discretized model of the process is parametric on the inter-event time. The controller is designed by cancellation, including an integral action to have the response to a disturbance step asymptotically reach zero and a gain, μ to tune the controller response, to

$$R(z) = \frac{z - e^{-T_s/\tau}}{\Gamma(1 - e^{-T_s/\tau})} \cdot \frac{\mu}{z - 1} \tag{5.3}$$

As can be seen, the controller explicitly contains T_s, so this is not a transfer function *stricto sensu* but it becomes one only when a T_s is specified, that is, at every event, when the time from the last event is known. This results in a switched control scheme, as the transfer function that is applied at every event depends on the inter-event time. Note, however, that the pole of the controller does not change with T_s, and remains instead fixed at $z = 1$, thus the controller guarantees integral action regardless of the inter-event period. The same controller reads in state space form as

$$\begin{cases} x_R(k) &= x_R(k-1) + \dfrac{\mu}{\Gamma} e_T(k-1) \\ u_R(k) &= x_R(k) + \dfrac{\mu}{\Gamma\,(1 - e^{-T_s/\tau})} e_T(k) \end{cases} \tag{5.4}$$

The loop transfer function is then

$$L(z) = P(z) \cdot R(z) = \frac{\mu}{z - 1} \tag{5.5}$$

that no longer depends on the sampling time. The closed loop transfer function can be computed as $L(z)/(1 + L(z))$ and has a pole in $z = 1 - \mu$. From this, it is possible to obtain the acceptable range of values of the μ parameter in order to preserve the system stability, which is $0 < \mu < 2$. The

most appropriate value of μ was selected to be 0.5 based on a simulations campaign.

To prove the asymptotic stability of the proposed control scheme it can be considered that the following hypotheses in [54] do apply

- *Hypothesis 1* The system to control, which is (5.1), is a SISO (single input, single output) LTI (linear and time invariant) system.
- *Hypothesis 2* These surely exists a continuous time controller that stabilizes (5.1), as it is a simple single pole transfer function.
- *Hypothesis 3* The controller (5.4) is an event based controller.
- *Hypothesis 4* Events are triggered by the sensor only, as will be shown in Section 5.3.1.
- *Hypothesis 5* The inter-event time is quantized, again see Section 5.3.1.
- *Hypothesis 6* The transfer function of the process (5.1) does not contain a time delay.
- *Hypothesis 7* The timeout in the event generation scheme described in Section 5.3.1 provides the necessary upper bound for the inter-event time.
- *Hypothesis 8* The DVFS actuator keeps the frequency constant between controller interventions, thus acting as a zero order holder.
- *Hypothesis 9* The delay introduced by the controller algorithm computation is negligible, being in the order of nanoseconds, see Section 5.3.2, while the minimum inter-event period is in the order of hundred on microseconds.

It is thus possible to apply Theorem 3.1 of the quoted paper, and considering that the system dynamic matrix has as eigenvalues the pole of the controller, plus the pole of the process, it can be shown that it is Schur and with real and distinct eigenvalues for each possible inter-event time from the minimum to the longest possible timeout. Hence, this proves the asymptotic stability of the closed loop system under arbitrary switching.

To complete the controller design it is now necessary to select an event triggering rule, which is the logic that decides when events have to be generated.

5.3 Implementation and Hardware/software partition

This section discusses the hardware/software partitioning of the proposed control, focusing on two different aspects: reducing the computational overhead and increasing flexibility. Starting from the considerations of Sections 5.2.1 and 5.2.3, the obtained formal model of the control loop is stable regardless the event generation policy, which can thus be selected arbitrarily. Moreover, the possibility to generate sensing/actuation events only when required, the reduced complexity of the regulator, highlight the possibility to split its implementation between hardware and software. Specifically, the event generator module is implemented in hardware to interact with the thermal sensors, while the controller synthesized in Section 5.2.3 is implemented in software.

5.3.1 Event generation scheme

The proposed event generator scheme is designed in hardware as follows: the temperature sensor is sampled at a fixed interval q_s, that for the simulated 3D chip can be from 200μs to 500μs, as detailed in Section 5.2. In addition, a temperature threshold *delta* and a timeout are selected. Every q_s, a new temperature value is read, and if it differs from the last value when the controller was run (not the last measured temperature value) by more than *delta*, the controller is run again, otherwise nothing else is done. If instead the timeout expires the controller is forcefully run even if the temperature has not changed by more than the threshold.

The timeout value is also dynamically adapted, to satisfy the opposing constraints of control quality and low overhead. The decision to increase or decrease the timeout depends on the reason why the controller code was called. Every time the controller is called due to a timeout event, the timeout is increased, up to a maximum value that in the proposed implementation is 0.5s. If instead the controller was called due to a threshold event, the timeout is immediately reduced down to q_s, thereby forcing the controller to be run again when the next temperature sample is available. A schematic representation of the logic necessary to implement the event generation scheme is shown in Figure 5.5.

The event generation policy can be implemented in hardware, as a simple state machine connected to a data path to compare the absolute value of the current temperature reading with the one when the controller was last run, hence deciding if an interrupt has to be generated. In addition, the timeout can be easily implemented using a hardware counter incremented at a frequency equal to $1/q_s$. This logic has been implemented and simu-

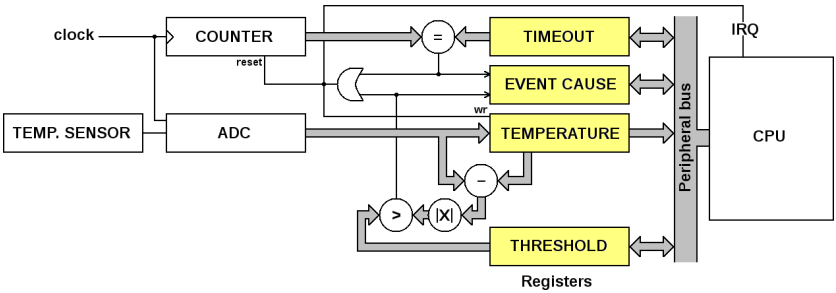

Figure 5.5: *Schematic view of the hardware event generator.*

lated in RTL Verilog, and then synthesized using Cadence Encounter using the NAND Gate Liberty standard cell library. Synthesis results are collected considering an operating voltage of 1.1 V and a clock frequency of 667 MHz, highlighting a per-core area and power overheads of $159(\mu\text{m})^2$, 471μW respectively, thus imposing a minimal impact for virtually any architecture.

5.3.2 Control algorithm

The control algorithm that needs to be executed when an event is generated is outlined in Listing 5.1.

The controller is implemented in the `run()` member function, that takes as input the core temperature `y`, the time since the last event `Ts`, and a flag that allows to differentiate the cases when the controller is run due to a threshold or timeout event. It returns the frequency value to be applied to the core, normalized in the $[0..1]$ range, and the next timeout that needs to be programmed in the event generation state machine. The controller is basically a C++ realization of (5.4) with antiwindup added, as well as the logic to select the next timeout value based on the event type.

The code was benchmarked using `RDTSCP` [14] instructions before and after calling the `run()` member function, and it takes on average 39 clock cycles on a Core i7 3630QM processor, including the function call overhead. Considering that the processor runs at 2.4GHz, the time required to run the controller is 16ns. This places the proposed solution among the fastest thermal controllers, having the added benefit of being event-driven and thus able to increase the event generation period when the temperature is not changing. For comparison, [10] claims that a C implementation of their proposed control algorithm requires 7μs.

Figure 5.6 shows the operation of the event generation scheme coupled

Listing 5.1: *C++ implementation of the event based controller.*

```
class EventBaseController
{
public:
  pair<float,float>
  run(float y, float Ts, bool timeout)
  {
    float e=SP-y;
    float xn=xo+k1*e;
    float u=xn+k1/(1.0f-expf(-400.0f*Ts))*e;
    if(u>1.0f)
    {
      u=1.0f;
      if(xn<xo) xo=xn;
    } else if(u<0.0f) {
      u=0.0f;
      if(xn>xo) xo=xn;
    } else xo=xn;
    if(timeout)
    {
      //This is a timeout event
      index=min(index+1,maxIdx);
    } else {
      //An change event occurred
      index=0;
    }
    return make_pair(u,timeoutTable[index]);
  }

private:
  float xo=1.0f; //Controller state variable
  char index=0;  //Index into timeout table

  const float SP=85.0f;
  const float k1=0.5f/25.0f;
  const float timeoutTable[9]=
  {
    0.0002, 0.0005, 0.001, 0.002, 0.005, 0.01,
    0.02,   0.05,   0.1,   0.2,   0.5
  };
  const int maxIdx=sizeof(timeoutTable)/sizeof(float)-1;
};
```

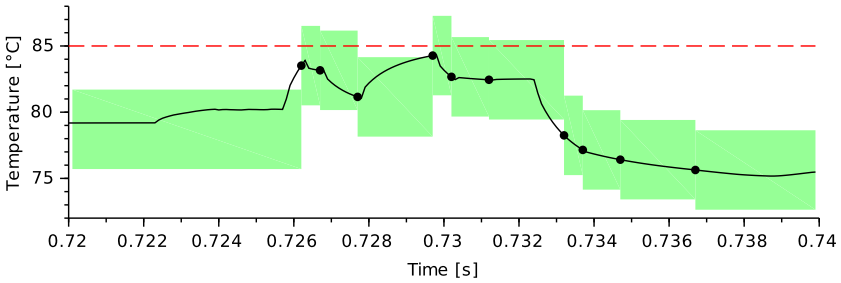

Figure 5.6: *Operation of the event-based controller.*

with the event-based controller. As long as the temperature remains within the range which is determined by the value it had when the controller was last run plus/minus the threshold *delta*, identified in the figure as the green region, the controller is not run until the timeout. Around time 0.726s the temperature exceeds the threshold, and the controller is run (the occurrence is shown as a black dot). The controller computes a DVFS action that prevents further temperature increases, and the timeout is reduced to the minimum value. Subsequent controller activations occur due to timeout events, up to around 0.73s where the temperature exceeds the threshold again. The last part of the figure, starting from t=0.7325s allows to appreciate the progressive increase of the timeout value if the temperature remains within the threshold.

Finally, the control algorithm can be easily implemented using fixed point arithmetic, and the `exp` operator can be substituted with a lookup table. This allows a straightforward implementation in environments where floating point operations are not allowed, such as within the Linux kernel.

5.4 Experimental results and validation

In this section we present a set of simulation results to assess the effectiveness of the proposed event-based control scheme. The simulation flow presented in Section 3 and 4 was used to simulate a 24-core tiled 3D-chip, where the main architectural parameters are reported in Table 5.1. The 3D-chip is composed of two silicon layers with cores on both of them, Figure 5.7 showing the first layer. Cores are split equally between the two layers, and grouped in tiles of three. Each tile has a NoC router to provide interconnection with the rest of the chip. The routers are organized in a 3D-mesh NoC with four routers per layer. The chip has a shared L2 cache composed of 8 physically distributed banks, one connected to each

Table 5.1: *Experimental setup: parameters of the 3D MPSoC.*

Processor core	2 GHz, out-of-order Alpha core
Int-ALU	4 integer ALU functional units
Int-Mult/Div	4 integer multiply/divide functional units
FP-Mult/Div	4 floating-point multiply/divide functional units
L1 cache	64kB 2-way set assoc. split I/D, 2 cycles latency
L2 cache	512KB per bank, 8-way associative
Coherence Prot.	MESI Directory-based 3 Virtual Networks (VNETs)
Router	4-stage virtual channeled wormhole
	64b link width, 4 VCs per VNET
	Frequency variable from 500 MHz to 2GHz
Topology	3D-mesh, 2 layers 2x2 tiles per each layer.
Tile	3 cores, 1 router and 1 L2 cache bank.
Technology	32nm at 1.1V

tile. For what concerns the physical parameters for the 3D die stacking, that are necessary for the thermal simulation, they are taken from [19], a paper published by Intel containing thermal resistances and capacitances of a real 3D stacked processor. Finally, for what concerns sensor placement, a set of benchmarks taken from MiBench were run on the simulated MPSoC, in order to identify the hot spot of each core, which was found to be the register renaming functional unit. A simulated thermal sensor was then placed in that functional unit, and this is the per-core temperature that is passed to the control algorithm, as well as the one shown in the following plots. In the case where a processor has multiple hot spots it would be possible to use multiple simulated temperature sensors, and feed the maximum temperature between the sensors to the controller, although on the simulated core there was no necessity to adopt such an approach.

The simulated architecture was employed to run six benchmarks from the MiBench suite, chosen so as to subject the architecture to varying loads, both CPU bound and I/O bound. Given the short duration of some of the MiBench applications, tests were performed by executing them in an infinite loop, thereby restarting the same MiBench benchmark once it has completed, up to a simulated time span of 1s. Five thermal management policies were tested, namely:

- the stop-and-go policy, based on simple on-off control that halts the core whenever the temperature exceeds a threshold, which is used as a baseline,
- two fixed-rate PI-based policies, with a sampling time of 200μs and 10ms, respectively,
- two event-based PI ones, with an event triggering threshold *delta* of

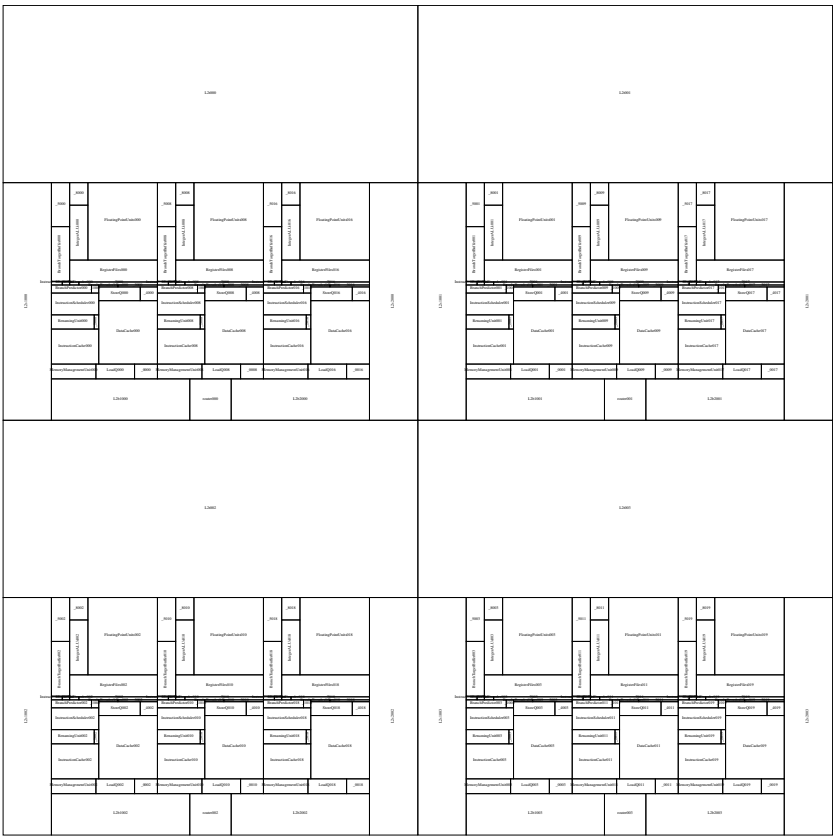

Figure 5.7: *The first layer of the simulated 3D chip showing the four tiles, each composed of three cores, a NoC router and a shared L2 cache.*

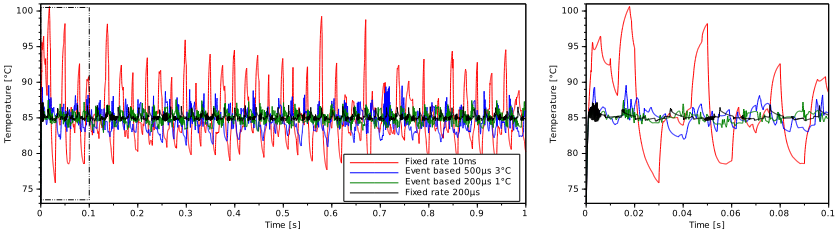

Figure 5.8: *The four tested policies applied to bitcount. Core 0 temperature during the full simulated time (left). Zoom of the first* $0.1s$ *of simulation (right).*

1°C and of 3°C, and a minimum inter-event latency q_s of 200μs and 500μs, respectively.

and in all four cases the temperature set point was 85°C.

It is important to point out that each policy was applied to all six benchmarks without any modification or application-specific tuning, to explicitly test its self adaptation capability, i.e. the ability to withstand heterogeneous operating conditions with a fixed set of parameters.

5.4.1 Thermal control quality

The first set of results is aimed at comparing the ability of the four controllers to keep the temperature set point.

Figure 5.8 shows the temperature trace of one of the cores (Core 0) with the four policies applied to bitcount, the right plot zooming on the area surrounded by the dashed rectangle in the left plot. The bitcount benchmark has been chosen because it is the one that results in the highest power density and is therefore the most difficult to control. The fixed rate 200μs policy apparently provides the best control, although at the cost of 5000 event/s, while the 10ms one definitely fails at keeping the core temperature within a safe limit, exceeding 100°C despite the control set point being 85°C. Both the event based policies deliver acceptable control, with a slight superiority of the (1°C, 200μs) one. Most important, said policies result on average in 320 events per second in the (3°C, 500μs) case, and 1153 events/s in the (1°C, 200μs) one. Therefore the event based policies result in a comparable control quality to the fast fixed rate controller with significantly less controller interventions and thus a lower overhead.

The next results shows the maximum observed temperature for each benchmark and the average completion time of each benchmark when run with a given thermal policy, as a way to consider both the thermal con-

trol quality and a performance metric in order to assess the performance degradation introduced by the policy-induced DVFS action.

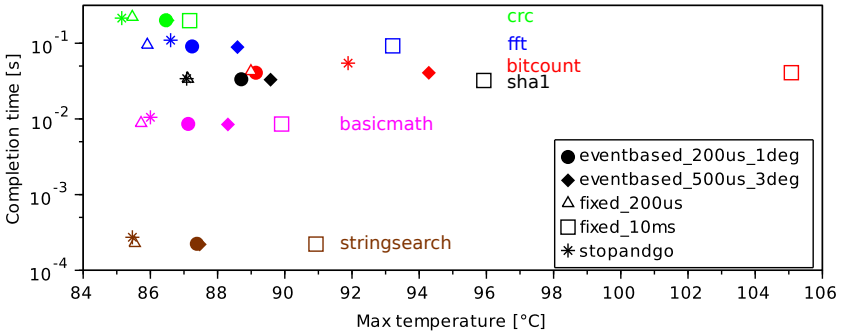

Figure 5.9: *Comparison of the proposed event-based controller with fixed rate policies and stop and go.*

Figure 5.9 reports a summary comparing the maximum observed temperature as well as the simulation times of the executed MiBench benchmarks considering the five implemented thermal policies. The figure shows the results of each benchmark with a different color, while different symbols of the same color are used to identify results for the same benchmark but different policies.

As can be seen, the results of the various benchmarks are placed on the plot on nearly horizontal lines, showing how the policies affect the execution speed of the various benchmarks in a comparable way. The only exception is the stop and go policy, which results in longer completion times, thus a greater performance penalty, as will be shown in further details in Section 5.4.3. Coming to the ability of the policies to control temperature, the fixed rate PI controller operated at 10ms invariantly results in the maximum observed temperature. For example, *bitcount* highlights a peak temperature of 105°C using such a thermal control policy. Thus, a too large actuation period results in a useless controller, that is not able to counteract the fast temperature variations. Conversely, the fixed rate 200μs PI controller provides the lowest temperature in nearly all benchmarks. To this extent, the impact on the computational overhead due to the policy as well as its flexibility represent two key features to select the most valuable thermal management solution. Considering the two fixed rate solutions, they achieve intermediate results in terms of temperature control, with the 200μs solution resulting in a better thermal control and, as will be shown in Section 5.4.3, with an acceptable overhead compared to the fast fixed rate PI.

5.4.2 Detailed thermal transients analysis

Thermal control quality aspects have been investigated in the previous sections where each application runs up to completion in an infinite loop and the disk access latency represents the longer time the CPU is idle. Conversely, in a more realistic scenario applications enter and exit the system continuously, thus imposing great load variations to the CPUs. This section discusses such *scenarii* comparing our methodology with the state of the art in two different use cases. First, the repeated execution of an application interleaved by a variable idle time is discussed. Second, the execution of a CPU-bound application is presented.

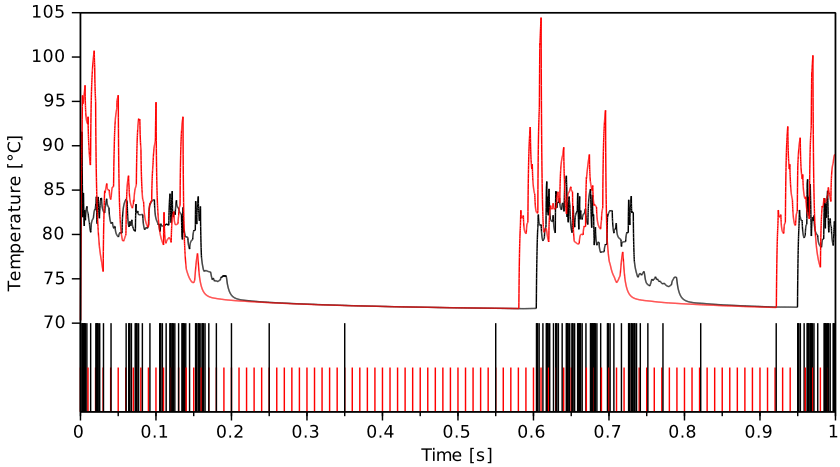

Figure 5.10: *Fixed rate control at 10ms, red, versus (3°C, 500µs) event-based control, black, with the* bitcount *benchmark interspersed with sleeps. Temperature traces are taken from core 0 (a core placed on the corner of the MPSoC).*

Figure 5.9 presents results considering the repeated execution of the *bitcount* application with variable length idle periods between each execution and the subsequent one. In particular, the fixed-rate 10ms and the event-based 3°C 500μs policies are compared. The top part in Figure 5.9 shows the temperature profile, while events are highlighted as vertical lines at the bottom. Observe how the event-based autonomously avoids undue interventions when the temperature is low or just "constant enough" – observe the plot around 0.75s – but at the same time reacts promptly when the load so requires, totaling an average of 152 event/s. On the other hand, the fixed rate controller continuously actuate at its fixed rate even during the periods where the temperature is low, e.g. between 0.2s and 0.6s, while is incapable

of controlling temperature during moments of intense activity (recall that the set point is 85°C).

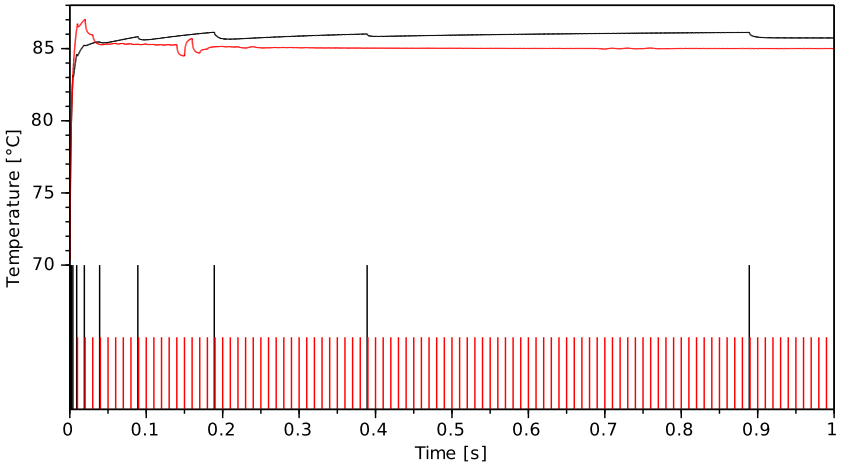

Figure 5.11: *Fixed rate control at 10ms,red, versus (3°C, 500µs) event-based control, black, with stringsearch – core 0 temperature and events.*

The second scenario compares the same policies against a CPU-bound application, *stringsearch*, that forces a constant CPU load. Figure 5.11 compares the performance of the fixed rate control at 10ms with the (3°C, 500μs) event-based controller. The figure shows how, although the core is fully loaded and thus the temperature is stuck at the threshold value, the event-based controller can operate quickly at the beginning to stop further temperature increases, and then progressively reduce the intervention rate. As a result, small fluctuations of the controlled temperature – within the chosen threshold – are paid back by an overhead reduction from 100 to 13 events per second.

5.4.3 Overhead analysis

The last part of this section is dedicated to a more detailed overhead analysis, to emphasize the need for a lightweight policy. When the system to control is a multicore processor, there are two ways a thermal control policy could impact its performance. First, since each core is alternatively used to perform useful work or thermal control, a heavyweight policy, such as one whose control algorithm takes a long time and/or needs to be operated frequently can "steal" CPU time to the applications. Second, a policy

	basicmath	bitcount	crc32	fft	sha1	stringsearch
Fixed 200μs	5000	5000	5000	5000	5000	5000
Event 200μs/1°C	71	1153	387	218	958	28
Event 500μs/3°C	14	320	212	17	472	13
Fixed 10ms	100	100	100	100	100	100
Stop and go	5000	5000	5000	5000	5000	5000

Table 5.2: *Number of control actions performed in 1 second time span by the simulated policies.*

that is too conservative in setting the core frequencies could slow down the applications unnecessarily.

The detailed simulation platform employed allows to measure each of these overheads independently. This is because the thermal control policy is executed in the simulator and not in the simulated cores, thus any variation in the benchmark completion times is to be attributed to the policy acting on DVFS, and to the overhead introduced by the policy activation can be measured separately.

Since all the compared policies require only a few tens of clock cycles to be executed, to measure their relative overhead it suffices to measure their activation rate. It should be stressed that if the comparison were performed with more complex policies, such as MPC-based ones [98], this would apparently no longer be the case. Table 5.2 reports the activation rate of each of the tested policies when applied on the individual benchmarks. Fixed-rate policies obviously have a constant rate, while with event-based control, the rate depends on the benchmark. The *bitcount* and *crc32* benchmarks have the highest variability in the CPU workload, and thus power consumption and temperature swing. As a result, more events are generated, but still four to 20 times less than the fixed rate control at 200μs. On the contrary, for workloads that result in a relatively constant temperature, the event rate is reduced drastically, even to values lower than the fixed rate control at 10ms. These results show how event-based control can be an enabler for implementing a thermal control policy in software with an acceptable overhead.

Table 5.3 conversely refers to the overhead introduced by the DVFS actuation, reporting the average execution time of one iteration of each benchmark. The bold face evidences the policy achieving the lowest time, which is reported in ms, while for better readability, the other (larger) times are shown as percentage increments with respect to the best one. The event-based control policy operating at 500μs achieves the lowest overhead in

	basicmath	bitcount	crc32	fft	sha1	stringsearch
Fixed 200μs	+3%	+3%	+10%	+6%	+4%	+2%
Event 200μs/1°C	+2%	+1%	+1%	+2%	+4%	+2%
Event 500μs/3°C	**8.453**	**40.683**	+1%	**88.872**	+3%	**0.220**
Fixed 10ms	+1%	+1%	**198.840**	+4%	**32.203**	+1%
Stop and go	+24%	+34%	+8%	+23%	+6%	+24%

Table 5.3: *Execution time of one iteration of each benchmark. The execution time is shown in milliseconds for the policy achieving the lowest time, and in percentage with respect to the best time for the other policies.*

most cases, while the other event based control – that results in a lower CPU temperature – has a maximum additional overhead of 2%. In two benchmarks the fixed-rate control policy at 10ms achieves the lowest overhead, but with a definitely poor control of the chip temperature. For example, in the *sha1* benchmark, that policy results in a peak temperature of 95.9°C, despite the set point being 85°C. The stop-and-go policy, finally, causes in an overhead exceeding 20% in most cases. This is not surprising, as this policy employs on-off control instead of the modulating DVFS actuator.

5.5 Conclusions

An innovative thermal control strategy based on event based control theory was proposed, suitable also for novel architectures exploiting 3D stacking.

From the control viewpoint, the main contribution was to structure the problem so that any uncertainty be relegated to the generation of exogenous disturbances for a dynamic system with constant parameters, that can be determined from design data; this allowed to conclude that reactive policies are preferable with respect to proactive ones, particularly as for robustness and controller simplicity. Notably, the problem is stated so as to naturally comprehend both 2D and 3D chip layouts.

On a similar front, a problem-tailored controller structure and synthesis was proposed, that yields a better tradeoff between temperature control and overhead with respect to standard forms like PIs, and to heuristics-based tuning approaches. Furthermore, as another major contribution, an event-based controller realization was introduced; this reduces the control computation effort compared to the other fixed rate control solutions with no loss of accuracy in controlling the chip temperature.

Moving to the technological side of the matter, the devised control solution is particularly suitable to be implemented in real multi-core architec-

tures, thanks to its flexibility and negligible overhead. In fact, the implementation of the controller takes a few tens of clock cycles and the pace at which it is invoked is really limited because of the adaptive nature of the event-based controller. The proposed event-based controller, allows to achieve a level of performance comparable to a fully hardware solution while retaining the flexibility of a software implementation. A proper hardware/software split of monitoring/policy can be sought, providing the operating system with a flexible interface to ease adaptation to different chip architectures.

Simulation results were reported, based on the presented simulation flow, to support the control strategy proposal, and testify its suitability for heterogeneous operating conditions without the need for application-specific tuning. The selected set of benchmarks exercised the ability of the policy to withstand diverse workload requirements, such as both CPU bound and I/O bound *scenarii*.

CHAPTER 6

Power-performance optimization for NoC routers

> Theory is when you know everything but nothing works. Practice is when everything works but no one knows why. In our lab, theory and practice are combined: nothing works and no one knows why.
>
> Unknown

THE many-core revolution allows to increase computational power providing highly parallel architectures to run multiple applications. At the same time the bus-based interconnects show their lacks in supporting parallel application generated traffic. In this perspective on-chip networks are regarded as one of the promising solutions to face the communication issues in such architectures. However, the communication demands continue to grow as more cores are integrated on a chip, and the NoC power consumption can no longer be neglected [20]. Moreover, the power budget is expected to become a significant concern in the dark silicon age, thus the challenge becomes to design high-bandwidth, low latency,

and power-efficient NoCs [70].

One promising paradigm for NoC design is the Global Asynchronous Local Synchronous (GALS) one [69]. Its usefulness stems from the fact that, as the number of transistors per chip keeps increasing, so does the power consumption required to distribute a skew-free clock throughout the chip. The GALS design paradigm partitions the chip in a number of frequency islands that are synchronous on the inside, and asynchronous at the boundaries. This design methodology can be adapted to NoC quite easily, by designing tiles composed of cores, caches and NoC routers, and using asynchronous network links to provide interconnection among tiles [62].

As the design of GALS NoCs requires to introduce resynchronizer circuits or asynchronous First-In First-Out queues (FIFOs) at the tile boundaries to prevent metastability [61], this opens up the possibility to change the frequency of tiles independently. In this perspective, Dynamic Voltage and Frequency Scaling (DVFS) and Dynamic Frequency Scaling (DFS) schemes for NoC routers represent viable solutions to further optimize the power-performance trade-off, by dynamically adapting the power consumption to the application needs. However, most of the proposed literature on this topic relies on heuristics to dynamically change frequencies, thus without guaranteeing any properties with respect to the whole system. Moreover, the proposed methodologies consider ideal actuators without considering their transient behavior or their overhead at all.

This chapter proposes a power-performance control-based scheme for NoC routers, that is able to adapt to the actual network load, ensuring closed loop stability. This result is based on a mathematical model of the frequency to contention relation for a NoC router that can be used to develop power-performance policies making use of dynamic frequency scaling actuators to automatically set the router frequencies depending on the actual load.

6.1 Optimizing the power-performance tradeoff in NoCs with DFS

In a typical NoC-based MPSoC, the cores have private L1 caches, and access the NoC upon cache misses, as well as to operate cache coherence protocols (i.e. to invalidate cache lines). This means that the NoC traffic highly depends on the memory access patterns of the code executed on the cores.

For this reason, the NoC traffic exhibits large variations over time, and can also be significantly low for extended periods of time, when the cache

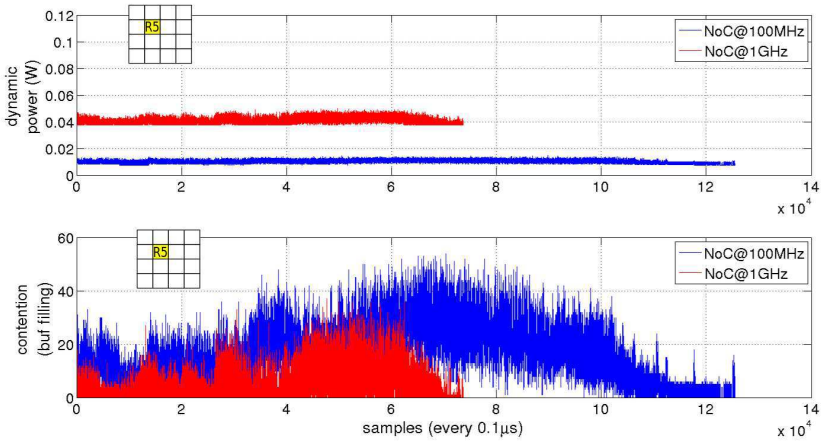

Figure 6.1: *Dynamic power and buffer contention for router five in a 16-core NoC multicore, for two possible frequency values used to clock the NoC routers. The cores run at 1GHz in both cases.*

miss rate is low. This motivates the application of DFS and DVFS schemes to NoC routers to reduce the operating frequency and thus their power consumption during periods of light NoC traffic.

To show the effect of changing the frequency of a NoC router on its performance and power consumption, Figure 6.1 shows the contention level as well the dynamic power for router five considering an 16-core 2D-mesh where the NoC runs at 100MHz (blue lines) and 1GHz (red lines), respectively, while cores run at 1GHz in both *scenarii*, executing the *qsort* benchmark from the MiBench suite [42]. It is obvious that operating the NoC at the higher frequency results in a lower contention level than using the low frequency, while the impact is also on the execution time and on power. The high frequency NoC provides better performance than the low frequency one, i.e. the time required to complete the benchmark are 7.4ms and 12.5ms respectively. However, a router in the high frequency NoC requires five times more dynamic power than the low frequency one. Thus, there is a need for effective methodologies to trade-off power and performance. Moreover, this figure also shows that the contention is not constant and conversely highly depends on the traffic generated by the executed code. This motivates the need for a run-time methodology to optimize power consumption without excessively affecting performance.

In this perspective, frequency scaling represents one of the most used

actuators to achieve such a goal [64]. Although when considering its benefits in terms of power consumption reduction, DVFS may appear a superior choice compared to DFS, in the context of NoC routers power-performance policies, DFS has some notable advantages.

First, the time overhead required to increase the frequency is invariably higher in a DVFS scheme compared to a DFS one. This is an important metric for a NoC power-performance scheme, because the cache access patterns may vary abruptly, and a fast increase in the NoC requests may significantly hurt performance if the NoC frequency is not promptly adjusted to face the traffic increase. The reason for the inherent slower response in DVFS is twofold. Standard DVFS schemes use an interlocking mechanism to avoid operating the controlled voltage and frequency island at a too high frequency and too low voltage, a matter that would cause critical path violations. Thus, when moving to a higher performance state the voltage is increased first, and only after it has settled, frequency starts increasing. This serialization introduces a significant timing overhead. Although this overhead can be improved through coupled DVFS control [4], voltage regulators are often off-chip devices, and due to the necessary output capacitor used for filtering the produced voltage, they tend to have significantly slower dynamics. Thus even if the frequency increase is performed in parallel with the voltage one, the PLL would have to wait for the voltage regulator dynamics, still slowing down a DVFS increase.

Second, the fact that voltage regulators are off-chip devices introduces a cost and board space overhead in implementing DVFS for NoC routers, especially considering that to achieve the maximum benefit in dynamic NoC frequency selection, policies that operate at a fine granularity are preferable, down to a per NoC router control, which would result in a significant number of voltage regulators. On the other hand, PLLs are easily integrated on-chip, and in the dark silicon age where one of the driving design choices is to utilize silicon area to achieve power efficiency [84] it is not inconceivable to adopt a per NoC router PLL clocking scheme. This is especially true considering that PLL can be implemented with a low area occupation. Consider for example [88], where a 2.5GHz PLL was proposed occupying an area of only $0.016mm^2$, despite being implemented in a 90nm process. On the other hand the area of a NoC router in the simulated multicore for the evaluation of the policy, see Section 6.4 is $0.37mm^2$, much larger than the PLL although the router is implemented at 32nm. If the PLL too were scaled at 32nm, the area difference would be even higher. A final reason against using a per NoC router voltage regulator is that although on-chip voltage regulators have been proposed in the literature [52], off-chip induc-

tors still need to be provided for their operation, and their adoption in the industry is still weak.

For this reason, it was decided to concentrate on a DFS scheme as the actuator of choice for the proposed NoC power-performance policy.

6.1.1 Novel Contributions

This chapter presents a framework for power-performance optimization in NoC-based architectures [104], allowing for switching between different policies ensuring closed loop stability at the same time. The work synthetically provides three different contributions:

- *Run-time control-based power-performance NoC framework.* Such framework has two levels. At the lower level a flexible policy to improve the NoC power efficiency is presented which sets the router frequencies based on their contention. Such a policy outperforms threshold-based ones allowing the NoC to operate in different power-performance points. For each of this points stability of the closed loop system is guaranteed. This lower level layer is an inner control loop, and is meant to be implemented in silicon, operating without direct OS intervention. Moreover, the low level enables to switch at run-time between these operating points still ensuring the stability property. This feature can be used by the OS to obtain a more power-oriented or performance-oriented behavior, thus opening the possibility of adopting higher-level policies. At the higher level it is possible to design a policy to optimize the power-performance trade-off based on user requirements. This policy is meant to be implemented within an operating system, changing a parameter of the lower level controller at run time, with a more coarse time granularity (seconds or more).

- *Accurate analytical formulation for both the NoC traffic and control policies* This section proposes a new analytical model for the impact of the router frequency on its contention, i.e. the filling level of router buffers that is used as a proxy for performance [64]. This model has been experimentally proven to be application independent. The stability analysis as well as the run-time policy evaluations have been supported by a complete mathematical formulation for both the controller and the process to be controlled (i.e. the NoC router), as well as the actuator (i.e. the PLL). Then a complete validation has been performed using a cycle accurate simulator while running MiBench benchmarks.

6.2 A NoC router model for power-performance policy design

To design a power-performance policy for NoC routers using DFS, it is first necessary to understand the impact of a the router frequency to its performance. Once such a model is available, control theory can be applied in order to design a proper controller.

6.2.1 Dynamics of a NoC router

This section presents a model relating the performance of a NoC router to its operating frequency, in the form of a linear, time-invariant dynamic system, estimated from simulations performed on real (i.e., non synthetic) traffic flow. In this perspective, the relation between frequency and contention for an on-chip network router is discussed, providing the inputs, the outputs and the internal law which governs the process. The next section exploits this model to design a family of controllers.

The proposed model describes how a frequency change impacts the contention level and is mainly intended to be used for designing fine-grained power-performance optimization schemes that work at the granularity of a single router, or a frequency island composed of multiple routers, by using DFS to change the frequency at run-time. The model was devised starting from a set of physical considerations, complemented by the identification of a single parameter. As a result, it is of the gray-box type [58], with its structure and some of its parameters dictated by the underlying process nature, and the remaining parameters identified based on experimental data.

The first step consists in defining the inputs and outputs of the model that are then bound to the metrics for power and performance for a router. Last, the internal structure of the router model is detailed, i.e. the state equations. In order to enhance the readability four quantities are here defined as follows:

- *power metric and actuator* - The router *frequency* is the chosen the proxy for power. Also, the router frequency is the control input that can be modified at run-time to optimize power.

- *performance metric* - Given a router j, the per router contention metric ($C_{i,j,t}$) is defined as the buffer utilization for router i due to flits that want to go to router j at time t. It is worth noticing that the *local contention* of a router i can be defined as $C_{i,i,t}$ that are the flits stored in the buffers of router i. This last definition can be used as a measure for local performance, as reported in [64].

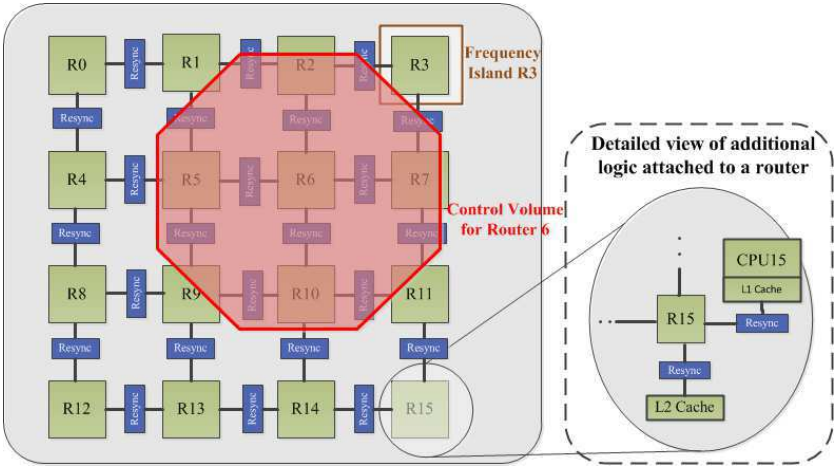

Figure 6.2: *Control volume for Router 6, in the case when each router in the NoC has its own frequency island. The control volume contains all one hop neighbors of the considered router. It includes L1 and L2 links to count injected flits from cores and cache memory.*

- *control volume* (N_i) - it represents the virtual envelope for a router i that contains all its direct neighbors as reported in Figure 6.2. The boundary of the control volume is crossed by all the router links connected to the neighbors of router i, as well as the links connecting router i to its CPU and L2 cache.
- *incoming flits* $InFlits_{N_i,t}$ - it is the number of flits that have been inserted in a control volume within a specific time interval.
- *outgoing flits* $OutFlits_{N_i,t}$ - this is the number of flits that have been processed by router i during a specific time interval.

Starting from the above definitions there are two main observations that can be discussed. First, one could try to use the local contention $C_{i,i,t}$ as the performance metric for router i, since a low contention means low latency due to low conflicts on shared router resources. However, the utilization of the local contention metric is not enough to model the frequency-contention relation on a specific router. Consider for example a two routers *scenario* as depicted in Figure 6.3. The downstream router (r_d) has an initial state, where frequency is zero and all the input buffers empty, while the upstream router (r_u) has a frequency greater than zero and flits stored at its input ports. As at router r_d the local contention $C_{r_d,r_d,t}$ is zero, each reasonable

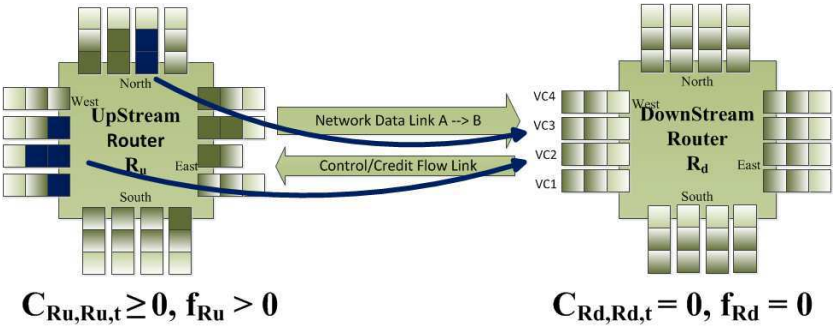

Figure 6.3: *Local contention limitations considering up- and downstream router pair, where the upstream one has flits for the downstream one that can be never sent.*

optimization policy would not increase the frequency to minimize power consumption. However, r_u has both $C_{r_u,r_u,t}$ and $C_{r_u,r_d,t}$ different from zero, since it has flits that want to go to r_d. In such a case flits can never be received by r_d, resulting in a stall situation.

In light of this issue, the contention for a router i is computed counting also the number of flits directed towards that router that are stored in the routers (and CPUs, memory controllers, etc.) one hop before, or, in other terms, the flits contained in the control volume and directed to i.

It must be noted that although having certain NoC routers at a frequency of zero is an extreme case, this model has the advantage of supporting also policies that may make use of this feature. Moreover, this choice for the definition of contention also has the advantage of an anticipated response to a sudden increase in the number of flits directed to a specific router to improve the performance of the control schemes. To this extent the global contention for a router i at time t can be defined as:

$$C^G_{i,t} = \sum_{j \in N_i} C_{j,i,t} \tag{6.1}$$

where $\sum_{j \in N_i} C_{j,i,t}$ represents the sum of the contentions on neighbors of the considered router i due to flits that want to go to router i.

The second consideration is related to the flow balance assumption. For each period the difference between incoming flits and outgoing flits represents the net inserted flits into the control volume. Having a measure of both incoming and outgoing flits becomes important in light of the fast dynamics of the NoC. For example, consider a control volume as the one depicted in Figure 6.2, and a burst of flits that reach the control volume

while no other flits arrive to the control volume after the burst. If the sampling period is short enough the incoming flits can be roughly considered as the flits in the control volume at the sampling time. Otherwise, if the sampling period is significantly lower than the router clock cycle, which is the typical situation encountered when running DFS policies, some of the incoming flits can be served by the router and leave the control volume in between two sampling time points. To this extent actual portion of the flits that impacts the contention can be measured as the net incoming flits, or, in other words, the difference between incoming flits and outgoing flits:

$$Net\_InF_{N_i,t} = InFlits_{N_i,t} - OutFlits_{N_i,t} \tag{6.2}$$

6.2.2 Model identification

Once a suitable contention metric has been found and the balance flow equation has been discussed, the goal is then to derive a dynamic model of the frequency to contention relation. A possible way is to start from physical considerations on the operation of a NoC router, and extend the balance of flits in the control volume by considering the contribution from flits present in the previous sampling period. By following such considerations, the following model was obtained:

$$C^G_{i,t} = C^G_{i,t-1} + Net\_InF_{N_i,t} - \alpha * f_{t-1} \tag{6.3}$$

First of all, the complete model includes saturations not shown here to ease its understanding. Those saturations simply account for the fact that the contention value can never become negative, nor increase above the sum of all buffer sizes in the control volume.

The model contains an integrator, which is easily explained considering that if no flits enter the control volume, and the frequency of the router is zero, the contention remains constant. Also, physical considerations suggest that flits entering the control volume and directed towards the considered router cause an increase in its contention, which explains the $Net\_InF_{N_i,t}$ term. In the proposed model the injected flits are considered as a disturbance, since they cannot be controlled, while the router frequency is the control input.

The frequency impact on contention depends on several implementation details, i.e. the internal router structure and the arbiter policies, however it is reasonable to assume that a frequency increase decreases the contention. Moreover, the impact the frequency has on the contention is function of the actual traffic pattern. Defining the input or/and output serialization as a set

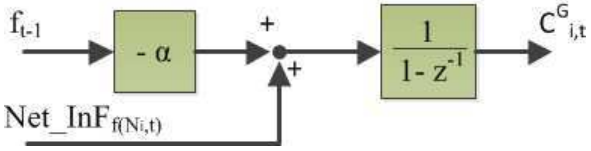

Figure 6.4: *Dynamic frequency-contention model. The frequency is a controllable input, while the injected flits is a non controllable input.*

of flits from a common input port or multiple flits that want to go to the same output port, respectively, the contention due to serialization increases with higher traffic, where collisions are more frequent. In this perspective, an unknown parameter, α, was added in Equation (6.3), to describe the frequency-contention model.

This last step needed to complete the model is therefore the identification of the α parameter based on experimental data. For this purpose, it can be noticed that the proposed dynamic model belongs to the family of deterministic autoregressive models with exogenous input model (ARX) [17, 58]. This allows the use of standard identification techniques to find the α parameter. The resulting ARX model is the one depicted in Figure 6.4.

6.2.3 Model validation

To assess the capability of the model to correctly represent the modeled phenomenon, a test is here reported comparing the estimated contention of a NoC router with the measured contention obtained through a contention sensor implemented in the proposed simulation flow. The presented result is focused on router $R5$ in an MPSoC with 16 cores and 16 NoC routers, while running the *stringsearch* MiBench benchmark. To this extent the router frequency was set at 1GHz using 10MHz as sampling rate for the contention, that is one hundred times lower than the router frequency. Figure 6.5 compares real contention samples (red line) with predicted data using the proposed model (black line). As can be seen, the model can predict the router contention based on its frequency and the $Net\_InF_{N_i,t}$ measure, even at very low sampling frequencies compared to the router clock.

It is worth noticing that the main use of this model is not to predict the router contention, and that the presented test is only used to assess its validity. The main use of this model is to design run-time power-performance policies. In this respect, its main contribution is to provide a model assigning a transfer function to a NoC router, allowing to design a closed loop controller while guaranteeing the stability of the system. When the con-

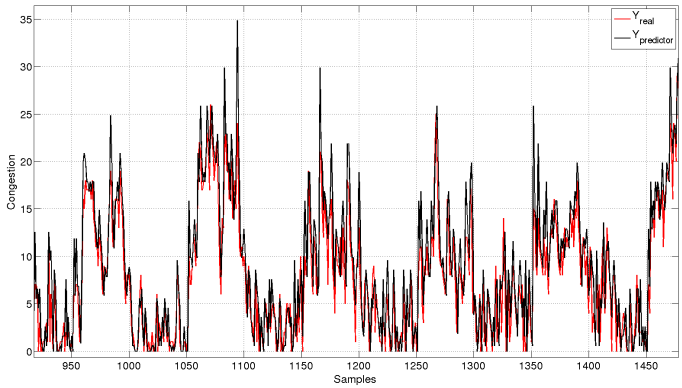

Figure 6.5: *Comparison of the proposed model with measured contention.*

troller is implemented, the router model is *substituted* by the actual NoC router, and the adherence of the model with the actual router behavior allow the controller to operated as expected during the design phase.

The controller has thus to sense the actual router contention, and actuate by changing it frequency. Therefore, the policy that will be presented in the next chapter requires a hardware implementation of a contention sensor for the NoC router, and the need to measure contention at runtime – that as defined previously depends on the control volume – requires communication between routers to exchange the buffer status, resulting in a decentralized control scheme. However, the control loop operates at a significantly lower frequency than the NoC itself, i.e. 10MHz that is 10 to 100 times slower considering frequencies for the NoC between 100MHz and 1GHz. Thus, the requirements for communications are relaxed.

6.3 NoC power-performance controller design

Starting from the model of the real process developed in Section 6.2, this part discusses the design of a control scheme aiming at router contention reduction, while being conservative in increasing its frequency, since it directly impacts on its dynamic power consumption.

The proposed control scheme is the one shown in Figure 6.6. The main idea behind this scheme is to measure contention at run time, and close a control loop around it to control the router frequency. To minimize the control scheme's complexity, which simplifies its hardware implementation, a

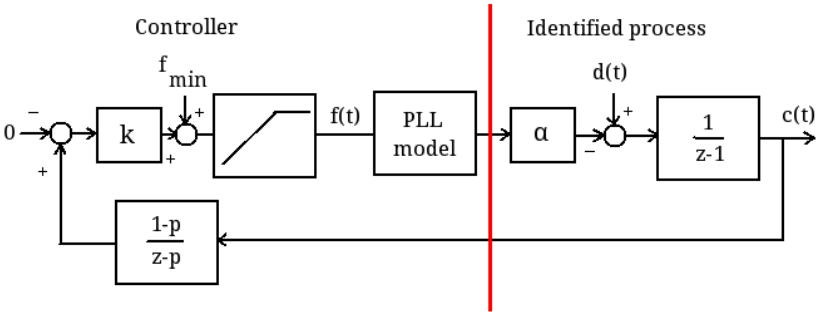

Figure 6.6: *Block diagram of the proposed NoC frequency control scheme.*

proportional control scheme was selected which, as the results will show, provides good performances.

Starting from the left side of the figure, the controller's set point is chosen as the constant zero, as ideally the desire is to have no contention. The set point is then subtracted by the (filtered) measured contention, obtaining an error value that is greater than zero if contention is higher than the set point. Note that the signs at the summation node are reversed with respect to a standard proportional control scheme due to the process having a negative gain, i.e. an increase in frequency causes a decrease in contention. The error value is then multiplied by k, the proportional gain. This parameter specifies how aggressively the controller will increase the router frequency in response to an increase in contention. The resulting value is however a frequency value that is zero with no contention. As it may be undesirable to completely halt the clock of a router, and PLLs often have a limited lock range and thus the frequency cannot be arbitrarily decreased, the computed frequency value is summed to a baseline frequency value, f_{min} to take these limitations into account. The last computation is a saturation at a value f_{max}, used to set a limit on the maximum frequency that the controller will select. The so computed frequency value is then applied to the PLL to actually change the router's frequency. It should be noted that the PLL response to a change in its frequency set point is not immediate, and its dynamics are significantly slower than the ones of the process to be controlled. Therefore, it is not possible to neglect the PLL dynamics during the design of the controller.

As the PLL is the slowest component in the control loop, it also determines the rate at which the controller is to be clocked. The PLL model has been described in Section 3.6, but for the purpose of the control loop

design it is important to note that the chosen PLL for the NoC router has a settling time to a step change in frequency of around 2us. To be able to correctly model its dynamics, the control scheme needs to operate 5 to 20 times faster, with faster sampling rates leading to better discretization of its dynamics and thus more accurate control. Since the range of frequencies at which the control loop has to operate forbids a software implementation, it was chosen to operate the loop with a discretization period 20 times faster than the PLL dynamics, leading to a 10MHz controller operation frequency. This value is used to quantize the continuous time PLL model, which is then used to compute the loop transfer function.

The last component to be discussed is the contention filter. Its purpose is to smooth the contention measure, which from the performed experiments was found to contain undesired high frequency components, caused by the fast traffic variations experienced by the NoC routers. The filter cutoff frequency can be tuned using the p parameter. Experimental results have led to the selection of a value of 0.99.

Once a model for all the components composing the control loop are available, it is possible to compute the loop transfer function, and determine the range of k values that guarantee the closed loop stability. A way to obtain this range is through root locus analysis [58].

Table 6.1 reports α and the maximum k values for different benchmarks extracted from a 2D-mesh 16-core for router 5 using 2 VCs per virtual network. These values complete the control loop model presented in Figure 6.6. It is worth noticing the maximum k values are reported on the root locus stability analysis while the minimum allowed k value is zero for each benchmark, which is equivalent to opening the control loop and always selecting the lowest frequency. Starting from the data it is clear how the product of the two numbers, i.e. α and k_{max} provide always the same results, without considering rounding. As each benchmark has a different identified model, i.e. different α, to to keep it stable in the control loop a different k_{max} is allowed. The selection of the k value when implementing the control loop can however be performed statically, without having to profile individual applications, by being conservative in the k selection.

Although each application results in a slightly different α, a common α value was identified, reported in the *ALL* line of Table 6.1. This value was obtained by performing the identification procedure using the concatenation of all the benchmarks, thereby taking into account a significant variety of NoC usage patterns. Subsequently, the control law was tuned using that value, i.e. not tailoring α for each specific application, without resulting in unstable behaviors. The independence of α from the applications was ex-

Table 6.1: *α and maximum k values for different benchmarks, considering a 2D-mesh 16-core architectures with 2 virtual channels (VCs) per virtual network (VNET). The number refers to router 5.*

name	α	k_{max}	$\alpha * k_{max}$
basicmath	0.847	0.229	0.194
bitcount	1.261	0.152	0.192
crc	1.709	0.113	0.193
dijkstra	0.462	0.415	0.192
fft	2.130	0.091	0.193
patricia	2.575	0.075	0.193
qsort	2.497	0.075	0.192
strsearch	1.433	0.135	0.193
ALL	1.880	0.102	0.192

perimentally proved since the controller provides comparable performance between all the applications, as detailed in Section 3.9.

6.3.1 Higher level control

While the controller presented so far can improve the power efficiency of a NoC by reducing the router frequency during periods of light load, in applications that are tightly power and energy limited, such as mobile devices, it may be desirable to dynamically steer the system to a more performance oriented or low power state based on application or user requirements. Many operating systems, for example, incorporate a feature that switches a laptop computer in a higher performance mode when the power cord is attached. To perform such a feature, the operating system needs access to low level actuators that allow to tune the system performance. Moreover, it is quite common to allow for user interaction to set the desired level of power-performance trade-off thus changing the system behavior at run-time. In this perspective threshold-based policies are not flexible enough to provide this feature as although the thresholds can be reset to different values, such modification requires for a deep understanding of the low level microarchitecture, thus a wrong tuning of the thresholds can steer to a saturated behavior leaving the frequency fixed at its higher or lower value.

This can easily be done with the presented control scheme by exposing the α parameter of the individual controllers as a memory mapped peripheral register, thus allowing the software to dynamically change the controller behavior. This allows the implementation of higher level policies to better fit the required power-performance level.

Table 6.2: *Experimental setup: processor and router micro-architectures and technology parameters.*

Processor core	1GHz, out-of-order Alpha core
Int-ALU	4 integer ALU functional units
Int-Mult/Div	4 integer multiply/divide functional units
FP-Mult/Div	4 floating-point multiply/divide functional units
L1 cache	64kB 2-way set assoc. split I/D, 2 cycles latency
L2 cache	2MB per bank, 8-way associative
Coherence Prot.	MESI
Router	3-stage wormhole virtual channelled switched with 64bit link width 4fl/VC 2/4 virtual channels (VCs) for each virtual network 3 virtual networks (Garnet network [2])
Topology	2D-mesh, based on Tilera iMesh network [95] for link width and NoC frequency (@1GHz)
Technology	45nm at 1.0V
PLL model	ω=4 $\cdot$ 10^6, ξ=0.6, See equation (3.1) worst case power consumption 2mW.

6.4 Experimental results and validation

This section details the results for the proposed NoC power-performance methodology in four different steps. First, the experimental setup is described in Section 6.4.1, while Section 6.4.2 compares the proposed policy against threshold based and fixed frequency ones, to bound the evaluation. It is worth noting that this comparison uses the router contention as a performance proxy, as detailed in Section 6.4.2 to assess the validity of the proposed model that relies on this metric. Section 6.4.3 discusses the control policies applied to the whole system, when simulated time and dynamic power are used as performance and power metrics, respectively. Last, the effect of switching the α parameter at runtime is shown in Section 6.4.4.

6.4.1 Experimental Setup

The proposed NoC power-performance policy is here tested on a representative 16-core 2D-mesh tiled architecture, where detailed parameters are reported in Table 6.2. Each tile is composed of an Alpha 21264 core running at 1GHz, with private L1 cache and a shared L2 cache composed of 16 physically distributed banks, one connected to each router, as shown in Figure 6.10. The architecture was used to test different *scenarii* considering several applications taken from the MiBench suite using the simulation toolchain detailed in Section 3 equipped with the PLL and the DFS actuator models. For each simulation, detailed traces were collected with the dynamic power consumption of each router, as well as latency and contention

Table 6.3: *Evaluated policies organized by classes.*

Class	NameID	Details
Static	**Upper**	Fixed router freq @ 1GHz
	Lower	Fixed router freq @ 100MHz
Threshold	**Th1**	3 switching freq 250-500-800 MHz threshold values buffer fill level 10-20 flits
	Th2	3 switching freq 100-500-1000 MHz threshold values buffer fill level 10-20 flits
we **Control**	**k=0.01**	the scheme presented in Section 6.3, where the NameID field in the table provides the actual k value
	k=0.04	
	k=0.075	
	k=0.15	

providing average measures to allow comparisons between different policies. The frequency value produced by the DFS policy was used as the set point for a PLL, modeled as detailed in Section 3.6, with $\omega = 4 \cdot 10^6$ and $\xi = 0.6$. The PLL frequency range was set between 100MHz and 1GHz, providing a realistic and reasonably flexible environment, i.e. the Tilera iMesh Noc is clocked up to 1GHz [95].

6.4.2 The baseline control policies

To assess the proposed control-based methodology, it was compared against two different classes of policies, *static policies* and *threshold policies*. Table 6.3 reports the details related to the employed policies. The *Upper* and *Lower* policies set the router frequency to 1GHz and 100MHz, respectively. These represent the baseline for all the other evaluations, since are the ones that result in the most performance oriented and power saving behavior compatibly with the chosen architecture. The next two are threshold-based policies, named *Th1* and *Th2*, that allow to switch between three different states, i.e. three different frequencies, using two different threshold levels. Such policies are used as representative threshold-based policies and were selected after a set of experiments against different set of parametrized threshold policies. Last, the proposed control-based solution is used, with four different k values to show the controller ability to obtain a more performance oriented or power saving behavior seamlessly, by changing the controller parameter.

The experiments here presented show how the control policies behave better and are more flexible than the threshold based or baseline static policies.

In this first experiment, the NoC power-performance policies were applied to one router only in the 16-core architecture detailed in Section 6.4.1,

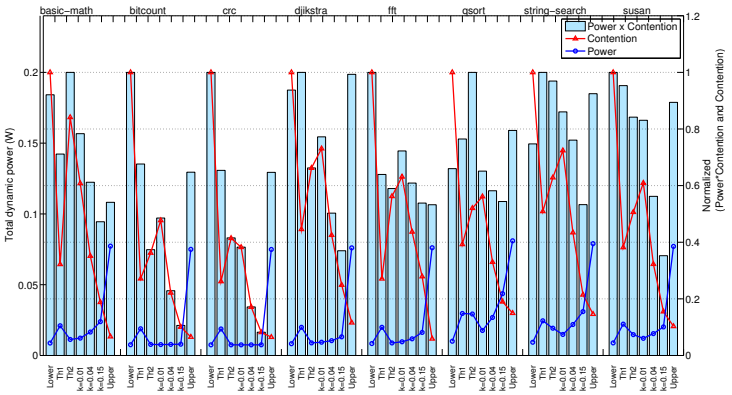

Figure 6.7: *Router 5 average power-contention trade-off considering different policies. Experiments run on a 16-core NoC multi-core, and the frequency of router 5 only is changed depending on the policy between 100MHz and 1GHz. Note that in a 4x4 2D-mesh as the one considered for these results, Router 5 is positioned at the second row and second column of the NoC topology.*

leaving the frequency of the other routers fixed at 1GHz, to better show the effect of the policies. Figure 6.7 shows the results using different benchmarks from MiBench. The selected router is router five, which is in the middle of the mesh, thus has to face higher traffic due to the employed XY routing strategy [63]. The figure reports the six evaluated policies for each benchmark at the bottom x axis, while the benchmark names are shown on the top x axis. The left y axis reports the dynamic power for the considered router, while the right y axis shows the contention and power times contention values. It is worth noticing that contention and the power contention product are normalized to one on a per-benchmark basis to better shape the figure. The figure reports three different measures, i.e. the dynamic power (blue line), the contention (red line) and the power contention product (light blue bars).

The first important remark is that the dynamic power consumption of a NoC router can exceed values in the order of 50mW, while the power overhead of a PLL can be as low as a few milliwatt, one example being [36] demonstrating a 1.35mW PLL capable of operating at 1.5GHz. This proves that the proposed dynamic policy is indeed feasible even considering the PLL consumption overhead.

The policies chosen for the comparison are three control policies, i.e. $k = 0.01$, $k = 0.04$ and $k = 0.15$, against the *Upper* and the *Lower* static

policies as well as the two threshold based policies *Th1* and *Th2*.

The reported results allow for three different considerations. First, the two static policies actually are the upper and lower bound for the analysis, since the *Lower* has the higher contention level for each benchmark, while the collected dynamic power is always the lowest. The same can be observed for the *Upper* policy that provides the lowest contention with the highest dynamic power consumption.

The second observation is devoted to the control-based policies that can explore the power-performance space. In particular, for each benchmark the contention red line reported for the control-based policies always decreases increasing k, since a stronger actuation is applied, i.e. higher frequencies, in face of a contention increase. On the other side also the blue lines for the control-based policies always increases with the k, even though in some cases the increase is so small that is not noticeable in the graph, since a more strong reaction to the contention increase means higher dynamic power consumption. This aspect is evident on the *string-search* bench in Figure 6.7, which has 72% of the contention of the static lower frequency policy using $k = 0.01$, than decreases to 43% with $k = 0.04$ and reaches 21% using a $k = 0.15$. The average power consumption is respectively 15, 22 and 31 mW.

The last point aims to the comparison between the proposed control-based methodology and the other policies. In particular, it can be noticed that the policy with $k = 0.15$ achieves the lowest power-contention product in all but the *fft* benchmark, and even in this benchmark it is only slightly worse than the *upper* policy which is the best one in this case. This shows how it is possible to choose a single value of k that provides optimal or near-optimal results regardless of the application being executed. This is an important fact, as it shows that there is no need to tune the k parameter on a per-application basis, but rather k can be tuned only to steer the system to a more performance oriented or power saving behavior.

6.4.3 Applying the proposed scheme to the whole system

This section shows the policy behavior when applied to the whole system. In this case all the routers are equipped with a DFS module running one of the presented policies. The results are shown in Figure 6.8, which contrary to the previous experiment uses the benchmark completion time as the performance metric, while the policies are obviously still based on contention. This has been done to assess the contention metric as a good performance metric, showing how a policy aimed at reducing contention can impact the

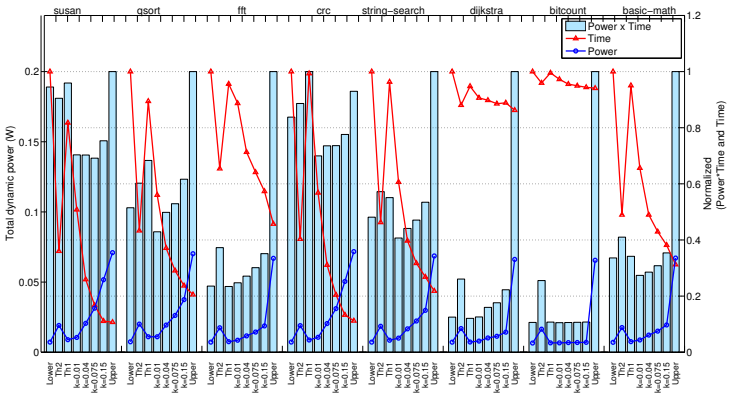

Figure 6.8: *Evaluation of dynamic power, total execution time and their product when all the routers are running the policy.*

simulation time. The left side y axis reports the dynamic power consumption (W), while the right side y reports both the benchmark execution time and the product between power and time, both normalized to one.

Four control policies were evaluate, with $k = 0.01$, $k = 0.04$, $k = 0.075$ and $k = 0.15$, against the *Upper* and the *Lower* static policies as well as the two threshold based policies *Th1* and *Th2*. Figure 6.8 shows the results. The control policies can a provide better trades-off than the static and threshold-based ones. This is testified by the $k = 0.01$ policy achieving the lowest power time product in all but three cases. Even considering those cases, it is still close to the best value, again proving the existence of a single application independent k value that is close to optimal regardless of the application being executed.

6.4.4 The run-time control framework

While Section 6.4.2 shows how the control-based policies can outperform the threshold-based methodologies, it is often a desirable feature to be able to dynamically select a more performance oriented or power saving policy based on application or user requirements. As outlined in Section 6.3.1, since the proposed controller has a single parameter that allows to steer the system to a more performance or power saving behavior, it opens the possibility of being extended with higher level policies, implemented at the operating system level.

This section demonstrates how the possibility to dynamically switch the

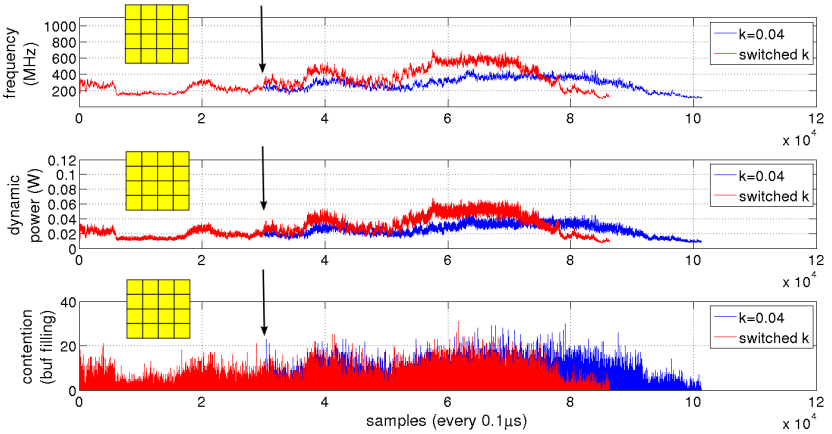

Figure 6.9: *Evaluation of dynamic power and total execution time on all routers running* qsort*, when the policy is changed from k=0.04 to k=.075 after 3ms of simulation.*

controller parameter can increase the flexibility of the system against different user requirements. This however makes the proposed controller a *switched* control system, and the stability property can be preserved if all the controllers in the switch set are stable and a minimum time between two switches is guaranteed, i.e. a *dwell time*. In this perspective all of the controllers we employ guarantee for an asymptotically stable closed-loop system, thus there exist a minimum finite time interval between two switches that ensures the stability of the switched system [102].

Figure 6.9 shows the capabilities of switching the k parameter. It refers to the *qsort* benchmark with all the router running the control based policy. The figure shows the imposed frequency, the dynamic power and contention level on router five only as the other routers have a similar behavior. In particular the blue line reports the data when all routers are running the *k=0.04* policy, while the red line shows the same data while all the routers are running a switched policy that uses *k=0.04* from the beginning of the simulation until 3ms, then switches to *k=0.075*.

The figure allows to evaluate three aspects. First, a change in the policy at 3ms does not produce an unstable behavior. Second, the *k=0.075* provides a stronger reaction to contention increase, since from 3ms to the end of simulation the frequency for *k=0.075* is higher than the frequency imposed using the *k=0.04*. Moreover, the switched policy allows the benchmark to finish the computation sooner, although with a higher power con-

sumption. To this extent we assess the possibility to change the control policy with a simple command from the higher levels to move the system towards a more conservative or more performance oriented behavior. However, the complete development of an optimal switching policy is out the scope of this thesis.

6.5 Conclusions

This chapter presented a complete methodology to trade-off power and performance in NoC architecture exploiting dynamic frequency scaling (DFS). The choice to limit the usage of DFS instead of the DVFS is motivated by many reasons mainly focused on the impossibility to easily integrate many DVFS actuators, i.e. one per NoC router on the same chip. Moreover, the voltage regulator could impose an excessive delay to the reaction of the system due to the NoC load variations, that are usually fast and wide. However, our methodology can be easily adapted to be used with DVFS actuators.

In particular, the work is organized in three steps. First, a model of the frequency-contention for a single router is developed as a dynamic system. Then, a set of controllers have been implemented guaranteeing closed loop stability and dynamic frequency adaption depending on the NoC load. Last, the possibility to dynamically switch the controller parameter has been evaluated, as a mean to adapt the system behavior to system or user requirements. The results compared different control policies against static and threshold policies [64] providing better results both in term of power saving and simulated time.

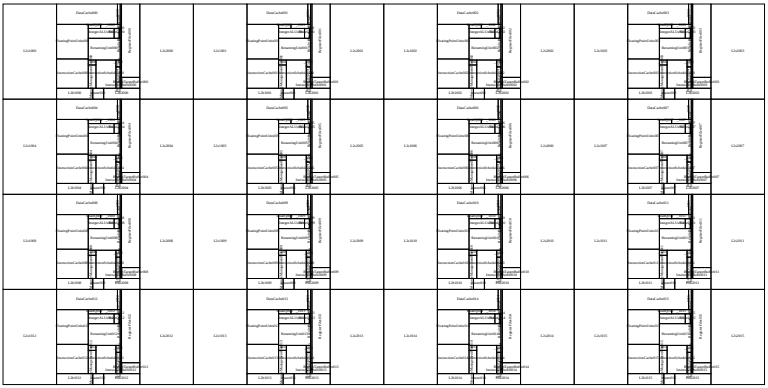

Figure 6.10: *The layout of the simulated MPSoC showing the 16 tiles, each composed of a core, a NoC router and a shared L2 cache.*

CHAPTER 7

Conclusions and future work

THIS dissertation addressed the thermal and energy management problem in MPSoCs. For what concerns the thermal management problem, it was decided to focus on modeling the chip thermal dynamics with the aim of developing a simulation framework capable of simulating future generation 3D die-stacked architectures. In addition, a novel and low overhead thermal control policy based on event-based control theory has been proposed that can effectively control the temperature of a 3D chip considering the thermal dynamics of said architectures.

For what concerns power and energy optimization, a policy has been proposed, taking advantage of the inherent need for resynchronizers in GALS NoC designs to employ DFS on a per-router basis, thus being able to reduce the frequency of individual NoC routers during periods of low traffic. This policy is backed up by a model of a NoC router frequency to contention relation.

Last, a flexible simulation flow has been developed capable of simulating in a cycle-accurate way NoC-based MPSoCs, including 3D die-stacked ones, supporting GALS NoC architectures, and providing microarchitectural sensors, as well as sensors for physical quantities such as power and

Table 7.1: *Summary of the research topics covered in this thesis, evidencing the achieved results.*

Topic	State of the art	Contribution	Validation
MPSoC Simulation	Simulation frameworks for power-performance tradeoff analysis are common [24, 74], while some add steady steady state thermal support [44, 103]. Transient thermal modeling coupled with a cycle accurate simulator is seldom provided, although it is fundamental for assessing DTM policies [11]	A cycle-accurate simulator with support for transient thermal analysis, GALS NoC architecture and per-core as well as per-NoC router DVFS coupled with detailed sensors and actuators including PLL and resynchronization models	Cycle accurate simulations using MiBench benchmarks against baseline GEM5 simulator
Thermal modeling	HotSpot [46] is the most commonly used 2D chip thermal simulator, while 3D-ICE [83] supports 3D die-stacked architectures with a focus on liquid cooling. Both are based on compact thermal models, and embed the system equation and the solver in an unified codebase, written in C	Developed a component-based thermal simulator in an object-oriented modeling language (Modelica). This simulator allows to flexibly combine the models of individual components to build a chip (either 2D or 3D) and the thermal dissipation stack	Comparison with HotSpot
Thermal control	PI-based policies [29] for reactive control, and MPC-based policies [10, 98] for predictive approaches. In both cases policies are operated at a fixed rate regardless of the thermal state of the chip	Application of event-based control theory to the thermal management problem. The resulting policy operating rate is adaptive thereby coupling a fast reaction time with a low overhead. Such a policy can counteract the fast thermal dynamics found in 3D chips while retaining the flexibility of a policy implemented in software	Compared to fixed rate PI policies using MiBench benchmarks
NoC power-performance	Threshold-based policies employing DFS [64] to adapt the router frequency to the NoC load	Developed an analytical model of the frequency to contention relation for a NoC router, and developed a policy based on proportional control	Compared to threshold-based policies using MiBench benchmarks

temperature, as well as detailed DFS and DVFS actuator models and being thus able to simulate thermal and power-performance policies in a realistic setting.

Table 7.1 summarizes the novel contributions of this dissertation, with a short description of the state of the art, the developed improvements, also mentioning the validation methods.

Possible future directions that could start from the work done and extend it include.

- Extending the simulation framework with more clock resynchronizer schemes, such as FIFO-based solutions, and a comparative study of the resynchronization overhead of the implemented synchronization schemes.
- Extending the proposed thermal simulator with additional interchangeable components such as thermoelectric coolers and heat pipes, in order to model and explore different thermal dissipation solutions for MPSoCs. Also, it would be interesting to model the case in which the MPSoC is being housed, as well as simulating the PCB on which it is mounted together with other components that generate heat, such as memories and voltage regulator, moving thus from a MPSoC-only thermal simulation to a full system thermal simulation.
- The implementation of the proposed policies in commercial and research prototype architectures, in order to assess their benefit outside of a simulation environment.

Bibliography

[1] H. Adrang and H.M. Naeimi. A type III fast locking time PLL with transconductor-C structure. In *Circuits and Systems, 2009. MWSCAS '09. 52nd IEEE International Midwest Symposium on*, pages 58–61, Aug 2009.

[2] N. Agarwal, T. Krishna, Li-Shiuan Peh, and N.K. Jha. Garnet: A detailed on-chip network model inside a full-system simulator. In *ISPASS*, 2009.

[3] A. Alhussien, Chifeng Wang, and N. Bagherzadeh. A scalable delay insensitive asynchronous noc with adaptive routing. In *Telecommunications (ICT), 2010 IEEE 17th International Conference on*, pages 995–1002, 2010.

[4] M. Altieri, W. Lombardi, D. Puschini, and S. Lesecq. Coupled voltage and frequency control for dvfs management. In *Power and Timing Modeling, Optimization and Simulation (PATMOS), 2013 23rd International Workshop on*, pages 207–214, Sept 2013.

[5] A. Amouri, H. Amrouch, T. Ebi, J. Henkel, and M. Tahoori. Accurate thermal-profile estimation and validation for fpga-mapped circuits. In *IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 57–60, 2013.

[6] H. Amrouch, T. Ebi, J. Schneider, S. Parameswaran, and J. Henkel. Analyzing the thermal hotspots in fpga-based embedded systems. In *23rd International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–4, 2013.

[7] A. Anta and P. Tabuada. To sample or not to sample: Self-triggered control for nonlinear systems. *IEEE Transactions on Automatic Control*, 55(9):2030–2042, 2010.

[8] A. Bar-Cohen. Thermal management of on-chip hot spots and 3d chip stacks. In *Microwaves, Communications, Antennas and Electronics Systems, 2009. COMCAS 2009. IEEE International Conference on*, pages 1–8, 2009.

[9] A. Bar-Cohen. Thermal management of on-chip hot spots and 3d chip stacks. In *Microwaves, Communications, Antennas and Electronics Systems, 2009. COMCAS 2009. IEEE International Conference on*, pages 1–8, Nov 2009.

[10] A. Bartolini, M. Cacciari, A. Tilli, and L. Benini. Thermal and energy management of high-performance multicores: Distributed and self-calibrating model-predictive controller. *IEEE Transactions on Parallel and Distributed Systems*, 24(1):170–183, 2013.

[11] Andrea Bartolini, Matteo Cacciari, Andrea Tilli, Luca Benini, and Matthias Gries. A virtual platform environment for exploring power, thermal and reliability management control strategies in high-performance multicores. In *GLSVLSI'10*, pages 311–316, New York, NY, USA, 2010. ACM.

[12] E. Beigne, F. Clermidy, S. Miermont, and P. Vivet. Dynamic voltage and frequency scaling architecture for units integration within a gals noc. In *Networks-on-Chip, 2008. NoCS 2008. Second ACM/IEEE International Symposium on*, pages 129–138, 2008.

[13] A Ben Ahmed, A Ben Abdallah, and K. Kuroda. Architecture and design of efficient 3d network-on-chip (3d noc) for custom multicore soc. In *Broadband, Wireless Computing, Communication and Applications (BWCCA), 2010 International Conference on*, pages 67–73, Nov 2010.

[14] *How to Benchmark Code Execution Times on Intel IA-32 and IA-64 Instruction Set Architectures*. `www.intel.com/content/dam/www/public/us/en/documents/white-papers/ia-32-ia-64-benchmark-code-execution-paper.pdf`.

[15] A. Bianco, P. Giaccone, and Nanfang Li. Exploiting dynamic voltage and frequency scaling in networks on chip. In *High Performance Switching and Routing (HPSR), 2012 IEEE*, pages 229–234, 2012.

[16] N. Binkert, B. Beckmann, G. Black, S.K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D.R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M.D. Hill, and D.A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, August 2011.

[17] Sergio Bittanti. Model identification and adaptive systems. In *Pitagora Editrice, Bologna,*, page 304 p. (in italian), 2004.

[18] T. Bjerregaard and S. Mahadevan. A survey of research and practices of network-on-chip. *ACM Computing Surveys*, 38(1):71–121, 2006.

[19] B. Black, M. Annavaram, N. Brekelbaum, J. DeVale, Lei Jiang, G.H. Loh, D. McCauley, P. Morrow, D.W. Nelson, D. Pantuso, P. Reed, J. Rupley, Sadasivan Shankar, J. Shen, and C. Webb. Die stacking (3d) microarchitecture. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 469–479, 2006.

[20] S. Borkar. Networks for multi-core chips: a contrarian view. In *Special Session at ISLPED*, 2007.

[21] D. Brooks and M. Martonosi. Dynamic thermal management for high-performance microprocessors. In *High-Performance Computer Architecture, 2001. HPCA. The Seventh International Symposium on*, pages 171–182, 2001.

[22] David Brooks, Vivek Tiwari, and Margaret Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th annual international symposium on Computer architecture*, ISCA '00, pages 83–94, New York, NY, USA, 2000. ACM.

[23] David Brooks, Vivek Tiwari, and Margaret Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th annual international symposium on Computer architecture*, ISCA, pages 83–94, New York, NY, USA, 2000. ACM.

[24] T.E. Carlson, W. Heirman, and L. Eeckhout. Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, pages 1–12, 2011.

[25] F.E. Cellier and E. Kofman. *Continuous system simulation*. Springer, 2006.

[26] S. Corbetta, D. Zoni, and W. Fornaciari. A temperature and reliability oriented simulation framework for multi-core architectures. In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI2012), University of Massachusetts, Amherst, USA*, aug. 2012.

[27] R. David, P. Bogdan, R. Marculescu, and U. Ogras. Dynamic power management of voltage-frequency island partitioned networks-on-chip using intel's single-chip cloud computer. In *Networks on Chip (NoCS), 2011 Fifth IEEE/ACM International Symposium on*, pages 257–258, 2011.

[28] R.H. Dennard, F.H. Gaensslen, V.L. Rideout, E. Bassous, and A.R. LeBlanc. Design of ion-implanted mosfet's with very small physical dimensions. *Solid-State Circuits, IEEE Journal of*, 9(5):256–268, Oct 1974.

[29] J Donald and M Martonosi. Techniques for Multicore Thermal Management: Classification and New Exploration. In *Computer Architecture, 33rd International Symposium on*, 2006.

[30] J. Donald and M. Martonosi. Techniques for multicore thermal management: Classification and new exploration. In *Computer Architecture. ISCA '06. 33rd International Symposium on*, pages 78–88, 2006.

[31] J. Donald and M. Martonosi. Techniques for multicore thermal management: Classification and new exploration. pages 78–88, 2006.

[32] M.L. Dreyer, K.-Y. Fu, and C.J. Varker. An electromigration model that includes the effects of microstructure and temperature on mass transport. *Journal of Applied Physics*, 73(10):4894–4902, 1993.

[33] T. Ebi, H. Amrouch, and J. Henkel. Cool: Control-based optimization of load-balancing for thermal behavior. pages 255–264, 2012.

[34] W.F. Egan. *Frequency Synthesis by Phase Lock*. John Wiley & Sons, 1999.

[35] R. Egawa, Y. Funaya, R. Nagaoka, A Musa, H. Takizawa, and H. Kobayashi. Design and early evaluation of a 3-d die stacked chip multi-vector processor. In *3D Systems Integration Conference (3DIC), 2010 IEEE International*, pages 1–8, Nov 2010.

[36] A. Elshazly, R. Inti, M. Talegaonkar, and P.K. Hanumolu. A 1.5GHz 1.35mW 112dBc/Hz in-band noise digital phase-locked loop with 50fs/mV supply-noise sensitivity. In *VLSI Circuits (VLSIC), 2012 Symposium on*, pages 188–189, June 2012.

[37] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. *Micro, IEEE*, pages 122–134, 2012.

[38] P. Fritzson. *Principles of object-oriented modeling and simulation with Modelica 2.1*. John Wiley & Sons, 2010.

[39] P. Fritzson, P. Aronsson, A. Pop, H. Lundvall, K. Nystrom, L. Saldamli, D. Broman, and A. Sandholm. Openmodelica - a free open-source environment for system modeling, simulation, and teaching. In *IEEE International Conference on Computer Aided Control System*, pages 1588–1595, 2006.

[40] Y. Fu, N. Kottenstette, C. Lu, and X.D. Koutsoukos. Feedback thermal control of real-time systems on multicore processors. pages 113–122, 2012.

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

[42] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. Mibench: A free, commercially representative embedded benchmark suite. In *Proceedings of the Workload Characterization, WWC-4. 2001 IEEE International Workshop*, pages 3–14, Washington, DC, USA, 2001. IEEE Computer Society.

[43] Paul Heremans, G. Van den Bosch, R. Bellens, G. Groeseneken, and H.E. Maes. Temperature dependence of the channel hot-carrier degradation of n-channel mosfet's. *Electron Devices, IEEE Transactions on*, 37(4):980–993, 1990.

[44] Ming-yu Hsieh, Arun Rodrigues, Rolf Riesen, Kevin Thompson, and William Song. A framework for architecture-level power, area, and thermal simulation and its application to network-on-chip design exploration. *SIGMETRICS Perform. Eval. Rev.*, 38:63–68, 2011.

[45] http://lwn.net/Articles/549580. (nearly) full tickless operation in linux 3.10.

[46] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M.R. Stan. Hotspot: a compact thermal modeling methodology for early-stage vlsi design. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 14(5):501–513, May 2006.

[47] W. Huang, K. Skadron, S. Gurumurthi, R.J. Ribando, and M.R. Stan. Differentiating the roles of ir measurement and simulation for power and temperature-aware design. In *IEEE International Symposium o Performance Analysis of Systems and Software*, pages 1–10, 2009.

[48] Wei Huang. *HotSpot–A Chip and Package Compact Thermal Modeling Methodology for VLSI Design*. PhD thesis, University of Virginia, 2007.

[49] A. Jaleel and B. Jacob. In-line interrupt handling and lock-up free translation lookaside buffers (tlbs). *IEEE Transactions on Computers*, 55(5):559–574, 2006.

[50] M. Kadin and S. Reda. Frequency and voltage planning for multi-core processors under thermal constraints. In *IEEE International Conference on Computer Design*, pages 463–470, 2008.

[51] A.B. Kahng, L. Bin, L.S. Peh, and K. Samadi. Orion 2.0: A fast and accurate noc power and area model for early-stage design space exploration. In *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, pages 423–428, April 2009.

[52] Wonyoung Kim, Meeta S. Gupta, Gu yeon Wei, and David M. Brooks. Enabling onchip switching regulators for multi-core processors using current staggering. In *In Proceedings of the Work. on Architectural Support for Gigascale Integration*, 2007.

[53] J. Kong, S.W. Chung, and K. Skadron. Recent thermal management techniques for microprocessors. *ACM Computing Surveys*, 44(3), 2012.

[54] A. Leva and A.V. Papadopoulos. Tuning of event-based industrial controllers with simple stability guarantees. *Journal of Process Control*, 23:1251–1260, 2013.

[55] A. Leva, F. Terraneo, and W. Fornaciari. Event-based thermal control for high-density processors. In *IEEE International Conference on Event-based Control, Communication and Signal Processing*, 2015. (Under review).

[56] S. Li, J.H. Ahn, R.D. Strong, J.D. Brockman, D.M. Tullsen, and N.P. Jouppi. The mcpat framework for multicore and manycore architectures: Simultaneously modeling power, area, and timing. *ACM Trans. Archit. Code Optim.*, 10(1):5:1–5:29, April 2013.

[57] M. Lis, Pengju Ren, Myong Hyon Cho, Keun Sup Shim, C.W. Fletcher, O. Khan, and S. Devadas. Scalable, accurate multicore simulation in the 1000-core era. In *Performance Analysis of Systems and Software (ISPASS), IEEE International Symposium on*, pages 175 –185, 2011.

[58] Lennart Ljung. *System Identification: Theory for the User*. Prentice Hall, 2 edition, January 1999.

[59] C. Ma, H.J. Mattausch, M. Miyake, T. Iizuka, M. Miura-Mattausch, K. Matsuzawa, S. Yamaguchi, T. Hoshida, M. Imade, R. Koh, T. Arakawa, and J. He. Compact reliability model for degradation of advanced p-mosfets due to nbti and hot-carrier effects in the circuit simulation. In *Reliability Physics Symposium (IRPS), 2013 IEEE International*, pages 2A.3.1–2A.3.6, 2013.

[60] S.E. Mattsson, H. Elmqvist, and Dynasim AB. Modelica - an international effort to design the next generation modeling language. In *Special issue on Computer Aided Control System Design, CACSD*, pages 16–19, 1997.

[61] I. Miro Panades and A. Greiner. Bi-synchronous fifo for synchronous circuit communication well suited for network-on-chip in gals architectures. In *Networks-on-Chip, 2007. NOCS 2007. First International Symposium on*, pages 83–94, 2007.

[62] I. Miro Panades, A. Greiner, and A. Sheibanyrad. A low cost network-on-chip with guaranteed service well suited to the gals approach. In *Nano-Networks and Workshops, 2006. NanoNet '06. 1st International Conference on*, pages 1–5, 2006.

[63] Asit K. Mishra, N. Vijaykrishnan, and Chita R. Das. A case for heterogeneous on-chip interconnects for cmps. In *Proceedings of the 38th annual international symposium on Computer architecture*, ISCA '11, pages 389–400, New York, NY, USA, 2011. ACM.

[64] Asit K. Mishra, Aditya Yanamandra, Reetuparna Das, Soumya Eachempati, Ravi Iyer, N. Vijaykrishnan, and Chita R. Das. Raft: A router architecture with frequency tuning for on-chip networks. *J. Parallel Distrib. Comput.*, 71(5):625–640, May 2011.

[65] Thomas Moscibroda and Onur Mutlu. A case for bufferless routing in on-chip networks. In *Proceedings of the 36th annual international symposium on Computer architecture*, ISCA '09, pages 196–207, New York, NY, USA, 2009. ACM.

[66] N. Muralimanohar, R. Balasubramonian, and N.P. Jouppi. Architecting efficient interconnects for large caches with cacti 6.0. *Micro, IEEE*, 28(1):69–79, Jan 2008.

[67] N. Nikitin and J. Cortadella. A performance analytical model for network-on-chip with constant service time routers. In *Computer-Aided Design - Digest of Technical Papers, 2009. ICCAD 2009. IEEE/ACM International Conference on*, pages 571–578, 2009.

[68] U.Y. Ogras, P. Bogdan, and R. Marculescu. An analytical approach for network-on-chip performance analysis. *CAD-ICS, IEEE Transactions on*, 29(12):2001–2013, 2010.

[69] U.Y. Ogras, R. Marculescu, P. Choudhary, and D. Marculescu. Voltage-frequency island partitioning for gals-based networks-on-chip. In *Design Automation Conference, 2007. DAC '07. 44th ACM/IEEE*, pages 110–115, 2007.

[70] J.D. Owens, W.J. Dally, R. Ho, D. N. Jayasimha, S.W. Keckler, and Li-Shiuan Peh. Research challenges for on-chip interconnection networks. *Micro, IEEE*, 27(5):96–108, 2007.

[71] Jun Pan and T. Yoshihara. A fast lock phase-locked loop using a continuous-time phase frequency detector. In *Electron Devices and Solid-State Circuits, 2007. EDSSC 2007. IEEE Conference on*, pages 393–396, Dec 2007.

[72] E. Pekkarinen, L. Lehtonen, E. Salminen, and T.D. Hamalainen. A set of traffic models for network-on-chip benchmarking. In *System on Chip (SoC), 2011 International Symposium on*, pages 78–81, 2011.

[73] M.D. Powell, M. Gomaa, and T.N. Vijaykumar. Heat-and-run: Leveraging smt and cmp to manage power density through the operating system. pages 260–270, 2004.

[74] Subodh Prabhu, Boris Grot, Paul V. Gratz, and Jiang Hu. 1 ocin tsim- dvfs aware simulator for nocs.

[75] A. Raghavan, L. Yixin, A. Chandawalla, M. Papaefthymiou, K.P. Pipe, T.F. Wenisch, and M.M.K. Martin. Computational sprinting. In *High Performance Computer Architecture (HPCA), 2012 IEEE 18th International Symposium on*, pages 1–12, 2012.

[76] Jose Renau, Basilio Fraguela, James Tuck, Wei Liu, Milos Prvulovic, Luis Ceze, Smruti Sarangi, Paul Sack, Karin Strauss, and Pablo Montesinos. SESC simulator, 2005. http://sesc.sourceforge.net.

[77] E. Rotem, A. Naveh, D. Rajwan, A. Ananthakrishnan, and E. Weissmann. Power management architecture of the 2nd generation intel core microarchitecture, formerly codenamed sandy bridge. In *Hot Chips 23 Symposium*, 2011.

[78] G. Semeraro, G. Magklis, R. Balasubramonian, D.H. Albonesi, S. Dwarkadas, and M.L. Scott. Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling. In *High-Performance Computer Architecture, 2002. Proceedings. Eighth International Symposium on*, pages 29–40, 2002.

[79] Jeff Sharp, Jim Bierschenk, and H.B. Lyon. Overview of solid-state thermoelectric refrigerators and possible applications to on-chip thermal management. *Proceedings of the IEEE*, 94(8):1602–1612, 2006.

[80] K. Skadron, T. Abdelzaher, and M.R. Stan. Control-theoretic techniques and thermal-rc modeling for accurate and localized dynamic thermal management. In *High-Performance Computer Architecture, 2002. Proceedings. Eighth International Symposium on*, pages 17–28, Feb 2002.

[81] *Intel Platform Innovation Framework for EFI System Management Mode Core Interface Specification*. `www.intel.com/content/dam/www/public/us/en/documents/reference-guides/efi-smm-cis-v09.pdf`.

[82] V. Soteriou, N. Eisley, Hangsheng Wang, Bin Li, and Li-Shiuan Peh. Polaris: A system-level roadmap for on-chip interconnection networks. In *ICCD 2006.*, pages 134 –141, 2006.

[83] A. Sridhar, A. Vincenzi, D. Atienza, and T. Brunschwiler. 3d-ice: A compact thermal model for early-stage design of liquid-cooled ics. *Computers, IEEE Transactions on*, 63(10):2576–2589, Oct 2014.

[84] M.B. Taylor. A landscape of the new dark silicon design regime. *Micro, IEEE*, pages 8–19, 2013.

[85] F. Terraneo, A. Leva, D. Zoni, and W. Fornaciari. MPSoC Thermal Management using Event-Based Control. In *Design Automation Conference (DAC)*, 2015. (Under review).

[86] F. Terraneo, D. Zoni, and W. Fornaciari. A cycle accurate simulation framework for asynchronous noc design. In *International Symposium on System on Chip (SoC)*, pages 1–8, 2013.

[87] L. Thiele, L. Schor, H. Yang, and I. Bacivarov. Thermal-aware system analysis and software synthesis for embedded multi-processors. In *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, pages 268–273, June 2011.

[88] T. Toifl, C. Menolfi, P. Buchmann, M. Kossel, T. Morf, R. Reutemann, M. Ruegg, M.L. Schmatz, and J. Weiss. A 0.94-ps-RMS-jitter 0.016-mm2 2.5-GHz multiphase generator PLL with 360 deg; digitally programmable phase shift for 10-Gb/s serial links. *Solid-State Circuits, IEEE Journal of*, 40(12):2700–2712, Dec 2005.

[89] B. Vaidyanathan, Yu Wang, and Yuan Xie. Cost-aware lifetime yield analysis of heterogeneous 3d on-chip cache. In *Memory Technology, Design, and Testing, 2009. MTDT '09. IEEE International Workshop on*, pages 65–70, Aug 2009.

[90] B. Vandeveldeand and E. Beyne. Improved thermal fatigue reliability for flip chip assemblies using redistribution techniques. *Advanced Packaging, IEEE Transactions on*, 23(2):239–246, 2000.

[91] A. Vassighi and M. Sachdev. *Thermal and Power Management of Integrated Circuits*. Springer, 2006.

[92] V. Vasyutynskyy and K. Kabitzsch. Event-based control: overview and generic model. In *Proc. 8th IEEE International Workshop on Factory Communication Systems*, pages 271–279, Nancy, France, 2010.

[93] E.N. Wang, L. Zhang, Linan Jiang, Jae-Mo Koo, J.G. Maveety, E.A. Sanchez, Kenneth E. Goodson, and T.W. Kenny. Micromachined jets for liquid impingement cooling of vlsi chips. *Microelectromechanical Systems, Journal of*, 13(5):833–842, 2004.

[94] R. Weerasekera, L.R. Zheng, D. Pamunuwa, and H. Tenhunen. Extending systems-on-chip to the third dimension: performance, cost and technological tradeoffs. In *Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on*, pages 212–219, Nov 2007.

[95] D. Wentzlaff, P. Griffin, H. Hoffmann, Liewei Bao, B. Edwards, C. Ramey, M. Mattina, Chyi-Chang Miao, J.F. Brown, and A. Agarwal. On-chip interconnection architecture of the tile processor. *Micro*, 2007.

[96] B. Wittenmark, K.J. Åström, and K.E. Årzén. Computer control: An overview. *IFAC Professional Brief*, 2002.

[97] Shan Yan and Bill Lin. Design of application-specific 3d networks-on-chip architectures. In *Computer Design, 2008. ICCD 2008. IEEE International Conference on*, pages 142–149, Oct 2008.

[98] F. Zanini, D. Atienza, L. Benini, and G. De Micheli. Thermal-aware system-level modeling and management for multi-processor systems-on-chip. pages 2481–2484, 2011.

[99] F. Zanini, D. Atienza, and G. De Micheli. A control theory approach for thermal balancing of mpsoc. In *Design Automation Conference, 2009. ASP-DAC 2009. Asia and South Pacific*, pages 37–42, 2009.

[100] F. Zanini, C.N. Jones, D. Atienza, and G. De Micheli. Multicore thermal management using approximate explicit model predictive control. pages 3321–3324, 2010.

[101] Francesco Zanini. *Design of Thermal Management Control Policies for Multiprocessor Systems on Chip*. PhD thesis, École Polytechnique Fédérale de Lausanne, 2011.

[102] Guisheng Zhai, Bo Hu, K. Yasuda, and A.N. Michel. Qualitative analysis of discrete-time switched systems. In *American Control Conference, 2002.*, volume 3, pages 1880–1885 vol.3, 2002.

[103] D. Zoni, S. Corbetta, and W. Fornaciari. Hands: Heterogeneous architectures and networks-on-chip design and simulation. In *IEEE ISLPED'12*, aug. 2012.

[104] D. Zoni, F. Terraneo, and W. Fornaciari. An analytical, dynamic, power-performance router model for run-time noc optimizations. In *IEEE 26th International SOC Conference (SOCC)*, pages 290–295, 2013.