Design of Thermal Management Control Policies for Multiprocessor Systems on Chip

Francesco Zanini

Submitted for the degree of Doctor of Philosophy in Science
EPFL, Lausanne

October 2011

Abstract

The contribution of this thesis is a thorough study of thermal aware policy design for MPSoCs. The study includes the modelling of their thermal behavior as well as the improvement and the definition of new thermal management and balancing policies.

The work is structured on three main specific disciplines. The areas of contributions are: modeling, algorithms and system design.

This thesis extends the field of modeling by proposing new techniques to represent the thermal behavior of MPSoCs using a mathematical formalization. Heat transfer and modelling of physical properties of MPSoCs have been extensively studied. Special emphasis is given to the way the system cools down (i.e. micro-cooling, natural heat dissipation etc.) and the heat propagates inside the MPSoC.

The second contribution of this work is related to policies, which manage MPSoC working frequencies and micro-cooling pumps to satisfy user requirements in the most effective possible way, while consuming the lowest possible amount of resources. Several families of thermal policies algorithms have been studied and analyzed in this work for both 2D and 3D MPSoCs including liquid cooling technologies.

The discipline of system design has also been extended during the development of this thesis. Thermal management policies have been implemented in real emulation platforms and contributions in this area are related to the design and implementation of proposed innovations in real MPSoC platforms.

Keywords: Thermal, Management, Model, Predictive control, Temperature, Algorithm, 3D-MPSoC, Optimization, Accuracy.

Riassunto

Il contributo di questa tesi e' uno studio dettagliato dei sistemi di gestione termica(thermal management) per sistemi a multiprocessore(MPSoC). Lo studio prende in considerazione ogni singolo aspetto della tematica trattata: dalle equazioni matematiche che descrivono il comportamento termico degli MPSoC, agli algoritmi di gestione termica, alle politiche di smistamento del workload. La tesi e' multidisciplinare e si sviluppa a partire da tre tematiche principali: modellizzazione, algoritmi e architettura del systema.

Questa tesi estende la tematica della modellizzazione proponenedo nuove tecniche utilizzando un formalismo matematico il comportamento termico di un MPSoC. I meccanismi di trasferimento termico interni all' MPSoC sono stati studiati in dettaglio. Una speciale attenzione e' stata data ai sistemi di raffreddamento(micro-cooling) con liquido refrigerante.

Il secondo contributo di questa tesi riguarda le politiche di gestione energetica e termica per quanto riguarda la scelta della frequenza di funzionamento e della velocita' della pompa dei sistemi di raffreddamento a liquido. Come obbiettivo, il sistema di gestione dell' MPSoC deve rispondere in modo pronto alle richieste dell' utente impiegando il minor numero possibile di risorse e con il minor dispendio energetico possibile. In questa tesi mettiamo a confronto alcune famiglie di algoritmi per sistemi di tipo MPSoC sia planari che utilizzando tecnologie 3D con raffreddamento a liquido.

Il terzo contributo riguarda l'architettura implementativa dei sistemi di gestione termica degli MPSoC proposti in questa tesi. Questi sistemi sono stati sviluppati su piattaforme di simulazione e le principali innovazioni sono legate alla definizione di tali piattaforme software.

Parole chiave: Modellizzazione, Controllo predittivo, Gestione termica, Temperatura, Algoritmi, 3D-MPSoC, Ottimizzazione, Accuratezza.

Acknowledgements

First of all I would like to thank GOD for having given me everything I have achieved.

I would like to thank my advisor, Prof. Giovanni De Micheli who gave me the opportunity to pursue research in the beautiful research. He's a person with great vision, who is always interested in exploring new research directions and ideas. He gave me complete freedom in performing research. He has given excellent personal support to all his students, being more of a mentor. During my Ph.D., he gave me the opportunity to travel and see many realities in the best research labs in my field.

I am also thankful to Professor David Atienza for his help in the development of my thesis. He guided me with patience and he also helped me in developing the skills I have now. I thank also him for always finding the time in his busy schedule to read my papers during the night or in the early mornings.

I really thank Professors Luca Capisani, Luca Benini and Colin Jones as well as Dr. Srinivasan Murali for helping me in understanding the theory behind my research work and at the same time giving me directions on how to solve mathematical problems encountered during my research.

I really thank Professor Stephen Boyd for giving me the opportunity to visit his lab at Stanford University and let me see the beauty of the research environment in Silicon Valley. I also thank him for spending a lot of his precious time with me. I really thank also Professor Subhasish Mitra and Yang Wang for the inspiring discussions I had with them during my Stanford experience.

I thank Mohamed Mostafa Sabry Aly for his greatful collaboration in the development of some of my papers. I also thank my colleague Ciprian Seiculescu for his help in correcting homeworks in the Design Technology course at EPFL. Both these people are very friendly, smart and extremely hard working. For me it has been a pleasure to work with them.

I really thank Dr. Sandro Carrara, Nicolas Genko, Alessandro Cevrero, Andrea Bartolini, Martino Ruggiero, Giacomo Paci, Prof. Babak Falsafi, Prof. Heinrich Meyr for the time spent with me and the interesting ideas inspiring discussions.

I also thank the professors and the people who collaborated to the En-

trepreneurial Thought Leaders Seminars (Stanford University) and to the program Venture Challenge (EPFL) for their ideas inspiring and motivational lectures.

I would like to thank Rodolphe Buret for his kindness in resolving the many technical issues that would crop up from time to time. Special thanks must go to Christina Govoni who were helpful beyond words in addressing administrative issues in all my years at EPFL.

I'd like also to thank all the professors from Stanford University, University of Parma, the National University of Ireland, ALaRI and EFPL that helped me in improving my skills.

I am also grateful to my PhD jury members for reading and evaluating my thesis in detail and for providing constructive feedback.

I really would like to thank my parents Fulgenzio and Franca, my brothers Roberto and Davide, my sister Linda for psychologically supporting me during my studies. I'd like also to thank my grandmother Adele for preparing me good food when I was coming back home from Lausanne. I'd like also to thank the support of my uncles Ersilia and Luigi and all my relatives in general. I'd like to thank my uncles Umberto, Sarro, Riccardo for showing me that high achievements in life can be obtained without loosing humility or faith in God. I also Would like to thank all the people who prematurely died before the end of my Ph.d: my grandfathers Pietro and Franco, my grandmother Lea and my uncles Ermete and Maria Teresa.

I'd like also to thank my parents, the University of Parma, EPFL, National University of Ireland, the Franchetti Institute and the ALaRI partners for the economical support that they gave me during my studies.

Special thanks to all my Stanford University and EPFL friends that I made during my studies. For the conversations I had with them and for all the time they always had for me.

Contents

List of Figures

1 Introduction

1.1 Motivation

Some of the greatest challenges of modern society are related to energy consumption, dissipation, and waste. Among these, present and future technologies based on nanoscale materials and devices hold great potential for improved energy conservation, conversion, and harvesting. A prominent example is that of integrated electronics, where power dissipation issues have recently become one of its greatest challenges. For example, Figure 1.1 shows that data center energy consumption have doubled in the last five years, with waste heat requiring drastic cooling solutions like air conditioning systems. These cooling solutions increase even more the energy consumption of data centers. If present growth trends are maintained, data center and overall electronics power use

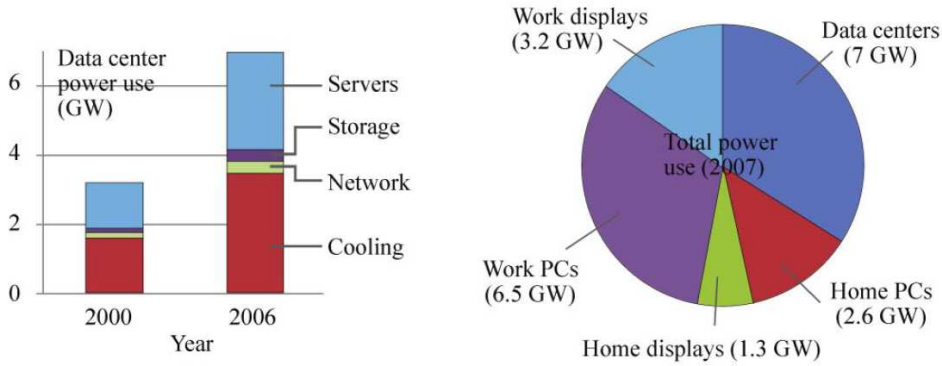

Figure 1.1: (left) Data center power use in the U. S. (right) Approximate breakdown of total power used by digital electronics in the U. S. [67]

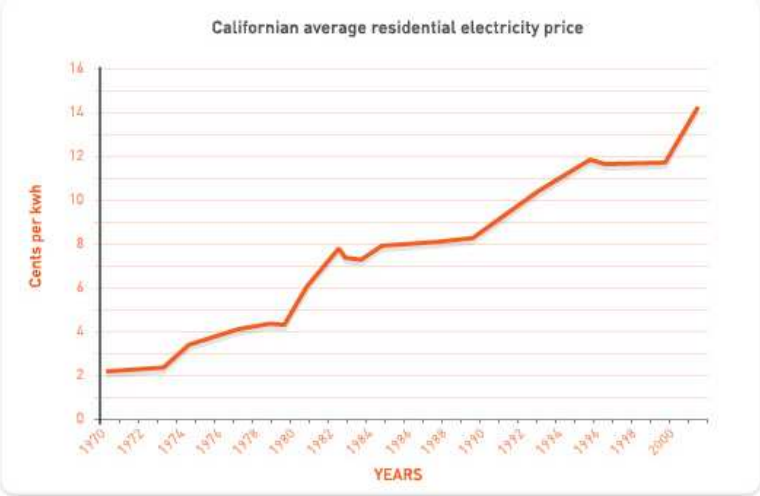

Figure 1.2: California average residence electricity price. [85]

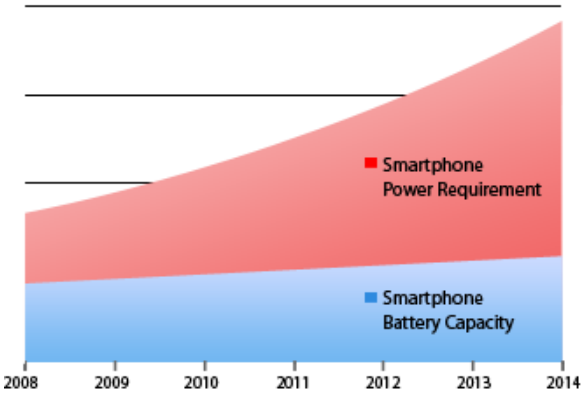

Figure 1.3: Smartphone Battery Capacity vs. Power Requirements [1]

could reach one third of total U. S. consumption by 2025 [67].

Moreover the cost of electricity keeps on growing. Figure 1.2 shows the cost for electricity in California where many data centers are located nowadays. Since 1980, the annual electric rates have increased an average of 2.91% and can reasonably be assumed to continue to rise at least that much over the next 25 years.

Energy consumption has become one of the primary concerns in electronic design due to the recent popularity of portable devices such as smartphones and laptops. Figure 1.3 shows battery capacity versus power requirements in the case of smartphones. The battery capacity has improved very slowly (a factor of 2 to 4 over the last 30 years), while the computational demands have drastically increased over the same time frame.

Figure 1.4 shows two power density trends in microprocessors power den-

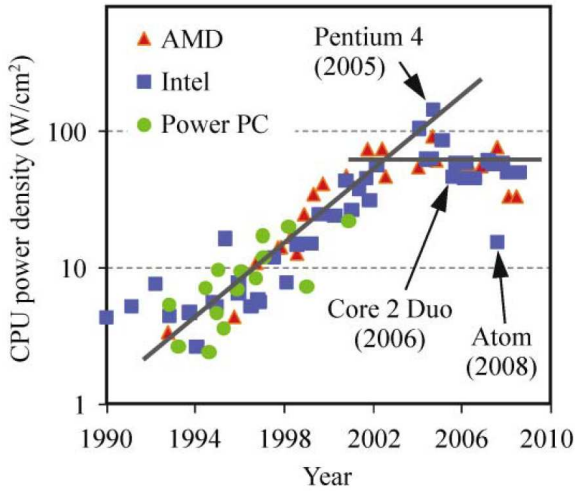

Figure 1.4: Power density trend of microprocessors [67]

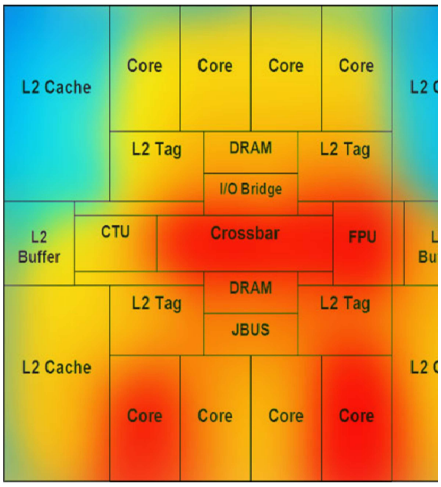

Figure 1.5: Thermal profile of Sun UltraSPARC T1(niagara) platform [43]

sity. The first trend is the one of high-performance microprocessors such as pentium 4, where performance is guaranteed by allowing heat spreaders, fans and liquid cooling technologies on the device. The second trend is the one of low-performance devices such as core 2 Duo and Atom where the power density has to be limited to a fixed maximum value. The reason is because either the limited heat extraction capabilities of the portable device or the fact that the device has to last long without recharging its battery.

A further problem is shown in Figure 1.5. As it can be noted, the thermal profile of a commercial MPSoC is quite nonuniform and hot spots (localized MPSoC areas with unsafe working temperature) may arise. This effect does not only affect the performance of the system, but also leads to unreliable circuit operation and affects the life-time of the chip [79]. Figure 1.6 shows

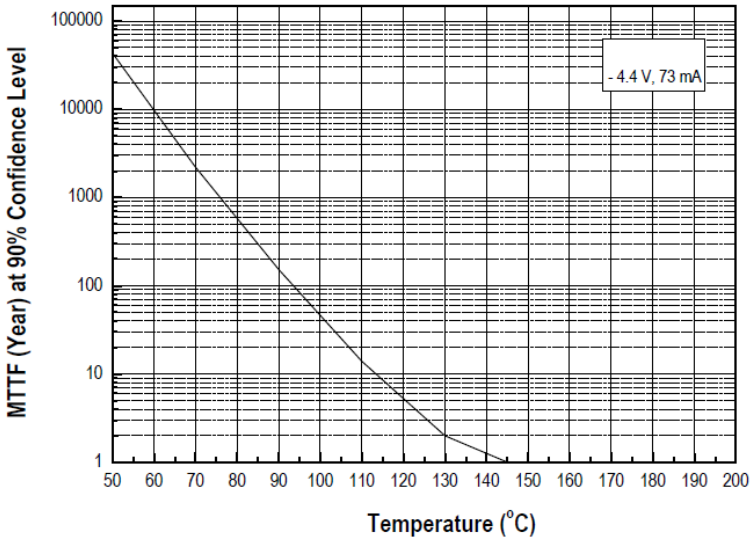

Figure 1.6: Mean Time To Failure Predicted at 90% Confidence Level (courtesy of: ASB inc.)

how temperature affects the lifetime(Mean Time to Failure) of a silicon device. As it can be noted there is an approximately exponential behavior between the junction temperature and the lifetime of the silicon device. Thus, thermal management for multicore architectures is a critical matter to tackle.

In the last years, thermal management and balancing techniques received a lot of attention. Many state-of-the-art thermal control policies operate power management by employing *dynamic frequency and voltage scaling* (DVFS) based techniques [60],[37]. This technique scales down the frequencies and the voltages of some specified units to save power and optimize performance. The problem with this technique is that the frequent abrupt change in working frequencies and voltages produces thermal cycling that raises the failure rate of the system [35], [90]. In addition, discontinuous power-mode transitions, both in voltage and frequencies scaling, waste additional power [42]. For the aforementioned reasons there are many trade-offs in power management techniques that are not easy to handle properly during the runtime execution of the MPSoC and with a low computational overhead.

Moreover, new challenges are related to emerging technologies for heat extraction. Heat extraction is based on a "heat sink". A heat sink is a component or assembly that transfers heat generated within a solid material to a fluid medium, such as air or a liquid. Examples of heat sinks are the heat exchangers used in refrigeration and air conditioning systems and the copper dissipator placed on top of microprocessors in desktop computers. A heat sink is physically designed to increase the surface area in contact with the cooling fluid surrounding it, such as the air. In past decades the research to increase the heat extraction performed by this element was focused on changing design

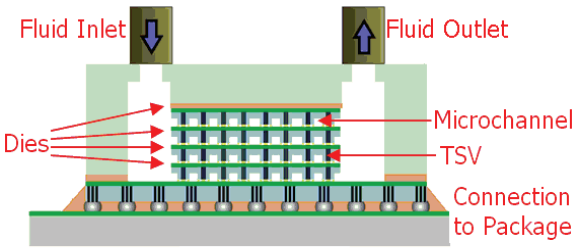

Figure 1.7: Cross-section of a test stack with liquid cooling.

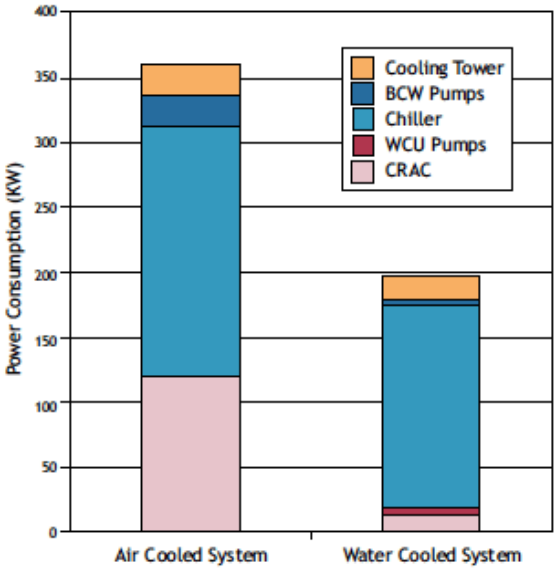

Figure 1.8: Comparison in the data center power consumption for an air cooled versus water cooled system [28]

factors such as: air velocity, choice of material, fin (or other protrusion) design and surface treatment. In recent years with the increase of power density the research has been focused on changing the coolant fluid. Not only air is used nowadays but also fluids circulating and exchanging heat with the heat spreader. The pipes where the coolant liquid is circulating can be either integrated inside MPSoCs [51]-[50] or placed on top of them [78]-[73] as shown in Figure 1.7. Experiments have shown that when a coolant fluid is pumped through the microchannels, up to $3.9KW/cm^3$ [15] of heat can be extracted. Figure 1.8 shows the power saving of using liquid cooling in data centers versus air cooling. Energy consumption within the data center is reduced because *computer room air conditioners* (CRACs) are replaced by more efficient modular *water cooling units* (WCUs). These new cooling technologies represent a key perspective for novel thermal management policies to both reduce cooling power consumption and at the same time increase performance.

1.2 Contributions

The contribution of this thesis is a thorough study of thermal aware policy design for MPSoCs. The study includes the modelling of MPSoCs thermal behavior as well as the improvement and the definition of new thermal management and balancing policies. MPSoC modelling, input parameters required by the optimization process as well as all the factors that improve its quality are analyzed in detail. Thus, the project has been develop on three main axes:

- Modeling: heat transfer and modelling of physical properties of MPSoCs. The goal is to model with a mathematical formalization the thermal behavior of MPSoCs. Special emphasis is given to the way the system cools down (i.e. micro-cooling, natural heat dissipation etc.) and the heat propagates inside the MPSoC.

- Algorithms: control theory based and convex optimization based optimization algorithm to manage the MPSoC to maximize performance, increase reliability and minimize power consumption. The policy manages MPSoC working frequencies and micro-cooling systems to reach its goal in the most effective possible way and consuming the lowest possible amount of resources.

- System design: design and implementation of proposed innovations in real MPSoC platforms. Policies are implemented in real emulation platforms. This field deals with constraints and problems related to the design of real implementation of proposed policies in 3D MPSOC systems.

1.2.1 Modeling

The objective of this part of the thesis is to improve the accuracy of current models and extend them from a 2-D to a 3-D perspective including liquid cooling methods. Contributions are the following:

A study of the integration method with reference to stability and accuracy. The integration method is used by the thermal management policy to predict the future thermal profile of the chip based on power dissipation values, current chip thermal profile and MPSoC floorplan. Stability and accuracy are two important points in discrete-time integration methods. New mathematical methods have been proposed.

Micro-cooling in 3D MPSoC. With the increasing of power density in new technologies, new cooling devices have been introduced. These device are integrated either inside MPSoCs [51], [93], [50] or they are placed on their heat spreading layers [78], [73]. An example of micro-cooling is shown in figure 1.9 [78]. New modelling techniques have been proposed to address the inclusion of new active and passive MEMS-based cooling systems for MPSoCs.

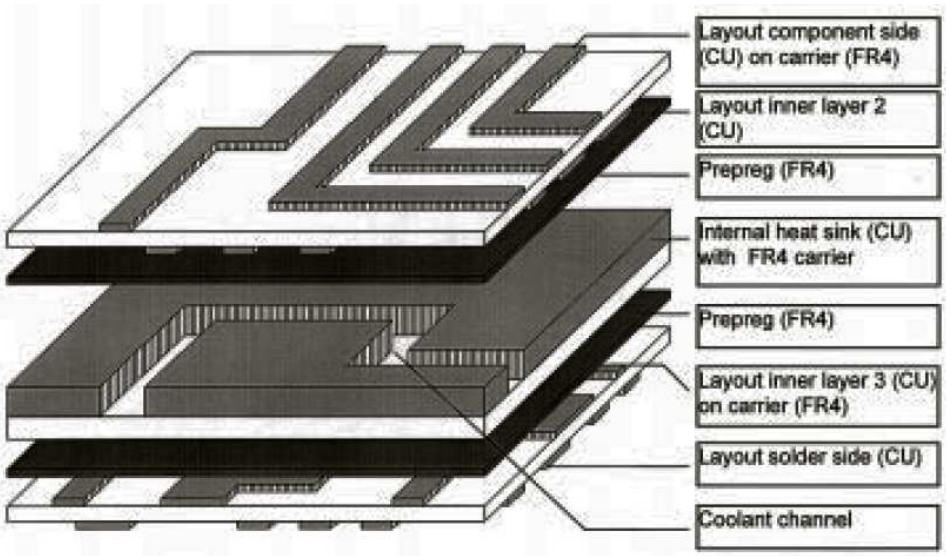

Figure 1.9: Example of a micro-cooling system from [78]

1.2.2 Algorithms

The contribution is a novel set of thermal management algorithms. These algorithms are based on a receding horizon approach that is called *model predictive control* (MPC) [2].

This mathematical technique tries to solve the thermal management problem inside an MPSoC in a similar way to playing chess. The control problem is indeed formulated over an interval of L time steps. The result of the optimization is an optimal sequence of future control moves, in the same way a player in chess is predicting future movements of the other player. Only the first sample of such a sequence is actually applied to the process, the remaining moves are discarded. At the next time step, a new optimal control problem based on new temperature measurements is solved over a shifted prediction horizon.

The goal of the thermal management algorithm is to satisfy performance user constraints while keeping the chip in a reliable condition. This condition implies that there are no localized areas in the chip where the silicon temperature is above a manufacturer specified vale (i.e. 370°). Another requirement is the fact that the chip does not have rapid thermal variations or large gradients in its thermal profile.

Algorithms have been analyzed according to their hardware implementation requirements and also to their optimality. Indeed there is a trade-off in designing the algorithm between its accuracy and its optimality. All aforementioned issues have been considered while designing and comparing proposed families of algorithms for both air and liquid cooling technologies.

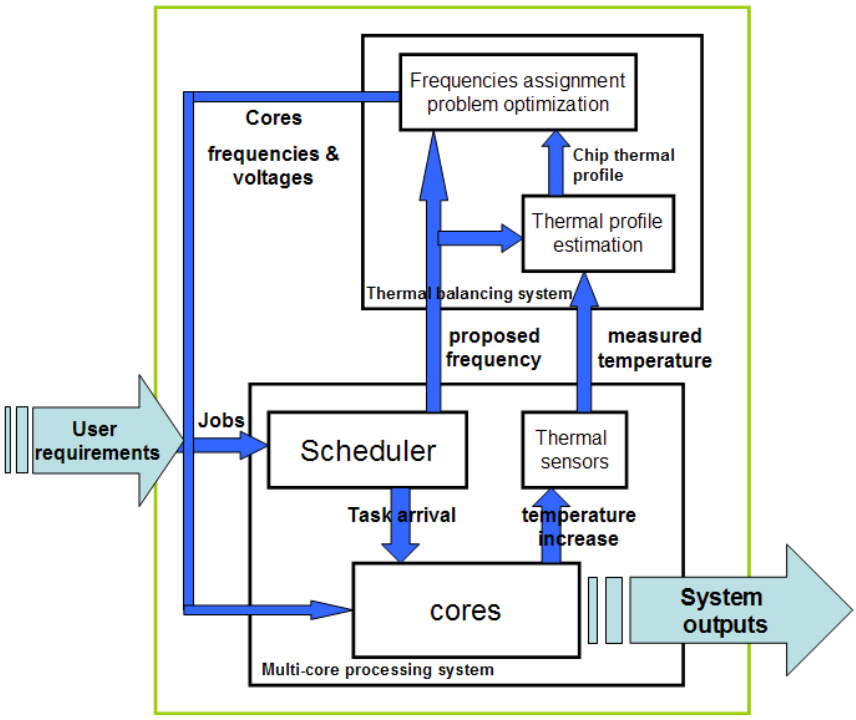

Figure 1.10: Block model representation of the system

1.2.3 System design

I describe contributions related to the design of implementations of proposed policies in 3D MPSOC systems. Figure 1.10 shows the block model representation of the system. The system can be divided into two main parts: the *multi-core processing system* (MPS) and the *thermal management system* (TMS).

The first one is the part of the MPSoC that takes user requirements and produces desired outputs. It can be seen as the plant to control. It consists of the cores, a scheduler and some thermal sensors that make some temperature measurements in some specific points of the chip.

The second part can be seen as the control system of the plant. It tries to optimize its performance by using information it receives from the MPS. The TMS consists of two subparts: one that estimates the thermal profile and one that solves the frequency assignment problem. The first one has to estimate and reconstruct the thermal profile starting from measurements coming from sensors. The second subpart has to find the optimum frequencies and voltages assignment for the cores in order to both meet performance requirements, reduce power consumption and improve chip reliability preventing hotspots. Constraints on the maximum temperature of the MPSoC are also respected in the optimization process. The optimization algorithm can be implemented with a look-up table(explicit) or an embedded solver(implicit).

The feasibility of the thermal management policy implementation has been addressed. To compare policies we consider trade-offs between the optimality and its implementation requirements. The policies have been tested with real scenarios common in many applications.

1.3 Assumptions and Limitations

As with any research work, there are some realistic assumptions that are used for building the proposed methods:

- Application and architecture scaling: The first assumption is that the number of cores on a chip is increasing and MPSoCs integrate several applications. This is a realistic assumption, as the chip complexity has been scaling roughly in accordance with the Moore's law. Today, there is indeed a convergence of several different applications onto the same platform, with massive computation and communication complexity.

- Synchronous design: It is assumed that the entire MPSoC supports different working frequencies. We also assume that functional units composing the MPSoC can be grouped into specific frequency islands each one working at a specific frequency(which can, if needed, be varied dynamically). Because of the aforementioned considerations, *globally asynchronous locally synchronous* (GALS) architectures are not supported.

- Power model: It is assumed that the power model of every functional unit inside the MPSoC is dependent on its working frequency. This is true in most applications, However for memories it is true only when cache misses are in the order of few percent (true in most architectures nowadays).

The main purpose of this work is to provide design methods. Thus, we tested our methods only on few specific case studies. However, without any loss of generality, every technique presented in this thesis can be applied to a large variety of MPSoCs. I envision that the methods presented in this work will have wide applicability, dealing with different architectures and operating conditions.

Due to the above assumptions, methods presented in this work do have some limitations:

Methods presented in this work can be applied only to architectures that can provide different frequency modes. Power data used in this work are abstracted at the component level and not at the transistor level. This means that we know the power consumption of the many blocks or functional units composing the MPSoC, however we do not know how this power is distributed among the different transistors. This is because of the lack of detailed power consumption information related to most commercial MPSoC architectures.

1.4 Thesis Organization

This thesis is divided into the following sections:

Chapter 2 gives a background on state-of-the-art techniques about the fields related to this work. Modeling, algorithms and system design are the main areas described by this survey.

Chapter 3 introduces mathematical models used in this thesis. Models are related to the heat transfer model of the MPSoC. A liquid cooling model of a 3d-MPSoC is also provided.

Chapter 4 introduces the concepts developed to model the workload of an MPSoC system. Moreover some considerations are made about the system energy models used in this thesis. Workload prediction is also introduced and two estimation techniques are presented.

Chapter 5 introduces air cooling algorithms. Four families of policies are analyzed and compared by both theoretical studies and experimental tests. We also provide a classification of the policies according to their problem formulation complexity and their computational effort requirements. The 2D-MPSoC case study used to run the simulations is also presented in detail.

Chapter 6 introduces liquid cooling algorithms. Two novel algorithms are proposed here. The first one is based on a centralized controller based on convex optimization. The second controller is a distributed structure based on the interaction between a global unit and many small controllers. The 3D-MPSoC case study used to run the simulations is also presented in detail.

Chapter 7 introduces techniques to perform a detailed thermal profile estimation of the MPSoC structure. Two techniques are presented here to achieve a temperature estimation by using few thermal sensors placed in specific locations on the MPSoC.

Chapter 8 presents the thermal simulation infrastructure used to test and compare policies presented in this thesis. The infrastructure consists of many parts written with three different programming languages and simulated on different simulation platforms.

Chapter 9 concludes the dissertation by summing up the contribution of this research and highlighting some possible extensions of this work in other areas.

2 Background

In this chapter we give a background material helpful for understanding the algorithms and methods proposed subsequently in this thesis. The background material is presented grouped into the three main areas of contribution of this work: Modeling, Algorithms and System design.

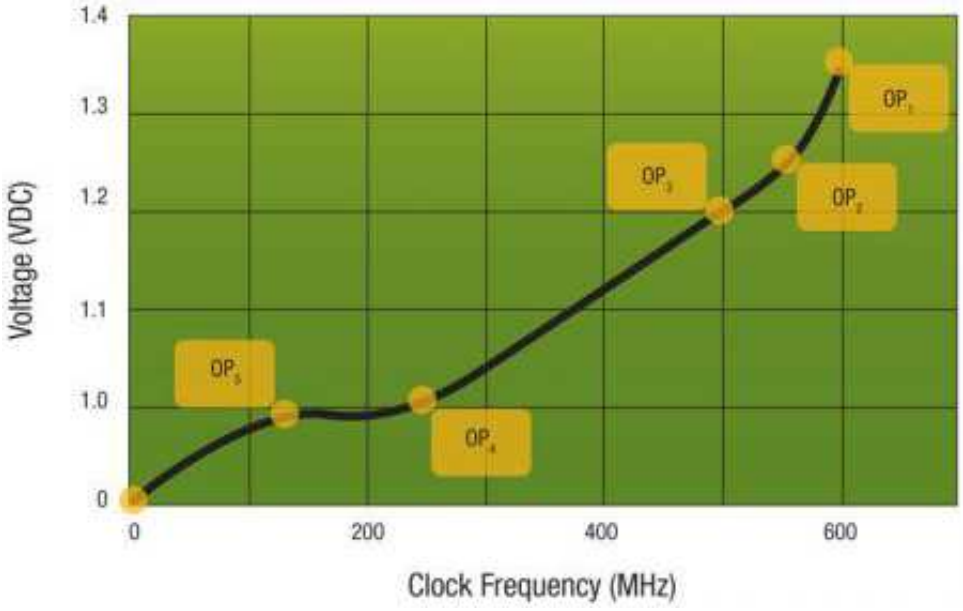

Figure 2.1: Operating Points in the voltage Vs frequency design space for an ARM Cortex A8 CPU

2.1 Modeling Background

The goal of this section is to give the basic background material to be able to understand innovations proposed in this thesis about new mathematical thermal models for MPOSoCs.

2.1.1 Dynamic voltage and frequency scaling

Dynamic voltage and frequency scaling (DVFS) is a technique to reduce energy consumption by changing processor speed and voltage at run-time depending on the needs of the applications running. This method is widely used as part of strategies to manage switching power consumption in battery powered devices such as cell phones and laptop computers. Low voltage modes are used in conjunction with lowered clock frequencies to minimize power consumption associated with components such as CPUs and DSPs; only when significant computational power is needed will the voltage and frequency be raised. Special supply regulation circuits are required to be able to deliver a multiple set of voltages to the SoC. Figure 2.1 shows possible voltages and frequencies operating points for the case of an ARM core.

It is important to vary both the voltage and the frequency because if only processor frequency is scaled, the total energy savings would be small or zero as power is inversely proportional to cycle time and energy is proportional to the execution time and power. The switching power dissipated by a chip using static CMOS gates is CV^2f, where C is the capacitance being switched per clock cycle, V is voltage, and f is the switching frequency, so this part of

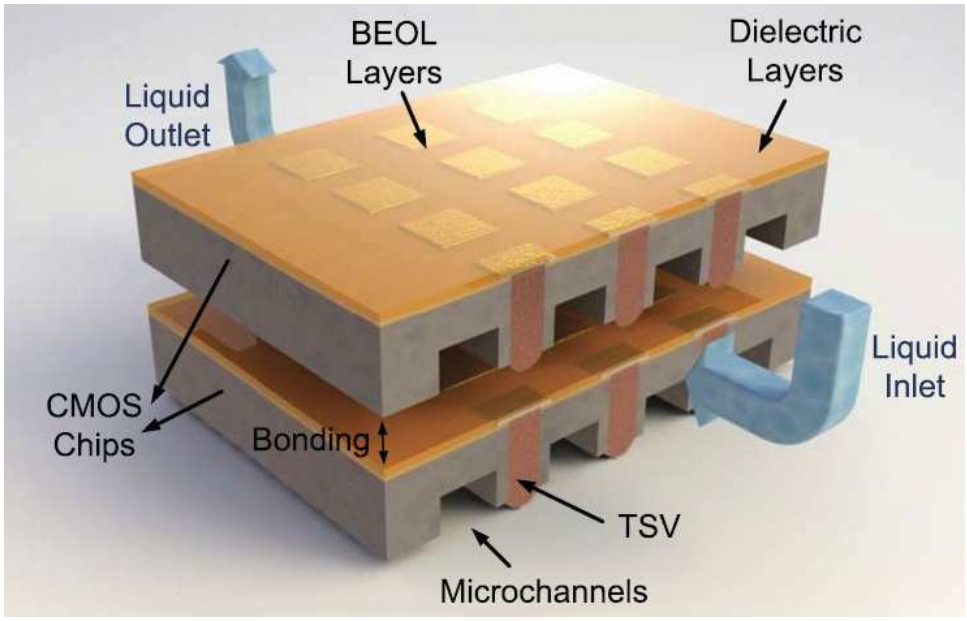

Figure 2.2: Simplified illustration of 3D stack with inter-tier liquid cooling

the power consumption decreases quadratically with voltage [70]. The formula is not exact however, as many modern chips are not implemented using only CMOS, but also uses pseudo nMOS gates, domino logic etc.

Moreover, there is also a static leakage current, which has become more and more accentuated as feature sizes have become smaller (below 90 nanometers) and threshold levels lower. When leakage current is a significant factor in terms of power consumption, chips are often designed so that portions of them can be powered completely off. This capability is an additional challenge to be managed in new thermal management systems.

2.1.2 3D-MPSoC and liquid cooling

A *three-dimensional integrated circuit* (3D IC, 3D-IC, or 3-D IC) is a chip in which two or more layers of active electronic components are integrated both vertically and horizontally into a single circuit. 3D integration [8] is a recently proposed design method for overcoming the limitations with respect to delay, bandwidth, and power consumption of the interconnects in large *multi-processor system-on-chip* (MPSoC) chips, while reducing the chip footprint and improving the fabrication yield. The main reason of all these benefits is the introduction of connections from one die to the other. These vertical wires are called *Through Silicon Vias* (TSV) and they allow us to make connections shorter as compared to a normal 2D chips [59]. A simplified illustration of a TSV in a 3D stack is shown in Figure 2.2.

Figure 2.2 is an example of 3D MPSoC where liquid cooling(liquid inlet/liquid outlet) is used as cooling mechanism. The reason for using liquid

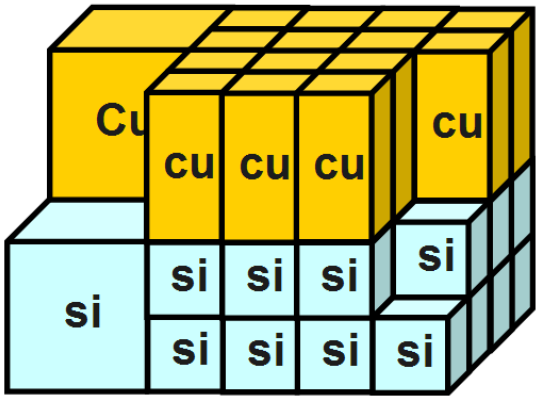

Figure 2.3: Cubic cells Model of the MPSoC

cooling is because of the higher thermal resistivity [39],[68], which irregularly spread in the 3D chip stack. Hence, it is more difficult to remove the heat from 3D systems with respect to conventional 2D MPSoCs.

Conventional back-side heat removal strategies, such as, air-cooled heat sinks and micro-channel cold-plates only scale with the die size and are insufficient to cool 3D MPSoC with hot spot heat fluxes up to 250W/cm2, as expected in forthcoming 3D MPSoC stacks [14]. On the contrary, inter-tier single and two-phase liquid cooling is a potential solution to address the high temperatures in 3D MPSoCs, due to the higher heat removal capability of liquids in comparison to air [16].

The use of convection in microchannels to cool down high power density chips has been an active area of research since the initial work by Tuckerman and Pease [88]. The heat removal capability of interlayer heat-transfer with pin-fin in-line structures for 3D chips is investigated in Brunschwiler et al. [14].

2.1.3 Thermal modelling

Skadron et al. [81] and Paci et al. [64] have developed a thermal-power model for super-scalar architectures. It not only predicts the temperature variations between the different components of a processor, but also accounts for the increased leakage power and reduced performance. Their results clearly prove the importance of hot spots in high performance systems. Based on this model, many architectural extensions have been proposed.

The model exploits the well known analogy between electrical circuits and thermal models. It decomposes the silicon die and heat spreader in elementary cells which have a cubic shape (Figure 2.3) and use an equivalent RC-model for computing the temperature of each cell. By varying the cell size can trade-off the simulation speed of the thermal with its accuracy. The coarser the cells become, the fewer cells we need to simulate, but the less accurate the temperature estimates become.

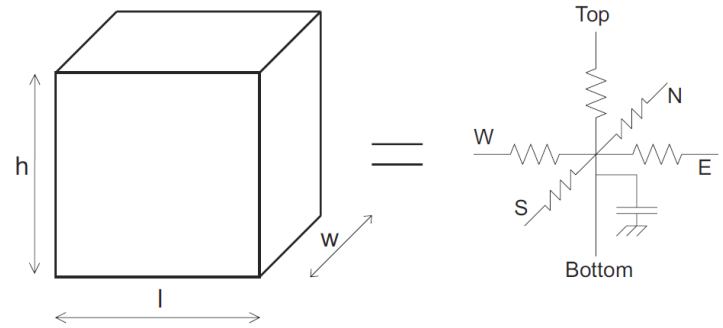

Figure 2.4: Equivalent RC circuit of a cell

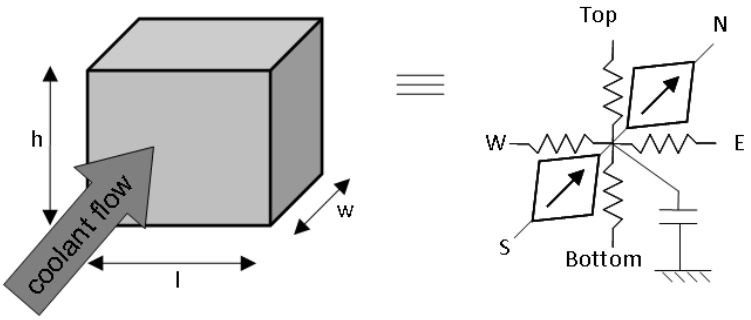

Figure 2.5: Equivalent circuit of a fluid thermal cell.

A thermal capacitance and five thermal resistances are associated with each cell (Figure 2.4). Four resistances are used for modeling the horizontal thermal spreading whereas the fifth is used for the vertical thermal behavior. See [64] for further details.

These concepts model with a good accuracy 2D and 3D chips using air cooling. In the case of liquid cooling, the model has been modified to the one presented by Sridhar et al. [83]. This model is exactly the same for any silicon or metal structure. The only difference is the introduction of the way the liquid cooling flow has been modelled. A graphical representation is shown in Figure 2.5.

Cells corresponding to microchannels containing the cooling fluid are modelled like any other cell with the addition of a current source and a current sink. To model the heterogeneous characteristics of the variable flow rate in microchannels, Sridhar et al. [83] introduced the ability to change the intensity of the current source and the resistance value of the cell at runtime. Thus, the interlayer material composing the MPSoC is divided into grid cells, where each grid cell except for the cells of the microchannels has a fixed thermal resistance value depending on the characteristics of the interface material. Thermal properties of the microchannel cells are computed based on the liquid flow rate through the cell at runtime as presented in Sridhar et al. [83].

2.1.4 Thermal sensing

Any thermal management algorithm to control the system must know the overall state (or thermal profile) of the MPSoC. This means that the temperature of every single cell in which the floorplan has been divided must be known.

A study of the thermal profile estimation problem has been presented in Memik et al. [56] and Sharifi and Rosing [80]. The proposed solutions are based on techniques trying to reduce temperature differences between thermal sensors and hot-spots by using the minimum possible number of sensors for a certain accuracy. The problem with these approaches is that since hot-spots are application dependent, there is no guarantee that all hot-spots are detected during the lifetime of the device.

A better approach to estimate these temperatures is to use a state estimator [33]. A state or thermal profile estimator is an algorithm able to derive the current thermal profile based on measurements in some specific locations on the chip with a specific rate. The parameter that measures how much a system is observable is called Observability. Observability refers to the property of a system that enables the reconstruction of the state variables given the inputs [33]. It means that we are able to reconstruct completely the thermal profile of the chip given the inputs only by looking at the measurements coming from the sensors.

The placement problem in an MPSoC is the problem of selecting right locations of thermal sensors to both minimize the number of sensors and maximize the observability of the system. Sumana and Venkateswarlu [86] select the location of the sensing element according to a sensor strategy using just described observability concept. Joshi and Boyd [41] solved the problem of making a system observable by employing of graph theory. The problem of choosing a set of measurements from a much larger set that also minimizes the estimation error is solved by Boukhobza and Hamelin [10] using a convex optimization based approach. This last method approximately solves the problem and has no guarantee that the performance gap is always small.

2.2 Algorithms Background

The goal of this section is to give the basic background material to be able to understand concepts behind the family of algorithms proposed in this thesis.

2.2.1 Reactive policies

This family of methods does not exploit history information. The policy takes reactive decisions based on information related to the current thermal profile and frequency setting of the MPSoC to control. For this reason they are defined as reactive policies.

The problem with this family of policies is that they react only when the thermal situation starts to be critical. For this reason their reaction is usu-

ally fast and abrupt frequency and corresponding temperature variations are generated. These variations undermine the reliability of the system.

Clock gating and DVFS-based policies

Dynamic power management, (see Benini and Micheli [5] for more details), was developed first for single processors. Lu et al. [52] devised a software architecture that enables system designers to investigate power management algorithms in a systematic fashion.

Merchant et al. [58] devised a variable speed processor which is thermally controlled based on an estimation on the hottest junction temperature of the chip. The problem with this method is that it uses a simplistic thermal model to predict the hottest junction temperature of the chip. This assumption degrades the quality of results.

Benini et al. [4], Paleologo et al. [65] and Qiu et al. [69] proposed policies based on stochastic optimum control. These methods take power management decisions based on theories of Markov decision processes and stochastic networks. Markov processes have by definition no memory of the past history of the system.

Early methods based on monitoring the idle time in processors have been presented by Halfhill [36], where DVFS has been used to turn off functional units when they were not being used. Later, more refined power management policies working at the OS-level were proposed. Rosing and Boyd [76] designed a methodology for managing power consumption in networks on chip. Xie and Hung [94] presented a set of scheduling mechanisms for MPSoCs that perform temperature management at the system-level.

Donald and Martonosi [26] reduced significantly localized hotspots by using thread migration techniques. Coskun et al. [24] designed a dynamic scheduling algorithm with negligible performance overhead. Bircher and John [7] used processor counters to get an online monitoring of the overall power consumption. Reda et al. [74] achieved temperature and hotspot tracking by a smart sensor allocation technique.

2.2.2 History based policies

To improve the efficiency of thermal management policies, memory has been introduced inside thermal management algorithms. This family of methods indeed, take decisions based on past information related to thermal profile, power consumption, frequency and liquid cooling setting and thermal model of the MPSoC. Past history information is used to predict the consequence that decisions taken by the policy have on the MPSoC.

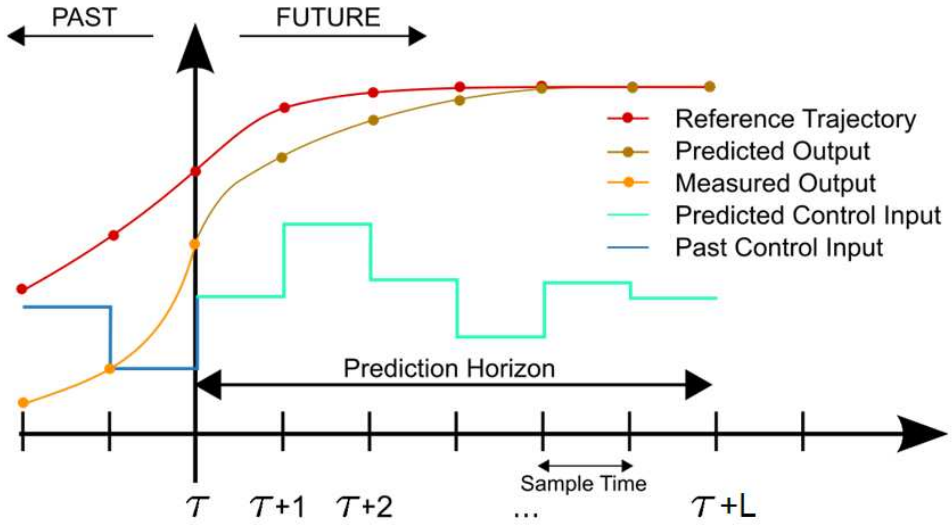

Figure 2.6: Predictive policies using history information, graphical representation [courtesy of Martin Behrendt]

Model predictive control and convex optimization

Model Predictive Control (MPC) is a technique that uses information on the MPSoC model and on the history to improve the desired performance target. A graphical representation is shown in Figure 2.6. The red line on Figure 2.6 shows the reference trajectory that in our case represents the requirements that the MPSoC has to satisfy. The yellow line is the measured output that the system can provide (i.e. MPSoC executed workload). At time τ the current state of the system is sampled and a specific control strategy is computed (light-blue line) (via a numerical minimization algorithm) for a relatively short time horizon in the future: $[\tau,\tau + L]$. The prediction horizon is called L. Specifically, the policy explores state trajectories (brown line) that emanate from the current state at time τ and find a control strategy until time $[\tau,\tau+L]$.

Only the first step of the control strategy is implemented, then the plant state is sampled again and the calculations are repeated starting from the new current state (now at time $\tau + 1$), yielding a new control and new predicted state path. The prediction horizon keeps being shifted forward and for this reason this approach is also called receding horizon control. The control problem is indeed formulated over an interval of L time steps. The result of the optimization is an optimal sequence of future control moves, in the same way a player in chess is predicting future movements of the other player.

In this thesis, we propose many different ways to solve the optimization problem. The first one is implicit and requires to solve on-line the minimization problem every time the policy is applied. With the second approach the optimization problem is solved off-line in a way that makes explicit the dependence of the solution on input parameters. Bemporad et al. [2] have shown

that the optimal explicit controller is *piecewise affine*. In other words, the state space can be divided by in a set of regions, bounded by linear inequalities (i.e., a polytope), and in each region a different linear controller can be specified and computed off-line. The main problem with this approach is that the number of coefficients to store is usually large for a complex MPSoC system. As a result, this method can be implemented only in MPSoC described by simplified thermal models using a small number of states and input/output variables. Moreover the policy has to be formulated over a limited interval of few time steps in the future to keep the implementation of the policy feasible. In this thesis, to solve this problem, we present a new explicit approximation method, that reduces computational complexity and enables its online implementation. As a result, this approach can be extended to complex MPSoC models without the need of a numerical embedded solver.

Another way to solve the receding horizon optimization problem is to express the optimization problem in terms of a convex function to be minimized. I propose a problem formulation where all equations are convex models that can be solved with polynomial-time complexity (in the number of variables and constraints) using interior point methods as shown in Boyd and Vandenberghe [12]. To solve the optimization problem, an efficient solver is CVX [34]. In this thesis, we refer to this approach as "convex optimization". The most relevant advantage of this approach is that since the computation of the solution takes a small computational effort, the problem formulation can be more complex and include advanced features without requiring a large amount of hardware resources.

History exploitation policies

In two recent works, history information has been exploited to improve thermal management policies. Coskun et al. [22] proposed a policy that exploits a temperature forecast technique based on an auto-regressive moving average model. In the work by Lee et al. [48], the policy uses auto-regressive moving average applied to performance counters to find the correlation between the applied voltage setting and the measured MPSoC temperature. Coskun et al. [23], propose a novel technique that adapts the thermal management policy to the current workload characteristics. The adaptation is done online exploiting information related to the workload history. Both these approaches are based on statistical methods to avoid hotspots. For this reason, there is not a formal guarantee to complete avoid this problem.

Murali et al. [62] proposed a convex optimization based policy. The policy is computed off-line and solutions are stored in a look-up table that is read at run-time. Input parameters needed for the optimization are the thermal profile, chip physical parameters and scheduler requirements. However, apart from chip parameters, the other two input data can assume many values. Assumptions needed to make the system feasible from an implementation perspective degrade the quality of results.

Two recent approaches presented by Cochran and Reda [19] and Zhang and Srivastava [101] describe two methodologies to achieve thermal prediction without completely relying on the thermal model, on thermal sensors and on power consumption statistical properties. In the work by Wang et al. [91] a chip-level power control algorithm based on optimal control theory is presented. This algorithm can control the power consumption of the MPSoC and can maintain the temperature of each core below a specified threshold. Magno et al. [54] tailors a similar concept for multi-modal video sensor nodes.

Aforementioned policies do not completely avoid hot spots, but they simply reduce their frequency. The reason is that the interaction between the prediction method, the thermal behavior of the MPSoC and the frequency assignment of the MPSoC has not been addressed as a joint optimization problem. The action taken by the policy to avoid hot spots does not address the problem from a global optimum perspective.

Liquid cooling based policies

Several works by Bhunia et al. [6] and Lee et al. [47] have explored the feasibility of liquid cooling as cooling method for 3D MPSoCs. Then, prior liquid cooling work by Coskun et al. [21] evaluates existing thermal management policies on a 3D system with a fixed-flow rate setting, and also investigates the benefits of variable flow using a policy to increment or decrement the flow rate based on temperature measurements, but without considering pump energy consumption.

Thermal management methods for 3D MPSoCs using a variable-flow liquid cooling have been recently proposed by Coskun et al. [20] and Sabry et al. [77]. In Coskun et al. [20], the proposed controller forecasts maximum system temperature, and uses this to proactively set the flow rate. The prediction method is based on autoregressive moving average to predict the maximum temperature for the next interval. In the work by Sabry et al. [77], the policy uses a fuzzy logic controller. Decisions here are taken based on experimentally proven rules of thumb to control the temperature profile of the 3D MPSoC while ensuring performance requirements to be satisfied. The problem with both previously mentioned approaches is that there is no formal guarantee to completely avoid hotspots. The first approach is indeed based on a not very accurate thermal model, while the policy of second approach is based on rule of thumbs.

2.3 System Design Background

The goal of this section is to give the basic background material to be able to understand how the emulation platform used to compare proposed thermal management systems works.

2.3.1 Validation of policies

To be able to test and compare policies proposed in this thesis, a real hardware simulation platform is needed. This platform has to support multiprocessors systems as well all real time data transfer mechanisms that happens in state-of-the-art MPSoCs. In the past, a number of architectural level multiprocessor simulators have been developed by the computer architecture community for performance analysis of large-scale parallel machines.

The tools proposed by Magnusson et al. [55], Rosenblum et al. [75] and Hughes et al. [38] operate at a very high level of abstraction: their processor models are highly simplified in an effort to speedup simulation and enable the analysis of complex software workloads. Furthermore, they all postulate a symmetric multiprocessing model (i.e. all the processing units are identical), which is universally accepted in large-scale, general-purpose multiprocessors. This model is not proper for embedded systems, where very different processing units (i.e general purpose, DSP, VLIW) can coexist.

Moreover hardware modeling accuracy is highly desirable because it would make it possible to use the same exploration engine both during architectural exploration and hardware design. These needs are well recognized in the *Electronic Design Automation* (EDA) community and several simulators have been developed to support SoC design. However, these tools (proposed by Mentor Graphics [57], CoWare [25], Van et al. [89], Mukherjee et al. [61] and Falsafi and Wood [31]) are primarily targeted towards single-processor architectures (e.g. a single processor cores with many hardware accelerators), and their extension toward MPSoCs, albeit certainly possible, is a non-trivial task.

2.3.2 Real-time hardware simulation framework

To develop the simulation infrastructure used in this work, we use MPARM proposed by Benini et al. [3]. In analogy with current SoC simulators, this design space exploration engine supports hardware abstraction level and continuity between architectural and hardware design, and at the same time it fully supports multiprocessing. In contrast with traditional mixed language co-simulators such as the one proposed by Mentor Graphics [57], all components of the system are modeled in the same language. A graphical block diagram representation of the system architecture is shown in Figure 2.7.

The multiprocessor simulation framework uses SystemC as simulation engine. The simulated system currently contains a model of the communication architecture (compliant with the AMBA bus standard), along with multiple masters (CPUs) and slaves (memories) (Figure 2.7). The intrinsic multi-master communication supported by the AMBA protocol is exploited by declaring multiple instances of the ISS master module, thus constructing a scalable multiprocessor simulator.

The processing modules of the system are represented by cycle accurate models of cached ARM cores. The module (Figure 2.8) is internally com-

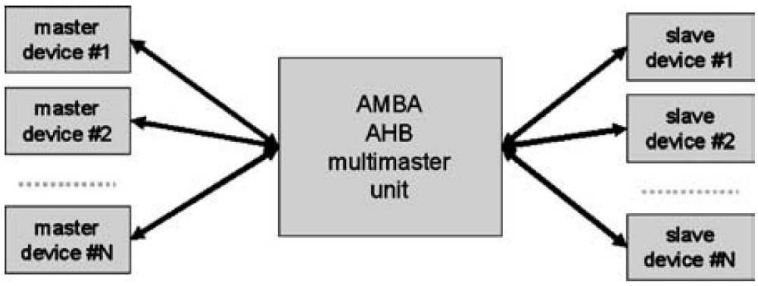

Figure 2.7: System architecture [3]

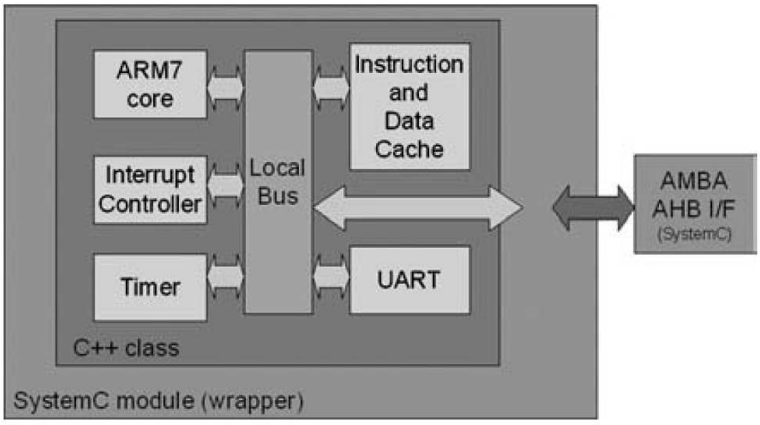

Figure 2.8: Processing module architecture [3]

posed of the ARM CPU, the first-level cache and peripherals (UART, timer, interrupt controller) simulator written in C++. It was derived from the open source cycle accurate SWARM (software ARM) simulator (proposed by M. Dales [53]) encapsulated in a SystemC wrapper. The insertion of an external (C++) *Instruction Set Simulator* (ISS) is subject to the necessity of interfacing it with the SystemC environment. For example, accesses to memories and interrupt requests must be trapped and translated to SystemC signals. Another important issue is synchronization. The ISS, typically written to be run as a single unit, is capable of being synchronized with the multiprocessing environment (i.e. there must be a way to start and stop it maintaining cycle-accuracy). The ISS is also capable of being multi-instantiable, since there will be one instance of the module for each simulated processor.

The simulation architecture is provided with two hierarchies of memories, namely cache memory and main memory. The cache memory is contained in the processing module and is directly connected to the CPU core through its local bus. Each processing module has its own cache, acting as a local instruction and data memory; it can be configured as a unified instruction and data cache or as two separate banks of instruction and data caches. Configuration parameters include also cache size, line length and the definition of

non cacheable areas in the address space. Main memory banks reside on the shared bus as slave devices. They consist of multiple instantiations of a basic SystemC memory module. Each memory module is mapped on its reserved area within the address space; it communicates with the masters through the bus using a request-ready asynchronous protocol; the access latencyexpressed in clock cyclesis configurable.

2.3.3 Simulation platform

To simulate the thermal management systems in a real simulation scenario we need a complete platform for the simulation of a MPSoC, enabling investigation in the parameter space to come up with the most efficient solution for a particular application domain. The platform needs also distinctive requirement of simulation speed, accuracy and capability to support design space exploration. The platform we decided to use is MPARM. This platform makes use of SystemC as simulation engine, so that hardware and software can be described in the same language, and is based on an AMBA bus compliant communication architecture. We tailored this platform according to the needs of the thermal management systems under comparison. We realized that the flexibility and the accuracy of this tool has been fundamental in the comparison and the validation of the thermal management systems proposed in this thesis.

3 Thermal Models

In this chapter we describe the mathematical models used to represent the thermal dynamics of a generic 2D/3D-MPSoC with air/liquid cooling from a control theory perspective. The thermal dynamics are here analyzed in detail.

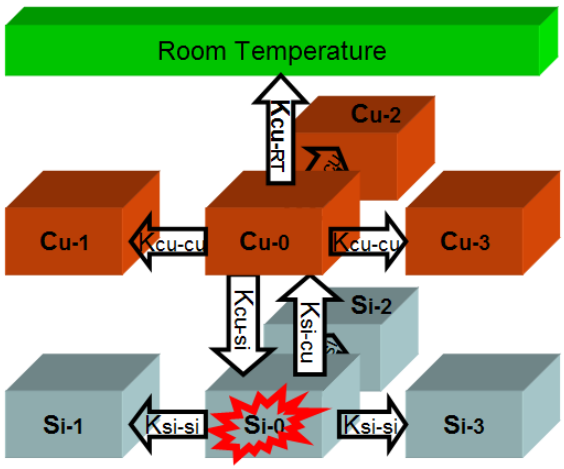

Figure 3.1: Modeling of the heat transfer inside a 2D-MPSoC

3.1 Thermal Modeling

In this section we describe mathematically how the thermal properties of the MPSoC can be represented by differential equations using a standard state-space representation.

3.1.1 State-space heat propagation model

As we presented in the background chapter, the chip floorplan is divided into grid cells with cubic shapes. Each functional unit in the floorplan can be represented by one or more thermal cells in the silicon layer. The thermal behavior of the MPSoC is computed through the interaction of multiple discrete-time differential equations modeling the thermal interactions between neighboring cells. A graphical representation of a 2D-MPSoC structure is presented in Figure 3.1. In Figure 3.1, the gray and brown blocks represent the silicon and copper layer, respectively. The ambient temperature is modeled as a layer with uniform temperature and infinite thermal capacity. The red mark inside the silicon cell represents the cell's power dissipation. At any moment in time, the temperature change of each block due to its neighbors is given by the temperature difference between the two blocks, multiplied by the constant that labels each pair of arrows.

The thermal model is formed by considering the heat conductances $\mathbf{G}$ and capacitances $\mathbf{C}$ of the cells (Paci et al. [64], Skadron et al. [81]). The differential equation modeling the heat flow is given by:

$$\mathbf{C} \cdot \left. \frac{\partial \mathbf{t}(\theta)}{\partial \theta} \right|_{\theta=\tau} = -\mathbf{G}(\tau) \cdot \mathbf{t}_\tau + \mathbf{p}_\tau \tag{3.1}$$

where $\mathbf{t}_\tau$ is the temperature vector at time $\theta = \tau$. In this model we assume that lateral heat capacitances are negligible. For this reason matrix $\mathbf{C}$ is di-

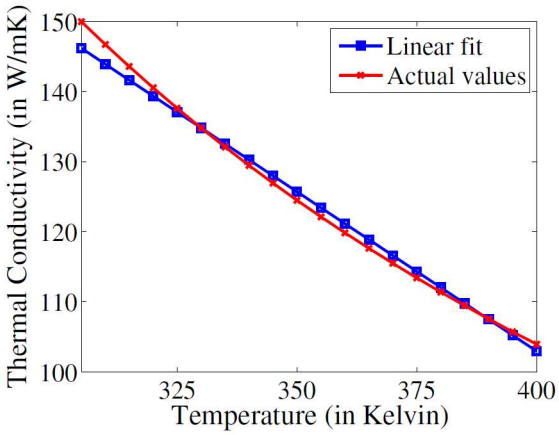

Figure 3.2: Silicon thermal conductivity and linear fit

agonal with the entries representing the thermal capacitance of the cells (in Joules/Kelvin). Matrix $\mathbf{G}$ is the thermal conductance matrix (the conductance values have units of Watts/Kelvin). The silicon thermal conductivity is shown in Figure 3.2. As analyzed by Ramalingam et al. [71], it varies with temperature in an approximately linear fashion. For this reason, matrix $\mathbf{G}$ is a function of the thermal profile of the MPSoC at time τ.

In this thesis we want to represent the thermal model using a standard state-space representation [33]. This representation uses a linear, time invariant discrete-time system model. Since the original thermal model is nonlinear, and coefficients are temperature-dependent (Paci et al. [64]), we need to linearize the solution of the differential equation 3.1 modeling the heat flow inside the MPSoC system. The rationale behind it is described in Zanini et al. [96], Skadron et al. [81] and Paci et al. [64]. In following subsections we will describe mathematically how the thermal model of the MPSoC can be represented by following equation:

$$\mathbf{t_{\tau+1}} = \mathbf{A}\mathbf{t}_\tau + \mathbf{B}\mathbf{p}_\tau + \mathbf{w} \tag{3.2}$$

This equation expresses the temperature profile vector at time step $\tau + 1$, $\mathbf{t_{\tau+1}} \in \Re^{\mathbf{n}}$, as a function of the current thermal profile at step τ and the input vector $\mathbf{p}_\tau \in \Re^{\mathbf{p+z}}$. This model is discrete-time with time step equal to $\Delta\tau$. The total number of cells in all layers of the 3D MPSoC structure is n, the total number of independent processing units is p and the total number of independent cooling liquid flow rates is z. Matrices $\mathbf{A} \in \Re^{\mathbf{n\times n}}$ and $\mathbf{B} \in \Re^{\mathbf{n\times(p+z)}}$ describe the heat propagation properties of the MPSoC while $\mathbf{w} \in \Re^{\mathbf{n}}$ takes into account of the fact that the MPSoC is at room temperature. If we refer to all temperatures as offset from the room temperature we can omit this vector. At time τ, the temperature of the next simulation step of cell i, i.e. $(\mathbf{t_{\tau+1}})_\mathbf{i}$ can be computed thanks to Equation 3.2. The first p entries of vector $\mathbf{p}$ are the power consumptions for each of the p independent processing units, while the remaining z entries are the normalized cooling flow rates for each of the z

independent microchannels.

3.1.2 First order ODE solvers

A first-order ODE solver is the simplest tool for solving the system of differential equations modeling the MPSoC. Because of its simplicity, it has been used by many state-of-the-art thermal simulators like the earliest versions of HotSpot [81] or real-time thermal emulation frameworks targeting embedded SoCs [64]. This integration method has been used in early thermal policies embedding a thermal profile model of the MPSoC (see [62] for more details). The *Forward Euler*(FE) method is described by Equation 3.3:

$$\mathbf{t}_{\tau+\boldsymbol{\Delta}\tau} = \mathbf{t}_\tau + \boldsymbol{\Delta}\tau \cdot \left.\frac{\partial \mathbf{t}(\theta)}{\partial \theta}\right|_{\theta=\tau} \tag{3.3}$$

where $\mathbf{t}_\tau$ and $\mathbf{t}_{\tau+\boldsymbol{\Delta}\tau}$ are the vectors containing the temperature of any thermal cell composing the MPSoC respectively at time τ and $\tau + \Delta\tau$. θ is the time, $\Delta\tau$ is the simulation step size and $\left.\frac{\partial \mathbf{t}(\theta)}{\theta}\right|_{\theta=\tau}$ is the vector containing the temperature rate of change at time τ.

The temperature rate of change at time τ in Equation 3.1 is given by:

$$\left.\frac{\partial \mathbf{t}(\theta)}{\partial \theta}\right|_{\theta=\tau} = -\mathbf{C^{-1}}\mathbf{G}(\tau) \cdot \mathbf{t}_\tau + \mathbf{C^{-1}}\mathbf{p}_\tau \tag{3.4}$$

while, according to Equation 3.3, we have that:

$$\left.\frac{\partial \mathbf{t}(\theta)}{\partial \theta}\right|_{\theta=\tau} = \mathbf{A}'(\tau) \cdot \mathbf{t}_\tau + \mathbf{B}' \cdot \mathbf{p}_\tau + \mathbf{w}' \tag{3.5}$$

by solving the system composed by Equation 3.4 and 3.5, we have that:

$$\mathbf{A}'(\tau) = -\mathbf{C^{-1}}\mathbf{G}(\tau) \tag{3.6}$$

$$\mathbf{B}' = \mathbf{C^{-1}} \tag{3.7}$$

where matrix $\mathbf{A}'(\tau)$ expresses the part of the on-chip temperature spreading process that depends only on the cell's temperature. This matrix models the thermal conductances and capacitances network of Figure 3.1 in the matrix form except K_{cu-RT}, which needs to be modeled separately. Matrix $\mathbf{B}'$ is a matrix where $\mathbf{B'_{i,j}}$ contains the conversion factor between the power assigned to functional unit j and the temperature increase in cell i. Matrices $\mathbf{A}'(\tau)$ and $\mathbf{B}'$ contain the system dynamics that depend entirely on the current state and on the given power assignment vector $\mathbf{p}_\tau$. The part of the dynamic system that is not controllable by the input vector, such as the heat dissipation of the copper layer due to room temperature, is expressed by vector **w**'. Then, by substituting 3.5 into 3.3, Eqn. 3.8 is obtained:

$$\mathbf{t}_{\tau+\boldsymbol{\Delta}\tau} = \mathbf{A}(\tau) \cdot \mathbf{t}_\tau + \mathbf{B} \cdot \mathbf{p}_\tau + \mathbf{w} \tag{3.8}$$

where:

$$\mathbf{A}(\tau) = \mathbf{I} + \mathbf{\Delta}\tau\mathbf{A}'(\tau) \tag{3.9}$$
$$\mathbf{B} = \Delta\tau\mathbf{B}' \tag{3.10}$$
$$\mathbf{w} = \Delta\tau\mathbf{w}' \tag{3.11}$$

Equation 3.8 is a time-varying state-space representation modeling the thermal behavior of the MPSoC using a first order ODE solver. The computation here is simple and requires only a matrix multiplication. The total number of cells in all layers of the 3D MPSoC structure is n, the total number of independent processing units is p and the total number of independent cooling liquid flow rates is z.

An important property of a solver is its "stability". An integration method is called numerically stable if an error does not exponentially grow during the calculation of the final solution. The FE method has potential stability problems when the chosen time-step for the thermal simulations is large, as shown in the literature [17], which will be explored and addressed in this paper.

A first order ODE method that is unconditionally stable is the *Backward Euler*(BE) method. This method is described by Equation 3.12:

$$\mathbf{t}_{\tau+\mathbf{\Delta}\tau} = \mathbf{t}_\tau + \mathbf{\Delta}\tau \cdot \left.\frac{\partial \mathbf{t}(\theta)}{\partial\theta}\right|_{\theta=\tau+\mathbf{\Delta}\tau} \tag{3.12}$$

Assuming $\mathbf{A}'(\tau) \simeq \mathbf{A}'(\tau + \mathbf{\Delta}\tau)$, and using Eqn. 3.13:

$$\left.\frac{\partial \mathbf{t}(\theta)}{\partial\theta}\right|_{\theta=\tau+\Delta\tau} = \mathbf{A}'(\tau + \mathbf{\Delta}\tau) \cdot \mathbf{t}_\tau + \mathbf{\Delta}\tau + \mathbf{B}' \cdot \mathbf{p}_\tau + \mathbf{w}' \tag{3.13}$$

we can obtain Eqn. 3.14 in a discrete-time domain:

$$\mathbf{t}_{\tau+\mathbf{\Delta}\tau} = \mathbf{A}(\tau) \cdot \mathbf{t}_\tau + \mathbf{B}(\tau) \cdot \mathbf{p}_\tau + \mathbf{w}(\tau) \tag{3.14}$$

where:

$$\mathbf{A}(\tau) = [\mathbf{I} - \mathbf{\Delta}\tau\mathbf{A}'(\tau)]^{-\mathbf{1}} \tag{3.15}$$
$$\mathbf{B}(\tau) = [\mathbf{I} - \mathbf{\Delta}\tau\mathbf{A}'(\tau)]^{-\mathbf{1}}\mathbf{\Delta}\tau\mathbf{B}' \tag{3.16}$$
$$\mathbf{w}(\tau) = [\mathbf{I} - \mathbf{\Delta}\tau\mathbf{A}'(\tau)]^{-\mathbf{1}}\mathbf{\Delta}\tau\mathbf{w}' \tag{3.17}$$

This method achieves unconditional stability at the cost of a significant increase in computational complexity with respect to the FE algorithm. The most expensive computational step for the BE method is the inverse matrix computation. It has been shown in the literature [64],[81] that the accuracy of both first order methods is $O(\Delta\tau^2)$ [17].

3.1.3 Second order ODE solvers

As a representative example of second order solvers, we analyze the *Crank-Nicholson*(CN) method also called trapezoidal. This integration algorithm combines FE with BE to obtain a second order method due to cancellation of the error terms. CN method reaches an accuracy of $O(\Delta\tau^2)$. This method can be described using Eqn. 3.18:

$$\mathbf{t}_{\tau+\boldsymbol{\Delta}\tau} = \mathbf{t}_\tau + \frac{\boldsymbol{\Delta}\tau}{\mathbf{2}} \cdot \left[\left.\frac{\partial \mathbf{t}(\theta)}{\partial\theta}\right|_{\theta=\tau} + \left.\frac{\partial \mathbf{t}(\theta)}{\partial\theta}\right|_{\theta=\tau+\boldsymbol{\Delta}\tau} \right] \tag{3.18}$$

Assuming $\mathbf{A}'(\tau) \simeq \mathbf{A}'(\tau + \boldsymbol{\Delta}\tau)$, and using Eqn. 3.5 and 3.13, we obtain:

$$\mathbf{t}_{\tau+\boldsymbol{\Delta}\tau} = \mathbf{A}(\tau) \cdot \mathbf{t}_\tau + \mathbf{B}(\tau) \cdot \mathbf{p}_\tau + \mathbf{w}(\tau) \tag{3.19}$$

where:

$$\mathbf{A}(\tau) = [\mathbf{I} - \mathbf{0.5} \cdot \boldsymbol{\Delta}\tau\mathbf{A}'(\tau)]^{-\mathbf{1}} \cdot [\mathbf{I} + \mathbf{0.5} \cdot \boldsymbol{\Delta}\tau\mathbf{A}'(\tau)] \tag{3.20}$$

$$\mathbf{B}(\tau) = [\mathbf{I} - \mathbf{0.5} \cdot \boldsymbol{\Delta}\tau\mathbf{A}'(\tau)]^{-\mathbf{1}} \cdot \boldsymbol{\Delta}\tau\mathbf{B}' \tag{3.21}$$

$$\mathbf{w}(\tau) = [\mathbf{I} - \mathbf{0.5} \cdot \boldsymbol{\Delta}\tau\mathbf{A}'(\tau)]^{-\mathbf{1}} \cdot \boldsymbol{\Delta}\tau\mathbf{w}' \tag{3.22}$$

The CN method is stable and has a higher accuracy in comparison to first order solvers when larger simulation time-steps are used. The accuracy of such second order methods is $O(\Delta\tau^3)$ [17].

3.1.4 Multi-step fourth order ODE solver

Numerical ODE solution methods, start from an initial point and take a small step in time to find the next solution point. This process continues with subsequent steps to compute the solution. Single-step methods (such as Euler's method) refer to only one previous point and its derivative to determine the current value. Multi-step methods take several intermediate points within every simulation step to obtain a higher order method. This way, they increase the accuracy of the approximation of the derivatives by using a linear combination of these internal additional points. A multi-step solver has been embedded in the version of 4.0 of HotSpot [81]. One particular subgroup of this family of multi-step solvers is the *Runge-Kutta*(RK4) method, which includes a fourth order solver. The algorithm that we use for implementing the RK4 solver employs a FE method to compute derivatives at the internal points. By using the model represented in Figure 3.1, this method is described by following equations:

$$\mathbf{k_1} = \Delta\tau \cdot [\mathbf{A}'(\tau)\mathbf{t}_\tau + \mathbf{B}' \cdot \mathbf{p}\tau + \mathbf{w}'] \tag{3.23}$$

$$\mathbf{k_2} = \Delta\tau \cdot [\mathbf{A}'(\tau + \mathbf{0.5}\boldsymbol{\Delta}\tau)(\mathbf{t}_\tau + \mathbf{0.5} \cdot \mathbf{k_1}) + \mathbf{B}' \cdot \mathbf{p}\tau + \mathbf{w}'] \tag{3.24}$$

$$\mathbf{k_3} = \Delta\tau \cdot [\mathbf{A}'(\tau + \mathbf{0.5}\boldsymbol{\Delta}\tau)(\mathbf{t}_\tau + \mathbf{0.5} \cdot \mathbf{k_2}) + \mathbf{B}' \cdot \mathbf{p}\tau + \mathbf{w}'] \tag{3.25}$$

$$\mathbf{k_4} = \Delta\tau \cdot [\mathbf{A}'(\tau + \boldsymbol{\Delta}\tau)(\mathbf{t}_\tau + \mathbf{k_3}) + \mathbf{B}' \cdot \mathbf{p}\tau + \mathbf{w}'] \tag{3.26}$$

$$\mathbf{t}_{\tau+\boldsymbol{\Delta}\tau} = \mathbf{t}_\tau + \frac{\mathbf{1}}{\mathbf{6}} \cdot (\mathbf{k_1} + \mathbf{2k_2} + \mathbf{2k_3} + \mathbf{k_4}) \tag{3.27}$$

Assuming $\mathbf{A}'(\tau) \simeq \mathbf{A}'(\tau + \mathbf{0.5\Delta}\tau) \simeq \mathbf{A}'(\tau + \mathbf{\Delta}\tau)$, we obtain:

$$\mathbf{t}_{\tau+\mathbf{\Delta}\tau} = \mathbf{A}(\tau) \cdot \mathbf{t}_\tau + \mathbf{B}(\tau) \cdot \mathbf{p}\tau + \mathbf{w}(\tau) \tag{3.28}$$

where:

$$\mathbf{F} = \Delta_t \mathbf{A}'(\tau) \tag{3.29}$$

$$\mathbf{A}(\tau) = \mathbf{I} + \frac{\mathbf{1}}{\mathbf{6}}[\mathbf{6F} + \mathbf{3F^2} + \mathbf{F^3} + \mathbf{0.25F^4}] \tag{3.30}$$

$$\mathbf{B}(\tau) = \Delta\tau \cdot [\mathbf{I} + \frac{\mathbf{1}}{\mathbf{6}}(\mathbf{3F} + \mathbf{F^2} + \mathbf{0.25F^3})] \cdot \mathbf{B}' \tag{3.31}$$

$$\mathbf{w}(\tau) = \Delta\tau \cdot [\mathbf{I} + \frac{\mathbf{1}}{\mathbf{6}}(\mathbf{3F} + \mathbf{F^2} + \mathbf{0.25F^3})] \cdot \mathbf{w}' \tag{3.32}$$

Note that this method does not require the inverse matrix computation. In addition, like FE, this method is not unconditionally stable, since RK4 method uses the FE for computing the rate of change of the temperature function in the internal point, and hence inherits its stability properties. The accuracy of this multi-step fourth order method can reach $O(\Delta_t^5)$ [17].

3.1.5 Changing the sampling rate

Equation 3.8 models the RC network representing the MPSoC behavior. To allow high accuracy in the discrete integration process, the sampling sampling period between $\mathbf{t}_\tau$ and $\mathbf{t}_{\tau+\mathbf{\Delta}\tau}$ has to be small (i.e. $10\mu s - 200\mu s$). A MPSoC heating process makes relevant changes in a time range that is several orders of magnitude the simulation time step cited before. This requires many iterations of Equation 3.8 and this is costly if, for each step, there is a thermal policy computation associated with it. To increase the sampling rate to $\Delta'\tau = v \cdot \Delta\tau$ without changing the value of $\Delta\tau$ the following mathematical derivation is presented.

Assuming that the input $\mathbf{p}$ does not change during $\Delta'\tau$ as well as matrices $\mathbf{A}$, $\mathbf{B}$ and vector $\mathbf{w}$, we have that Equation 3.8 turns into Equation 3.33:

$$\mathbf{t}_{\tau+\mathbf{\Delta}\tau} = \mathbf{A} \cdot \mathbf{t}_\tau + \mathbf{B} \cdot \mathbf{p} + \mathbf{w} \tag{3.33}$$

by iterating Equation 3.33, we have that:

$$\mathbf{t}_{\tau+\mathbf{2}\cdot\mathbf{\Delta}\tau} = \mathbf{A} \cdot [\mathbf{A} \cdot \mathbf{t}_\tau + \mathbf{B} \cdot \mathbf{p} + \mathbf{w}] \cdot \mathbf{t}_\tau + \mathbf{B} \cdot \mathbf{p}_\tau + \mathbf{w} \tag{3.34}$$

by iterating 3.34 using equation 3.33, we have that at step 3:

$$\mathbf{t}_{\tau+\mathbf{3}\cdot\mathbf{\Delta}\tau} = \mathbf{A} \cdot [\mathbf{A} \cdot [\mathbf{A} \cdot \mathbf{t}_\tau + \mathbf{B} \cdot \mathbf{p} + \mathbf{w}] \cdot \mathbf{t}_\tau + \mathbf{B} \cdot \mathbf{p} + \mathbf{w}] \cdot \mathbf{t}_\tau + \mathbf{B} \cdot \mathbf{p}_\tau + \mathbf{w} \tag{3.35}$$

The way 3.33 has evolved to Equation 3.35, we derive the following generalization that holds for any step v:

$$\mathbf{t}_{\tau+\mathbf{v}\cdot\mathbf{\Delta}\tau} = \mathbf{A^v}\mathbf{t}_\tau + \sum_{\mathbf{i=0}}^{\mathbf{v-1}} \mathbf{A^i}\mathbf{Bp} + \sum_{\mathbf{i=0}}^{\mathbf{v-1}} \mathbf{A^i}\mathbf{w} \tag{3.36}$$

exploiting matrix series properties we have that:

$$\begin{aligned}\mathbf{t}_{\tau+\mathbf{v}\cdot\boldsymbol{\Delta}\tau} &= \mathbf{A}^{\mathbf{v}}\mathbf{t}_{\tau} + [(\mathbf{I}-\mathbf{A})^{-\mathbf{1}}(\mathbf{I}-\mathbf{A}^{\mathbf{v}})]\mathbf{B}\mathbf{p} + \\ &\quad +[(\mathbf{I}-\mathbf{A})^{-\mathbf{1}}(\mathbf{I}-\mathbf{A}^{\mathbf{v}})]\mathbf{w} \end{aligned} \tag{3.37}$$

where I is the identity matrix with the same size of the square matrix A. By comparing eq. 3.37 with equation 3.33, we have that:

$$\mathbf{t}_{\tau+\mathbf{v}\cdot\boldsymbol{\Delta}\tau} = \mathbf{A}_{\mathbf{v}}\mathbf{t}_{\tau} + \mathbf{B}_{\mathbf{v}}\mathbf{p} + \mathbf{w}_{\mathbf{v}} \tag{3.38}$$

where

$$\begin{aligned}\mathbf{A}_{\mathbf{v}} &= \mathbf{A}^{\mathbf{v}} \\ \mathbf{B}_{\mathbf{v}} &= [(\mathbf{I}-\mathbf{A})^{-\mathbf{1}}(\mathbf{I}-\mathbf{A}^{\mathbf{v}})]\mathbf{B} \\ \mathbf{w}_{\mathbf{v}} &= [(\mathbf{I}-\mathbf{A})^{-\mathbf{1}}(\mathbf{I}-\mathbf{A}^{\mathbf{v}})]\mathbf{w} \end{aligned} \tag{3.39}$$

It is important to notice that this derivation holds only if the input $\mathbf{p}$ does not change during $\Delta'\tau = v \cdot \Delta\tau$ as well as matrices $\mathbf{A}$, $\mathbf{B}$ and vector $\mathbf{w}$. The assumption that matrix $\mathbf{A}$ does not change is an approximation that is as close to reality as the thermal profile does not make big variations in $\Delta'\tau$.

3.2 Thermal Simulation Analysis

In this section we provide a theoretical analysis about the effects of the parameters in the ODE solvers on the speed and the accuracy of simulation results. Then, we provide an experimental validation with the case study described in Section 5.2 of this thesis.

3.2.1 Stability analysis

Explicit integration methods where the inverse matrix calculation is not employed suffer from a potential instability problem. Instability means that the integration error can grow exponentially at every simulation step. This issue occurs when the simulation time step is bigger or comparable to the inverse of the maximum absolute temperature rate of change of the MPSoC [17]. The maximum allowable simulation step size Δ_{max} is then given by the following equation:

$$\Delta_{max} \approx \frac{1}{max\left|\frac{\partial \mathbf{t}(\theta)}{\partial \theta}\right|} \tag{3.40}$$

where $max\left|\frac{\partial \mathbf{t}(\theta)}{\partial \theta}\right|$ is the maximum temperature rate of change. In this section our goal is to find the maximum allowable simulation step size that avoids instability problems in the solver, assuming a worst-case scenario.

According to our analysis, based on the model shown in Figure 3.1, the highest worst case temperature rate of change in cell S_{i-0} occurs when the following conditions are true:

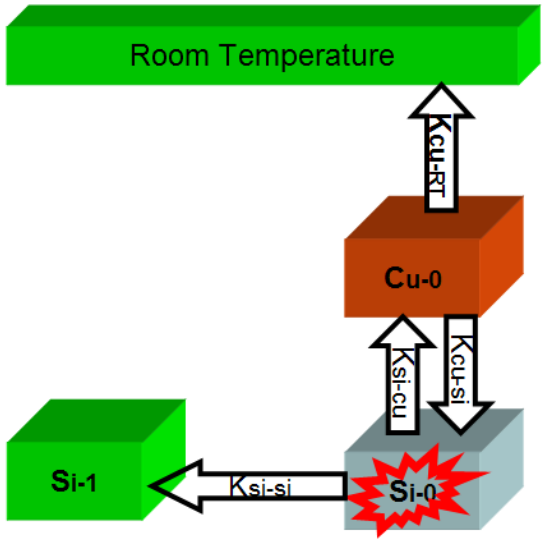

Figure 3.3: Circuit for the determination of ΔT_{cs} and ΔT_{ss}.

1. S_{i-0} is at room temperature
2. S_{i-0} has the highest power dissipated per cell P_{max}
3. Between S_{i-0} and C_{u-0}, there exists the maximum temperature difference possible, ΔT_{cs}.
4. Between S_{i-0} and neighboring cells, there exists the maximum temperature difference possible, ΔT_{ss}.

In this particular case, the maximum temperature rate of change is expressed by the following equation:

$$max\left|\frac{\partial \mathbf{t}(\theta)}{\partial \theta}\right| = \frac{\frac{P_{max}\Delta\tau}{C(S_{i-0})} + 4K_{si-si}\Delta T_{ss} + K_{si-cu}\Delta T_{cs}}{\Delta\tau} \tag{3.41}$$

where P_{max} is the highest power dissipated per cell and $C(S_{i-0})$ is the silicon cell thermal capacitance and $\Delta\tau$ the simulation time step. To determine the value of ΔT_{cs} and ΔT_{ss}, we use the structure of Figure 3.3.

This structure maximizes the temperature differences between the cell S_{i-0} and the other two cells (S_{i-1} and C_{u-0}) by modeling the scenario where the left half side of the cells in the floorplan are not dissipating any power and the right half side has the highest power density in the circuit. Thus, cell S_{i-1} is at room temperature (T_{amb}) and cell S_{i-1} is consuming P_{max}. Our target is to compute the temperature of S_{i-1} and C_{u-0} at the equilibrium point when the highest temperature differences are present between cells. When the transient response is finalized, the result is:

$$\Delta T_{ss} = \mathbf{t(S_{i-0})} - \mathbf{t(S_{i-1})} = \mathbf{t(S_{i-0})} - \mathbf{T_{amb}} \tag{3.42}$$

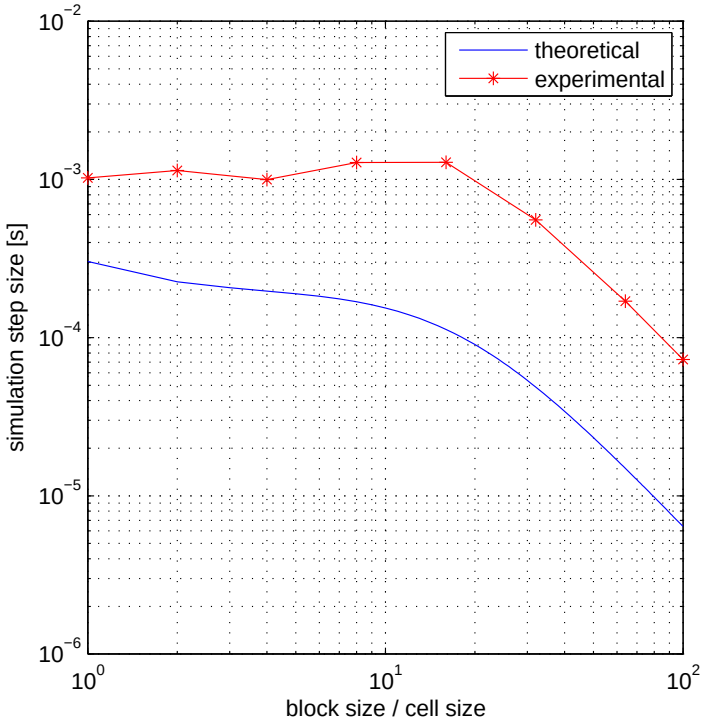

Figure 3.4: Experimental vs. theoretical values of Δ_{max} for various grid resolution values.

where $\mathbf{t(j)}$ is the temperature of cell j. According to preceding equations, to solve Equation 3.41 we need to determine the temperature value of cells S_{i-0}. By analyzing and solving the circuit in Figure 3.3, we obtain:

$$\Delta T_{ss} = \frac{\frac{P_{max}\Delta\tau}{C(S_{i-0})}(k_{cu-si} + k_{cu-RT})}{k_{cu-si}k_{si-si} + k_{cu-RT}(k_{si-cu} + k_{si-si})} \tag{3.43}$$

To determine the value of ΔT_{cs}, a different scenario needs to be considered. The scenario assumes that all silicon cells are consuming P_{max}. The structure is equivalent to the previous one in Figure 3.3, but this time without the arrow K_{si-si} or the cell S_{i-1}. In this case, at the equilibrium, the following equation holds:

$$\Delta T_{cs} = \mathbf{t(S_{i-0})} - \mathbf{t(C_{u-0})} \tag{3.44}$$

By solving the previously described network, we obtain:

$$\Delta T_{cs} = \frac{P_{max}\Delta\tau}{k_{si-cu}C(S_{i-0})} \tag{3.45}$$

Then, by substituting Eqn. 3.45, 3.43 and 3.41 into 3.40, we are able to compute Δ_{max}. The resulting function is a highly non-linear function that depends on parameters such as MPSoC thermal profile, thermal cell size and dimensions of the MPSoC.

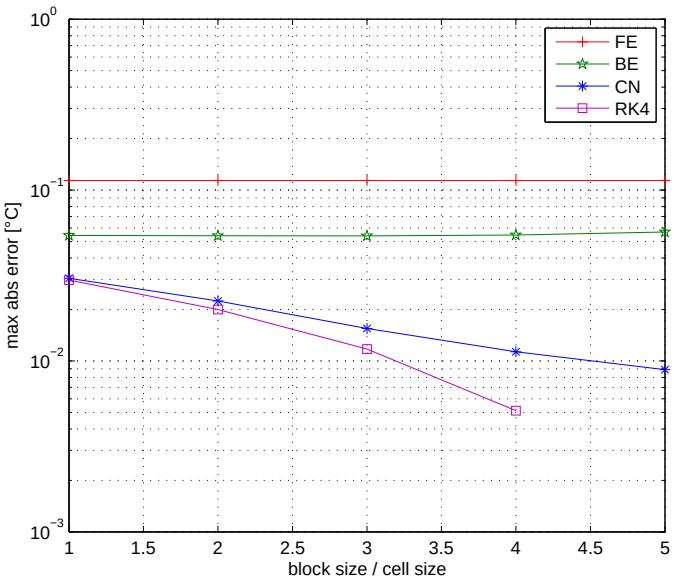

Figure 3.5: Accuracy vs. grid resolution of the floorplan.

In addition to the theoretical derivations, we simulated the 8-core MPSoC described in Section 5.2 using an explicit solver for various simulation step sizes ($\Delta\tau$). We then identified Δ_{max} as the largest $\Delta\tau$ value for which the integration method was working without getting unstable. We repeated the simulation for various grid resolution values. The comparisons between the theoretical and the experimental results are shown in Figure 3.4.

Results show that the theoretical analysis is in line with the experimental simulations, as both set of results have the same trends and the difference of values is very small. The reason of the small difference is that worst case scenario assumptions make our theoretical result more conservative: i.e., Δ_{max} is 7% lower in the 8-core case study described in Section 5.2. Note that the theoretical derivation using Eqn. 3.45, 3.43, 3.41 and 3.40 is much faster to compute than performing an experimental derivation of Δ_{max}.

3.2.2 Cell size influence

The cell size is the parameter that mostly affects the speed of the simulation. The computational complexity N_{op} is related to the grid granularity according to following equation:

$$N_{op} = N_0 \cdot \left(\frac{block\ size}{cell\ size} \right)^2 \tag{3.46}$$

where N_0 is the computational complexity to process the floorplan using a cell size equal to the smallest functional block of the MPSoC. As shown in Eqn. 3.46, the computational complexity increases quadratically with a linear

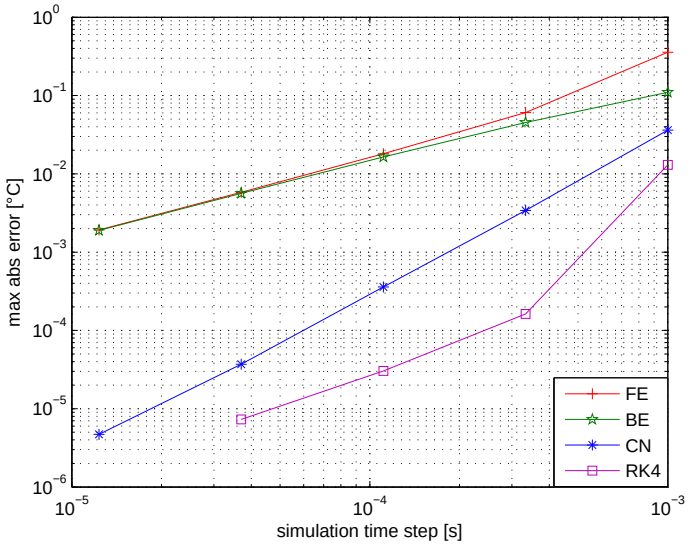

Figure 3.6: Accuracy vs. simulation time step Δ_τ.

increase in the grid resolution. Figure 3.5 shows the accuracy of the thermal model for the various solvers we have discussed.

As Figure 3.5 shows, with a high simulator order, the advantage of increasing the grid resolution is more obvious than the advantage observed for low-order simulators. In fact, for low accuracy solvers (such as FE or BE), the increase in accuracy is almost irrelevant considering the quadratic increase in computational complexity.

3.2.3 Simulation time step

In the previous section, we determined the value of Δ_{max}. For $\Delta\tau > \Delta_{max}$, explicit methods like the FE or the RK4 are unstable. On the other hand, if the simulation time step is reduced, the explicit methods obtain an increase in accuracy, but at a higher computational cost. The computational cost N_{op} is related to the time step size $\Delta\tau$, as shown in equation 3.47:

$$N_{op} = N_0 \cdot \left(\frac{\Delta_{\tau 0}}{\Delta_\tau} \right) \tag{3.47}$$

where N_0 is the computational complexity using a step size equal to $\Delta_{\tau 0}$. The computational cost is inversely proportional to Δ_τ. Figure 3.6 shows the accuracy change with respect to simulation step size.

The effect of decreasing Δ_τ is visible for all the solvers. In addition, the cost in terms of computational effort is relatively small with respect to the gain in accuracy. We also observe that usually more than one solver can meet the given error limit. For example, to achieve a desired maximum error of $10^{-2}\ ^\circ C$, the first order solver requires $\Delta_\tau \simeq 6 \cdot 10^{-5}$, the second order solver requires

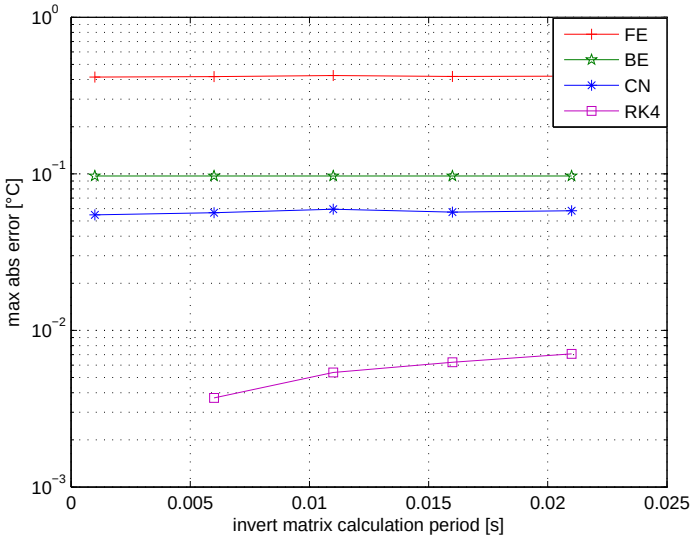

Figure 3.7: Accuracy vs. matrix calculation period.

$\Delta_\tau \simeq 5 \cdot 10^{-4}$, and the fourth order solver requires $\Delta_\tau \simeq 10^{-3}$. The method we present in Section 5 automatically identifies which option provides the fastest simulation result, according to the properties of the particular MPSoC architecture.

3.2.4 Matrix calculation period

All matrices describing the heat propagation inside the MPSoC are temperature dependent and they change over time during the runtime operation of the MPSoC. If the matrices are not computed sufficiently often, this can reduce accuracy. However the computation of these matrices takes time and slow down the simulation time.

The parameter that influences the accuracy / simulation time trade-off is the matrix calculation period (T_{MC}). This is the time that passes between two consecutive generations of matrices $A(\tau)$, $B(\tau)$ and $W(\tau)$. Figure 3.7 quantifies this error for different solvers and T_{MC} values.

As Figure 3.7 shows, for higher order solvers, reducing T_{MC} becomes more beneficial than reducing T_{MC} for low-order solvers. For low accuracy solvers, the increase of accuracy is not high enough to justify the increase in computational complexity.

3.3 Liquid Cooling

In 3D stacks, cooling cannot be handled and managed by conventional air cooling methods over the stack surface because of the large heat propagation. Interlayer liquid cooling is a potential solution to address thermal problems.

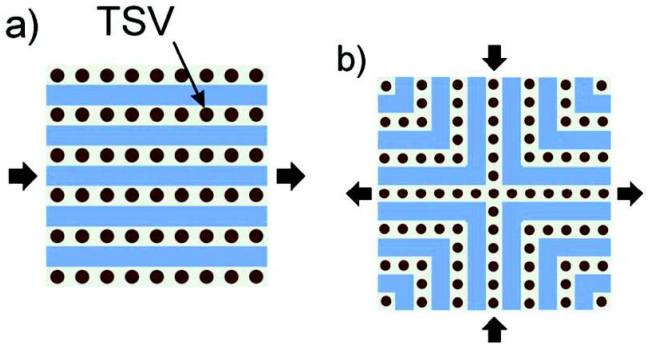

Figure 3.8: Top view of a) 2-port and b) 4-port microchannel fluid delivery architecture compatible with area-array interconnects.

Brunschwiler et al. [14] reported that liquid cooling solutions offer a higher heat removal capability in comparison to air and to the possibility to extract heat at various layers of the stack. Several works described in Chapter 2 by Bhunia et al. [6] and Lee et al. [47] have explored the feasibility of liquid cooling as cooling method for 3D MPSoCs. Thermal management methods using this technology have been recently proposed by Coskun et al. [20] and Sabry et al. [77]. This section shows how these structures are modeled.

3.3.1 Straight and bent microchannels

There are several ways to support liquid cooling, e.g., by adding/inserting to the stack a plate with built-in microchannels and/or by etching a porous-media structure between the tiers of the 3D stack [14, 15]. Experiments have shown that when a coolant fluid is pumped through the microchannels, up to $3.9 KW/cm^3$ [15] of heat can be extracted.

Porous-media structures can be designed with different forms according to the TSVs spacing requirements and the desired fluidic path [84, 83]. Figure 3.8 shows a planar view of two different structures. Although these structures use microchannels to guide the fluid, one of them uses straight channels with two ports (Fig. 3.8.a), while the other exploits bent channels and four ports (Fig. 3.8.b). In the following we will refer to these structures as 'straight' and 'bent' channels. Using multi-port bent channels is more beneficial than using straight channels, if the straight channel length is longer than the thermal developing length of the fluid [15]. Moreover, bent channels have different lengths, thus enabling different liquid flow rates between different channels.

Overall, thermal management of a 3D stack is achieved by a combination of active control of on-chip switching rates (the heat source) as well as active interlayer cooling with pressurized fluids (the heat sink). It is important to remember that the cooling system requires one (or more) pumps to circulate the fluid, as well as a heat exchanger to cool the fluid. The latter may be

passive (e.g., fin structure) or active (e.g., fan). At any rate, a relevant part of the system energy spent for cooling is due to the pump [77] and a minor part by the exchanger.

3.3.2 Interlayer cooling layer modeling

In this thesis, we extend previous compact modeling concept proposed by Sridhar et al. [83] to account for two major factors.

First, the fluid flow is no longer constant among the channels of the same layer, but it is related to the channel length [14]. Thus, different lengths of the channels lead to different fluid velocities [15]. The channels with the smallest length have the highest fluid velocity, while the longest channels have the lowest velocity.

Second, the fluid flow is not a single-dimensional flow. In fact, we allow the fluid to flow with more flexibility, given that the used porous-media is microchannels. Thus, the fluid enters from a direction that lies in one Cartesian axis (e.g., south) and leaves from another direction that lies on another axis (e.g., east).

In addition, in the target 3D-MPSoC stacks the microchannels have different lengths, which implies that the pumped flow rate is not distributed homogeneously between the microchannels.

Flow Rate and Channel Length

The relation between the flow rate Fl and the channel length L is as follows:

$$Fl = w_{ch} \cdot h_{ch} \cdot \nu_{bulk} \tag{3.48}$$

$$\nu_{bulk} = \frac{\nu_{darcy}}{\varepsilon} \tag{3.49}$$

$$\nu_{darcy} = \frac{\kappa}{\mu} \cdot \nabla P \tag{3.50}$$

$$\nabla P = \frac{\Delta P}{L} \tag{3.51}$$

where ν_{bulk} is the actual fluid velocity and ν_{darcy} is the fluid velocity multiplied by the cavity porosity. Parameters used in Equations 3.48-3.51 are shown in Table 3.1. Hence, the flow rate and channel length are inversely proportional, i.e., the shorter the channel length is, the higher the flow rate is. We validate the flow velocity obtained for each channel comparing the analytical model in Equations 3.48-3.51, with the experimental values shown in [15]. As Fig. 3.9 shows, the proposed analytical model provides us an acceptable method to calculate the flow rate for different channel lengths.

Thermal Capability and Pumping Power

Since we use varying flow rate as a control variable for energy-efficient thermal management, it is crucial to study the thermal capability of interlayer liquid

Table 3.1: Parameters definition used to relate the flow rate to the channel length

Parameter	Definition
w_{ch}	Channel width ($50\mu m$)
h_{ch}	Channel height ($100\mu m$)
ε	Cavity porosity (0.5)
κ	Cavity permeability ($7.17E-11m^2$)
μ	Dynamic viscosity ($1E-3Pascal \cdot sec$)
ΔP	Pressure difference between the inlet and outlet ports (1 bar)

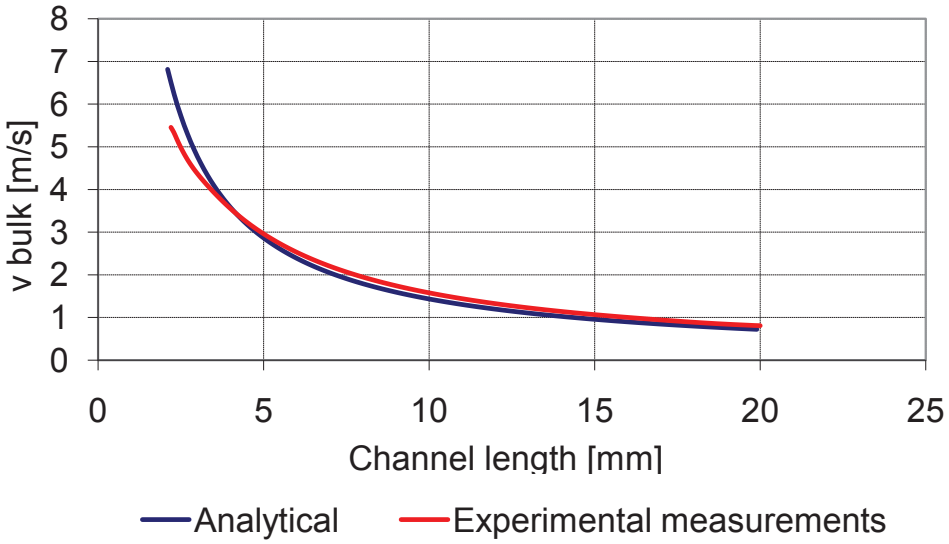

Figure 3.9: Comparison of the fluid flow velocity in different channel lengths between the analytical method (Equations 3.48-3.51) and the experimental results shown in Brunschwiler et al. [15].

cooling with respect to different pumping power values. Thus, each pumping power value is translated to a specific flow rate in our system. First, we use Bernoulli's equation to describe the pump power P_{pump} as follows:

$$P_{pump} = \frac{\Delta P \cdot Fl}{\zeta} \tag{3.52}$$

where ΔP is the pressure difference required, Fl is the fluid flow rate, and ζ is the pumping power efficiency. We use $\zeta = 0.7$, as it is a normal pump efficiency value [45, 29]. Since there is a linear relation between the pressure difference and the flow rate injected in the stack (Equations 3.48- 3.51), we can say that $P_{pump} \propto Fl^2$.

Next, we define the thermal capability of interlayer liquid cooling as the maximum heat flux absorbed by the fluid to keep the maximum temperature within the stack below 85^oC. To estimate this thermal capability, we use 3D-ICE [83] to record the maximum temperature of the stack at different thermal dissipation values, and with different flow rates. We limit the maximum flow

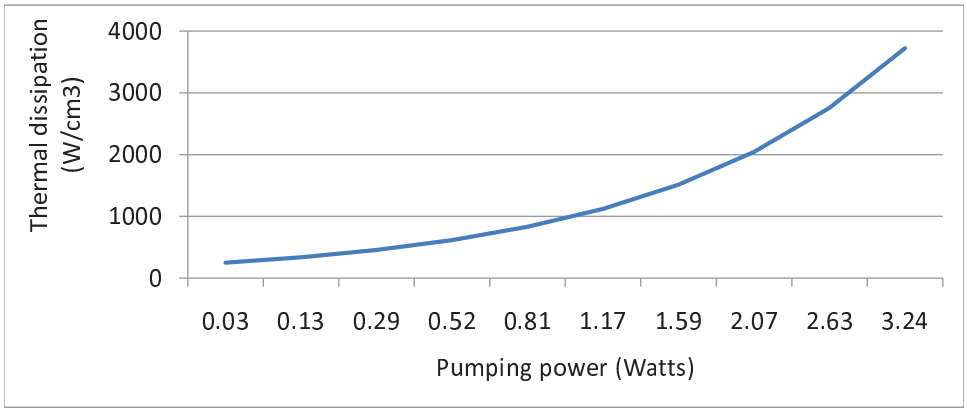

Figure 3.10: Rate of change of thermal capability of interlayer liquid cooling TC with respect to pumping power P_{pump}.

rate injected to be the one at $\Delta P = 1bar$, since it is the maximum safe pressure requirement within the stack [14].

Therefore, Fig. 3.10 shows the amount of minimum pumping power applied to keep the maximum temperature of the stack below 85^oC, at different thermal dissipation rates.

Flow Rate and Heat Extraction

The amount of heat r_i extracted in cell i by the fluid in the microchannel controlled by pump j can be approximated by the following equation:

$$r_i = m_j \cdot \gamma_{i,j} \cdot (t_i - t_{fluid}) \tag{3.53}$$

where the fluid temperature is t_{fluid}, t_i is the temperature of cell i and $\gamma_{i,j}$ is the constant modeling the channel heat extraction properties. Vector $\mathbf{m} \epsilon \Re^{\mathbf{z}}$ is the normalized amount of heat that can be extracted for each of the z independent pumps.

Thus, by varying vector $\mathbf{m}$, the cooling power (flow rate of the cooling liquid) is varied to achieve the desired heat extraction. In our model, we used the temperature mapping from [83] to derive $\gamma_{i,j}$.

Experiments have shown that by updating $\gamma_{i,j}$ every time the policy is applied ($10ms$ in our simulation setup), our approximation leads to a maximum error up to $\pm 5\%$.

3.4 Summary

This chapter has presented the derivation of all the models used in this thesis to describe formally the thermal and cooling models of a generic MPSoC.

The first section has described mathematically how the thermal properties of the MPSoC can be represented by differential equations using a standard state-space representation.

The second section has provided a theoretical and experimental analysis about the effects of the parameters in the ODE solvers on the speed and the accuracy of simulation results.

The third section has presented the way the liquid cooling is modelled in this thesis. Straight and bent microchannels formal models have been presented.

Energy and Workload Models

4

In this chapter I give an introduction to the concepts I developed to model the workload of an MPSoC system. Moreover I make some considerations about the system energy models I used in this thesis.

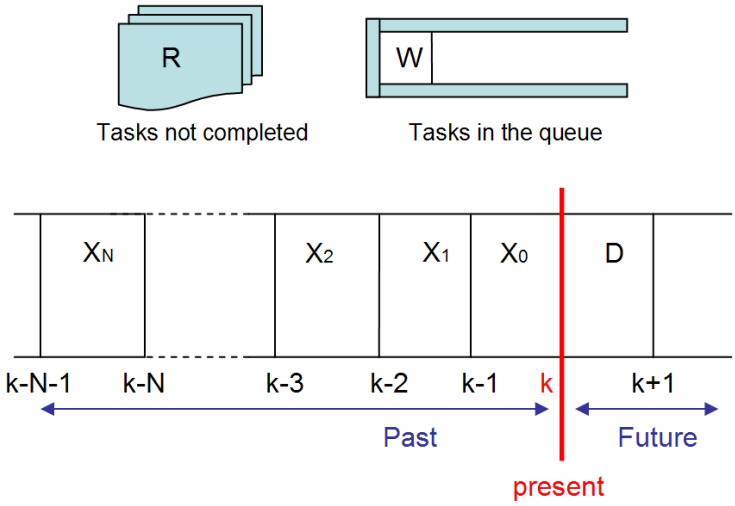

Figure 4.1: Scheduler viewpoint snapshot of the MPSoC at time k.

4.1 System Energy

Power saving, improving performance and increase chip reliability are three important challenges facing MPSoC designers. These three objectives must be reconciled with the fact that the MPSoC has to execute all the workload requested from the scheduler with the minimum power consumption and the smallest acceptable latency.

Figure 4.1 shows the status of the system from the scheduler point of view at a certain fixed time point k. The frequency of the sampling point is the frequency with which thermal policies are applied. The term R stands for tasks that are still in the processor and need to be completed, W are tasks arrived, waiting to be executed. X_i is the task workload arrived i previous time steps. N is the length of the past task arrival history. D is the task workload that will arrive in the next time step between the actual one k and the next one $k+1$.

Since the task arrival is a stochastic process, a prediction must be made in relation to the the workload that will be required by the scheduler in the next policy observation period D. If the scheduler employs a dynamic scheduling in the time frame corresponding to a window, it is not possible to know the workload in time frame D at time k in advance.

The ideal solution would be a method able to exactly estimate the number of tasks arriving in the next time slot. The better the estimation is, the better the performance of the thermal management system will be.

4.1.1 Energy efficiency quantification

We can assume a linear relation between the frequency of operation of a core and the amount of instructions it executes (Choi et al. [18]). We consider

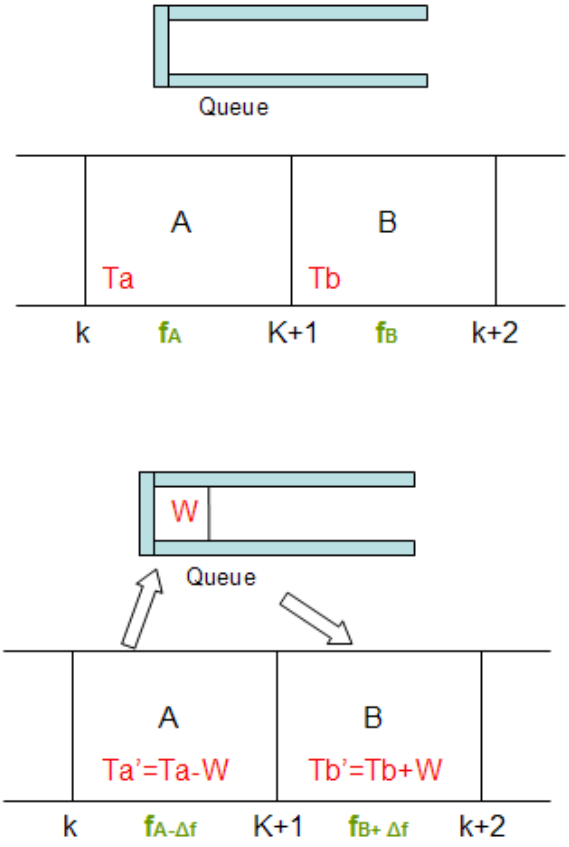

Figure 4.2: Effect of a frequency estimation error from a frequency perspective in a DVFS system. (top): ideal case; (bottom) real case with an estimation error Δf.

now two consecutive window frames A and B and we define as f_A and f_B the frequencies that will fulfill the scheduler requirements in the two consecutive time window A and B. The picture in top of Figure 4.2 shows the setup. As it can be noted the queue is empty since all tasks Ta and Tb are executed respectively in window A and B.

To compute the power consumption in this ideal case we use derivations from Rao et al. [72]. In his work, he showed that the relation between the frequency and the power can be approximated as a polynomial function with degree larger than one. The formula is quite complex. A good approximation is represented by the following Equation:

$$P(f_A) = K_p \cdot f_A^{\alpha} + L \tag{4.1}$$

where f_A is a frequency value, K_p, α, L constants, and $P()$ is the function that relates the frequency of a functional unit to its power dissipation. The term $K_p \cdot f_A^{\alpha}$ models the active power consumption. The constant term L models the part of the power that does not depend on the frequency such as the leakage power consumption.

Using Equation 4.1, the power consumption P_{ideal} due to the execution of tasks Ta and Tb in this scenario is given by the following equation

$$P_{ideal} = K_P \cdot f_A^{\alpha} + K_P \cdot f_B^{\alpha} + 2L \tag{4.2}$$

The bottom of Figure 4.2 shows a scenario where an estimation error Δf has occurred. Because the frequency is lower than expected, some tasks (W) are not executed and are stored in the queue at time step $k+1$. To compare both scenarios of Figure 4.2, we must have the same starting and ending conditions. This implies having all jobs executed by the time step $k+2$. To achieve that, queuing tasks need to be executed in time frame B. Thus, a frequency higher than the ideal one has to be used. The power consumption in this scenario P_{err} due to the execution of tasks Ta' and Tb' respectively in time frame A and B is given by the following equation:

$$P_{err} = K_P \cdot [(f_A - \Delta f)^{\alpha} + (f_B + \Delta f)^{\alpha}] + 2L \tag{4.3}$$

We define the energy loss E due to an estimation error as:

$$E = P_{err} - P_{ideal} \tag{4.4}$$

by substituting Equations 4.2 and 4.3 into Equation 4.4, we obtain:

$$E = K_P \cdot [(f_A - \Delta f)^{\alpha} + (f_B + \Delta f)^{\alpha} - f_A^{\alpha} - f_B^{\alpha}] \tag{4.5}$$

if α is greater than 1, the energy loss E in Equation 4.5 is greater than zero.

As a concluding remark, if there is a frequency estimation error, some of the tasks will stay in the queue and will be executed in the next time slot. This will cause the processor to run at a higher frequency in the next slot since now there are some extra jobs pending due to the estimation error. If the relation between the frequency and the power consumption were a linear function, there would be no waste of energy. Since the function is convex, Equation 4.5 holds and so an estimation error is crucial from a power-performance efficiency perspective.

4.1.2 Energy bounds

In the previous subsection we analyzed the effect of a workload estimation error on the power consumption needed to execute a specific workload. In quantifying the energy efficiency quantification, we assumed that tasks Ta and Tb are executed respectively in time windows A and B. This constraint makes the task delay equal to zero. Nevertheless it is not the minimum value of power consumption that we can get to execute all tasks $Ta+Tb$ during time windows A and B.

The empirical law that represents the power consumption as a function of the frequency setting expressed by Equation 4.1 is a convex function (Boyd

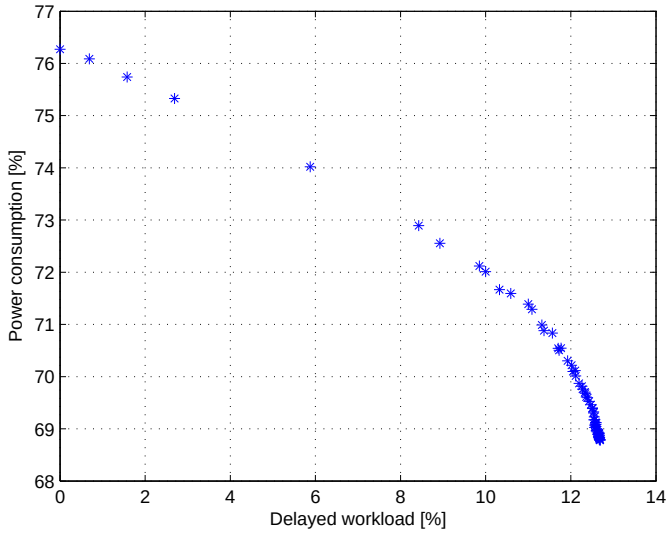

Figure 4.3: Example of normalized power consumption versus delayed workload for different optimization criteria ranging from power-oriented to performance-oriented optimizations.

et al. [11]). By applying basic properties of convex functions, we obtain the following:

$$\mathbf{p}_\tau + \mathbf{p}_{\tau+\epsilon} \geq \mathbf{2} \cdot \mathbf{p}_{(\tau+\epsilon)/\mathbf{2}} \qquad \forall \epsilon \in [\mathbf{0}, \mathbf{1}) \tag{4.6}$$

where $\mathbf{p}_\tau$ is the power consumption at time τ. Since frequency setting and executed workload are positively correlated (see Choi et al. [18]), then energy savings demand a uniformly-distributed workload. The problem is that workloads are usually not uniformly distributed during the run-time execution of the policy and scheduling task uniformly would increase latency.

Inequality 4.6 expresses this issue as follows:

$$\mathbf{p_{pow}} \leq \mathbf{p} \leq \mathbf{p_{perf}} \tag{4.7}$$

by indicating that power consumption $\mathbf{p}$ is bounded between two values. On the one hand, the lower bound ($\mathbf{p_{pow}}$) is the power value consumed when the workload is uniformly distributed. In this case, we optimize the execution for power minimization by allowing a non-zero task execution delay, but at the same time we require that the complete workload has to be executed.

On the other hand, the upper bound ($\mathbf{p_{perf}}$) is the power consumed when all tasks are executed at the same time they arrive. In this case, we optimize the execution for performance and the resulting task execution delay is zero. Clearly, the gap between these two numbers is highly dependent on the workload properties. Figure 4.3 shows an example of the resulting power consumption versus delayed workload for different optimization criteria ranging from power-oriented to performance-oriented optimizations.

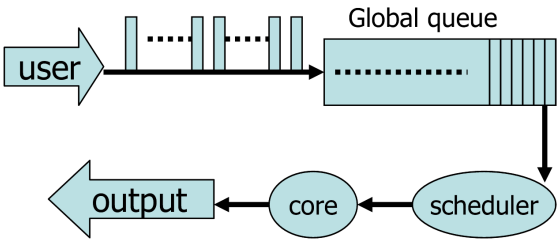

Figure 4.4: Overview of the system architecture.

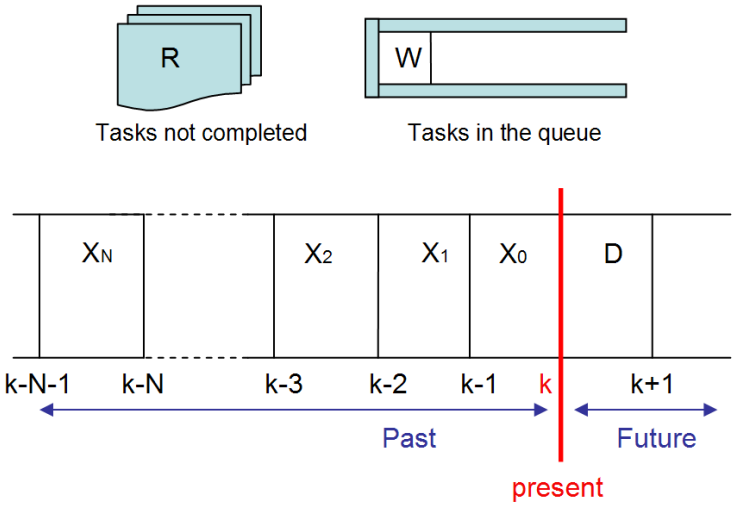

Figure 4.5: Snapshot of the task arrival process at time k.

4.2 Workload Model

4.2.1 System architecture

From an operating system perspective, the interaction of the MPSoC with a user generates a task arrival process. To simulate this interaction we use benchmarks. The benchmark is a set of programs, or other operations, to assess the relative performance of the MPSoC. Figure 4.4 shows an overview of the MPSoC system architecture abstraction.

Incoming tasks are first stored in a global queue. Each task has a specific execution time. In each scheduling cycle, the scheduler fetches the next available packet from the global queue and then dispatches the packet into a specific core for task processing. The result is the desired output.

4.2.2 Task arrival process

The task arrival is a stochastic process, as described in Figure 4.5. The picture shows the status of the system at a certain point k. The system is discrete-time

and it is sampled every T_p, where T_p is the period of time that passes between the sample at time k and and the sample at time $k+1$. The frequency of the sampling points is the frequency with whom thermal policies are applied. The term R stands for the amount of work to be executed by tasks that are still in the processor and need to be completed. W is the amount of work to be executed by tasks arrived waiting to be processed. X_i is the amount of work executed by task arrived i time steps before the current step K. N is the length of the task arrival history. D the amount of work to be executed by task that will arrive in the next time step between the actual one k and the next one $k+1$.

In the block diagram of Figure 4.5, the overall amount of work S to be executed between time K and time $K+1$ equals:

$$S = R + W + D \tag{4.8}$$

Equation 4.8 expresses that the system has to execute all uncompleted tasks (R) plus the ones not executed in previous time steps that are queuing (W) plus the one that will arrive in the next time step (D). If the overall amount of tasks S are executed, performance requirements are satisfied, otherwise the system will experience a performance loss.

4.2.3 Workload model

For each p clock islands(cores), the workload is defined as the minimum value of the clock frequency that the functional unit should have to execute the required tasks within the specified system constraints.

The workload requirement at time τ is defined as a vector $\mathbf{w}_\tau \, \epsilon \, \Re^{\mathbf{p}}$, where $(\mathbf{w}_\tau)_\mathbf{i}$ is the workload requirement value for input i at time τ. $(\mathbf{w}_\tau)_\mathbf{i}$ is the frequency that cores associated with input i from time τ to time $\tau+1$ should have in order to satisfy the desired performance requirement coming from the scheduler. The workload requirement is related to the overall amount of work to be executed S by following equation:

$$S = T_p \cdot \sum_{i=1}^{p} (\mathbf{w}_\tau)_\mathbf{i} \tag{4.9}$$

where T_p is the period between time k and k+1 and p is the number of cores. Equation 4.9 expresses the fact that the sum of all the workloads integrated over the time period T_p should be equal to S, the overall amount of work to be executed.

Our model is assumed to be continuous and ranging from $\mathbf{f}_{\min}$ to a max value $\mathbf{f}_{\max}$, the maximum frequency at which the cores can process data, namely

$$\mathbf{f}_{\min} \preceq \mathbf{w}_\tau \preceq \mathbf{f}_{\max} \quad \forall \, \tau \tag{4.10}$$

When $(\mathbf{w}_\tau)_\mathbf{i} > (\mathbf{f}_\tau)_\mathbf{i}$, the workload cannot be processed and so it needs to be stored (W) and rescheduled in the following clock cycles. The way we

measure the performance of the system in achieving the requested workload requirements at time τ is given by the vector $\mathbf{u}_\tau \epsilon \Re^{\mathbf{p}}$.

$$\mathbf{u}_\tau = \mathbf{w}_\tau - \mathbf{f}_\tau \tag{4.11}$$

We call $\mathbf{u}_\tau$ the undone workload at time τ and it expresses the difference at time τ between the requested workload and the workload that is actually executed by the MPSoC.

4.2.4 Frequency and power model

We model the MPSoCs as a synchronous with p clocks that are viewed as the inputs to the system: vector $\mathbf{f}_\tau \epsilon \Re^{\mathbf{p}}$ represents the value of the clock frequencies at time τ. The frequency value of input i at time τ is $(\mathbf{f}_\tau)_\mathbf{i}$. Clock frequencies are continuous and range from zero to a max frequency value $f_{\max}$. Previous statement is expressed by Inequality 4.12.

$$0 \preceq \mathbf{f}_\tau \preceq \mathbf{f}_{\max} \quad \forall\, \tau \tag{4.12}$$

where the symbol $\preceq$ means element-wise comparison, $f_{\max} \cdot \mathbf{1} = \mathbf{f}_{\max}$ and $\mathbf{1}$ is a vector of all ones of size p. The frequency vector represents our optimization variable.

At time τ, the relation between the normalized value of power dissipation $\mathbf{p}_\tau \epsilon \Re^{\mathbf{p}}$ and the normalized frequency of operation $\mathbf{f}_\tau$ is expressed by Equation 4.13.

$$\mu \mathbf{f}_\tau^\alpha = \mathbf{p}_\tau \quad \forall\, \tau \tag{4.13}$$

where μ is a technology-dependent coefficient. The constant α depends as on the technology as well and usually it takes a value between 1 and 2. If $\alpha = 1$, we have a linear dependence (i.e., frequency scaling) while if $1 < \alpha \leq 2$ we obtain a quadratic or sub-quadratic dependence (i.e., DVFS) [62].

We calculate the leakage power of processing units inside the MPSoC as a function of their area and actual run-time temperature. We use a base leakage power density of $0.25 W mm^2$ at $383°K$ for $90nm$ technology according to experimental results from Bose [9]. Thus, the leakage power at a temperature $T°K$ is given by:

$$P(T) = P_o \cdot e^{\beta(T-383)} \tag{4.14}$$

where P_o is the leakage power at $383°K$, and β is a technology dependent coefficient. We set $\beta = 0.017$ according to experimental results from Sabry et al. [77].

4.3 Workload Prediction

4.3.1 Workload arrival process

The workload arrival process can be modelled as a stochastic process. Without loss of generality we consider an MPSoC with 8 cores. Graph 4.6 shows the

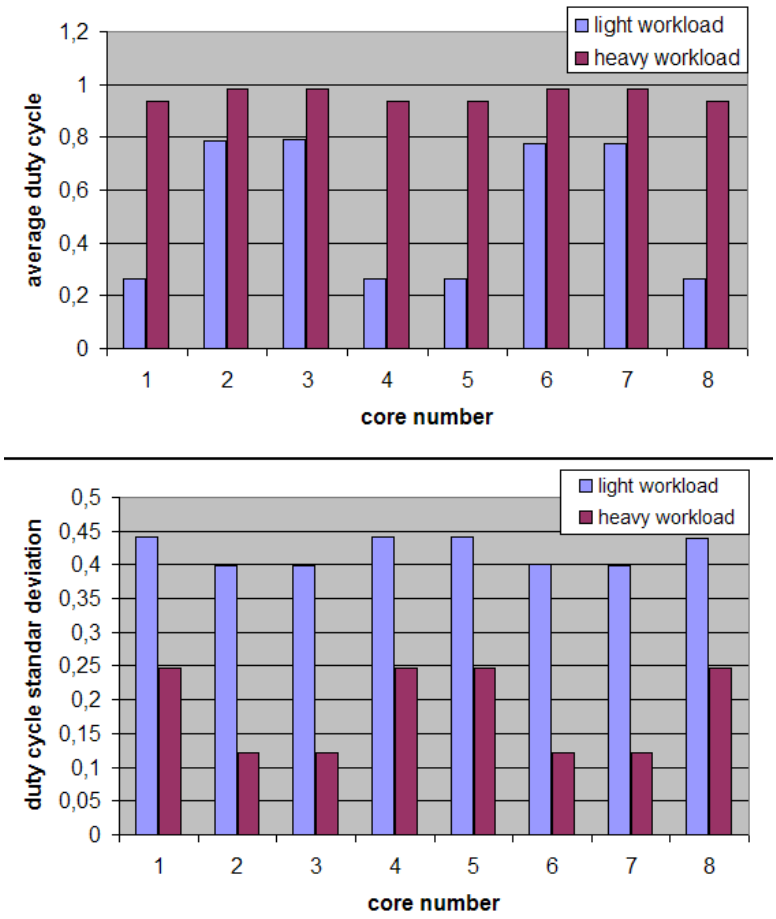

Figure 4.6: Duty factor mean and standard deviation of cores during system operation.

duty cycle mean and standard deviation of cores in the case of a light and heavy scenarios using the experimental setting described in Section 5.2.

In both cases the standard deviation of the duty cycle is not zero. The reason is because the overall task execution process can be seen as a stochastic **G**/**G**/**8** process with an infinite buffer length. The first two **G** mean that both the task arrival and the task execution time can have any distribution and **8** is the number of cores processing the tasks. From queuing theory we know that in these systems core utilization have a standard deviation that is higher when the core utilization is lower. This is because the probability that the queue is empty is higher and if the queue is empty some processors may stay idle while others are busy.

Figure 4.6 shows that, in case of a light workload, the standard deviation of core duty cycle is higher than 40% while in the other case is lower than 25%. Moreover the standard deviation assumes different values on different cores. Looking at the mean value of core duty cycle, the assumption of having a core utilization that is equal to one and the same for every core can be seen

as a 10% approximation in heavy workload scenarios but does not hold in light ones.

In next subsections, we propose two estimators to predict future workload requirements in time varying scheduling scenarios employing dynamic scheduling techniques.

4.3.2 Prediction accuracy

It is possible to take into account of the accuracy of the prediction in the problem formulation of a thermal management policy by embedding a weighting vector γ_t. It is defined according to Equation 4.15:

$$\gamma_{\mathbf{t}} = \beta_{\mathbf{t}} - \|\mathbf{w_t} - \hat{\mathbf{w}}_{\mathbf{t}}\| \quad \forall \quad \mathbf{N} - \mathbf{L} \leq \mathbf{t} \leq \mathbf{N} \tag{4.15}$$

where $\hat{\mathbf{w}}_{\mathbf{t}} \epsilon \Re^{\mathbf{p}}$ is the workload predicted at time t by the workload predictor, $\mathbf{w_t} \epsilon \Re^{\mathbf{p}}$ is the actual value of it and p is the number of different workloads requested at a specific time t. They correspond also to the number of processing cores of the MPSoC. The absolute value of the difference between the two represents the prediction error.

$\beta_{\mathbf{t}} \epsilon \Re^{\mathbf{p}}$ is a vector that adds a penalty for the workload that has been predicted, but not executed yet, in different and future time frames. This penalty function $\beta_{\mathbf{t}}$ can be chosen to be linear, quadratic, exponential or in any other way, according to the impact that a delayed execution of tasks has on performance. Indeed the more reliable the prediction is, the smaller the prediction error is and so the bigger γ_t is. This means that, since in our formulation the prediction is reliable, importance is given to the cost function corresponding to that future time frame.

4.3.3 Maximum energy concentration based estimator

Method description

This system is based on both a queue and a mean value prediction based on task arrival history. The derivation presented here is referred to Figure 4.1. The proposed frequency f_p to execute all the tasks by the time $K+1$ will be:

$$f_p = \frac{R + W + D}{p \cdot T_p} \tag{4.16}$$

where T_p is the period between time k and $k+1$ and p is the number of cores.

The problem is the estimation of the parameter D in the case of non-stationary task arrival processes. This means that statistical properties of the task arrival process are changing over time. The task arrival has been modelled this way since there is no guarantee that user requirements will not change over time.

To estimate statistical parameters (mean value) of this process, we use a *short-time discrete time fourier transform*(DSTFT) using a Kaiser window

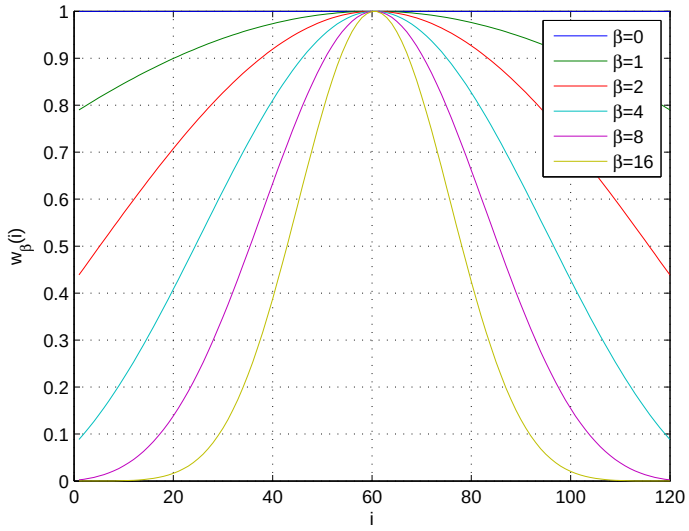

Figure 4.7: Kaiser window function for N=120 and different values of β.

(see Oppenheim and Schafer [63]). The DSTFT is a Fourier-related transform used to determine the sinusoidal frequency and phase content of local sections of a signal as it changes over time. It is a Fourier transform of the original signal multiplied by a windowing function. Among all the possible windowing functions, we decided to use a Kaiser window, because it is a relatively simple approximation of the prolate spheroidal function that satisfies the maximum energy concentration theorem [66].

Thus, an estimate of the parameter D is given by following equation:

$$D = \frac{\sum_{i=0}^{N} w_\beta(i) \cdot X_i}{\sum_{i=0}^{N} w_\beta(i)} \tag{4.17}$$

where N is the number of past time slot considered and $w_\beta()$ a Kaiser window of parameter β and length N. Figure 4.7 shows the shape of this type of windowing function for different values of N and β. As it can be noted, for $\beta = 0$, than D is the mean of the last N X_i.

The parameters N and β are left to the designer. The optimum size of this parameter depend on the way the task arrival process changes its mean and standard deviation over time. Moreover it depends also on the memory that is allocated to keep track of the history of the task arrival process. For slowly varying processes to have a large window size N is good since this way the prediction error is small. For fast varying processes, to have a small N is better.

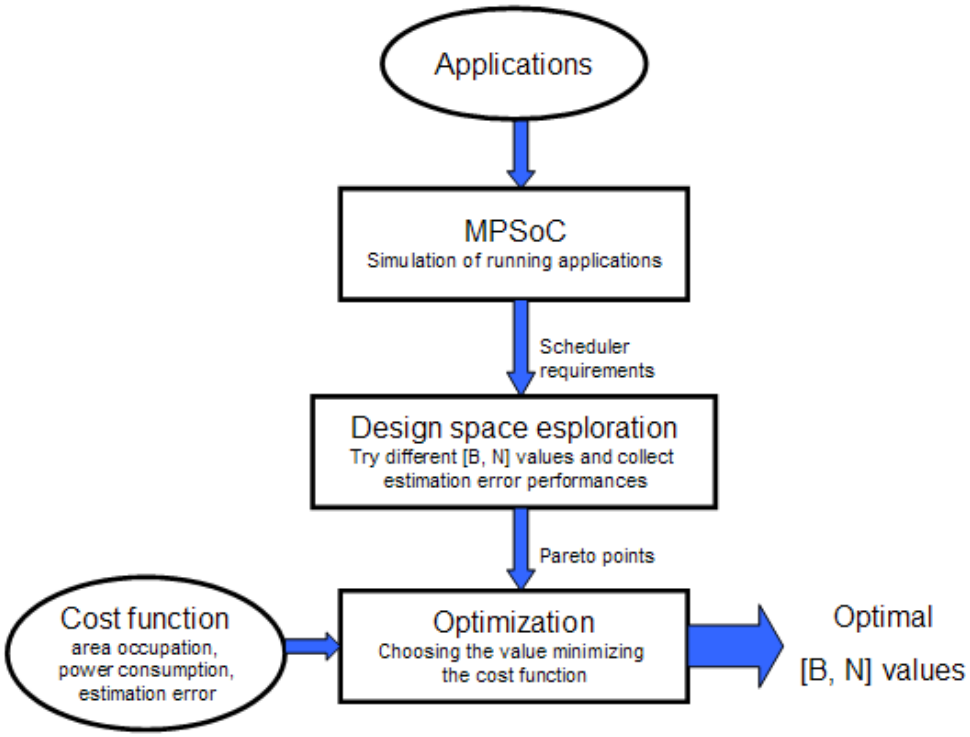

Figure 4.8: Block diagram description of the method used to derive β and N parameters from real applications or benchmarks.

Mathematical analysis

There are two fundamental parameters that are left to the designer: window width (history) N and the parameter β that determines the shape of the Kaiser window. To derive these parameters from real applications or benchmarks, we propose the method described in the block diagram of Figure 4.8.

Given some applications, they are run on the MPSoC. Scheduler requirements for every time window are collected and stored. After that, a design space exploration is made and the performance of different β and N parameters is tested.

After that, an optimization is made basing on both Pareto points obtained at the end of the design space exploration and a pre-defined cost function. This cost function takes into account how power consumption, area occupation and the estimation error is important in the optimization process. Finally the result of this is the pair β and N that will best fit specified needs.

System design

For the experiment, we consider the 8-core Niagara architecture described in Section 5.2. To derive N and β parameters of the Kaiser window, a design

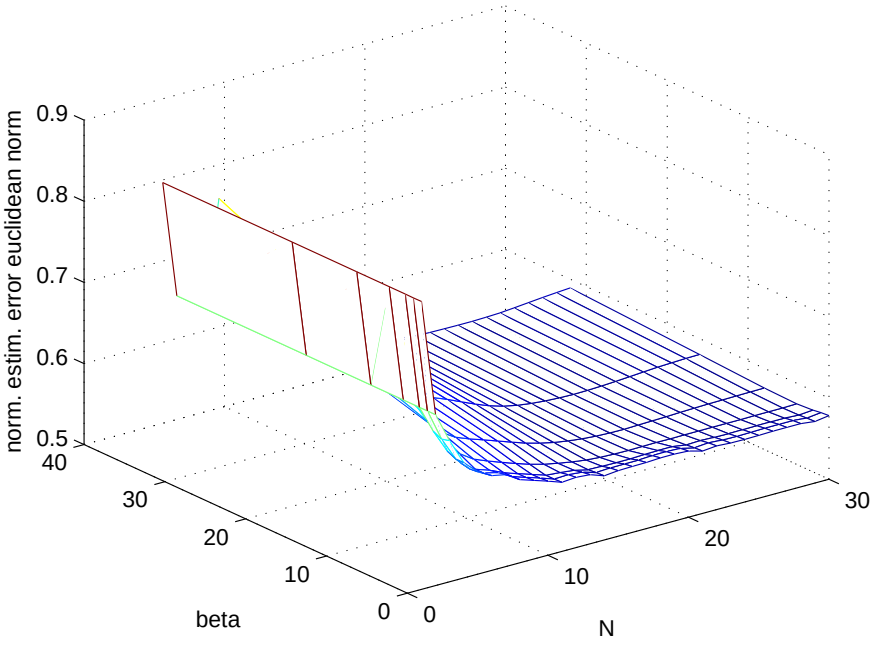

Figure 4.9: Euclidean norm of the estimation error normalized to the average maximum number of tasks executable by the MPSoC.

space exploration has been made. All possible values of β (between 0 and 32) and N (between 1 and 30) have been tested. The output of this study is reported in figure 4.9. The graph plots the Euclidean norm of the estimation error normalized to the average maximum number of tasks executable by the MPSoC. In this case, since there's only one point that minimizes the estimator error, the optimization is an easy task. Optimal points are: $\beta = 8$ and $N = 20$.

4.3.4 Polynomial least squares workload prediction

Method description

This technique makes the prediction by using a linear model to perform a best d^{th} order polynomial fit, because the prediction length L for this application is short and ranges usually from 1 to 9 samples.

The polynomial fit is performed by minimizing the error within the observed window of temperatures, by using the following function:

$$\|\mathbf{w} - \check{\mathbf{A}}\mathbf{x}\|_{\mathbf{2}}^{\mathbf{2}} \tag{4.18}$$

where $\mathbf{w_t}$ contains the frequency requirements $\forall t = 1...N$, N is the length of the observation window of historical data. Vector $\mathbf{x_t} \; \epsilon \; \Re^{\mathbf{d+1}}$ and matrix $\check{\mathbf{A}} \; \epsilon \; \Re^{\mathbf{d+1 \times (N+L)}}$ are used in the polynomial interpolation process.

$$\check{A} = \begin{pmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ 1 & 1 & 1 & 1 & \dots & 1 \\ 1 & 2 & 4 & 9 & \dots & 2^{d} \\ . & . & . & . & \dots & . \\ . & . & . & . & \dots & . \\ 1 & N+L & (N+L)^2 & (N+L)^3 & \dots & (N+L)^{d} \end{pmatrix}$$

Figure 4.10: Structure of matrix $\check{\mathbf{A}}$.

Equation 4.18 can be solved as a least squares minimization problem to derive vector $\mathbf{x}$. The prediction on the future workload requirement is performed by assuming that the linear model just derived will hold for the next L data samples.

Assuming this assumption hold, the future workload requirement is given by following equation:

$$\mathbf{w_t} = \check{\mathbf{A}}\mathbf{x_t}, \quad \forall \quad \mathbf{N} \leq \mathbf{t} \leq \mathbf{N} + \mathbf{L} \tag{4.19}$$

where $\mathbf{w_t} \, \epsilon \, \Re^{\mathbf{p}}$ for $t > N$ is the predicted workload requirement at time t. We tested the predictor on the benchmarks described in the experimental setup section of this thesis and we achieved good accuracies for short-term forecasts (L ranging from 1 to 9).

Design methodology

The structure of matrix $\check{\mathbf{A}}$ is shown in Figure 4.10. As it can be noted, the first row of matrix $\check{\mathbf{A}}$ considers only the first component of vector $\mathbf{x}$ since the first entry of the row is the only nonzero element. The second row considers all the history contained in vector $\mathbf{x}$, all with the same weight. The third row is a weight function that considers elements of vector $\mathbf{x}$ according to a quadratic weight. All rows of matrix $\check{\mathbf{A}}$ follow the same trend including the last row that weight entries of vector $\mathbf{x}$ with a d^{th} polynomial function.

Values of N (the length of the observation window) and d (the order of the polynomial fit) that provide the best prediction depend on the workload requirement (task arrival process) statistical properties. These kind of processes are usually non-stationary and depend on the interaction between the user and the MPSoC itself. For the aforementioned reasons, to chose these parameters we suggest to use empirical studies.

4.4 Summary

This chapter has introduced the concepts to model the workload of an MPSoC system. Moreover some considerations have been made about the system energy models used in this thesis.

The first section has introduced the concept that minimizing power saving, improving performance and increase chip reliability are three important tradeoffs facing MPSoC designers. A theoretical quantification of the energy efficiency is also provided.

The second section has described the power and the workload models used in this thesis. A description of the way the task arrival process is abstracted has also been presented. Finally the relation between the working frequencies of the cores composing the MPSoC and its power dissipation has been described.

The third section has provided a theoretical and experimental analysis about the effects that workload prediction has on the performance of any thermal management technique. Two workload estimation techniques are also presented.

Policies for Thermal Control with Air Cooling

5

This chapter we analyze and explore the use of four families of control techniques for thermal management of multi-processors system on chip (MPSoC). In particular, we aim at achieving an online smooth thermal control action that minimizes the performance loss as well as the computational and hardware overhead of embedding a thermal management system inside the multi processor system on chip. The optimization problem considers the thermal profile of the system, its evolution over time and current time-varying workload requirements. This problem is formulated as a finite-horizon optimal control problem. Thus, we implement the policies on a hardware simulation platform and we collect data of the system. A detailed comparison and analysis is also reported.

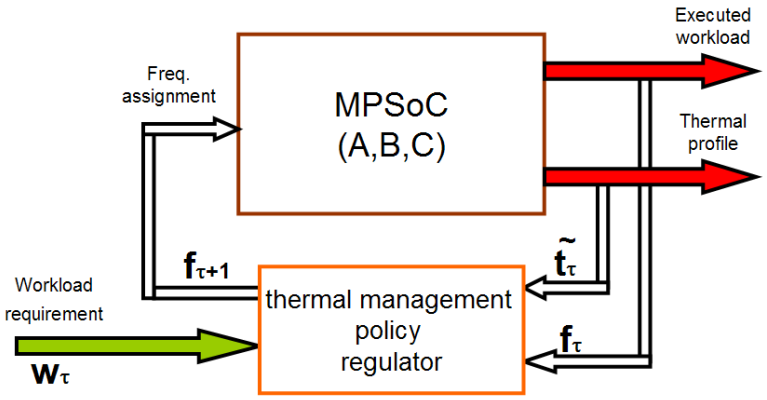

Figure 5.1: Diagram of a generic DVFS-based thermal management system

5.1 Introduction

The structure of state-of-the-art DVFS-based thermal management systems is reported in the diagram of Figure 5.1.

The thermal management policy regulator monitors the *multi processors system on chip* (MPSoC). Without loss of generality, we assume this MPSOC to be partitioned into p islands (or subsystems), each with independent frequency and voltage settings. For our purposes, we consider frequencies as inputs to the system, since we are abstracting away the computation. Vector $\mathbf{f}_\tau \epsilon \Re^{\mathbf{p}}$ represents the value of the clock frequencies at time τ. The frequency value of input i at time τ is denoted by $(\mathbf{f}_\tau)_\mathbf{i}$. Input i ranges from 1 to p.

The regulator sets working frequencies $\mathbf{f}_{\tau+\mathbf{1}}$ according to a specific policy. The frequency setting the regulator does at time τ is performed by taking into account the current frequency setting $\mathbf{f}_\tau$, temperature measurements $\tilde{\mathbf{t}}_\tau$ coming from on-die thermal sensors and a workload requirement coming from the scheduler $\mathbf{w}_\tau \epsilon \Re^{\mathbf{p}}$. For each functional unit $i = 1..p$, the workload is defined as the minimum value of the clock frequency that the functional unit should have in order to execute the required tasks within the specified system constraints. The regulator provides a frequency assignment that minimizes the difference between the required and the achieved workload.

The author of this thesis proposed previously various policies for thermal management [96], [95], [99], [100]. These policies were developed and analyzed independently. In this thesis, we provide a comprehensive comparison of four families of thermal management methods for multi processors system on chip. Thus, this chapter provides the analytical and experimental basis for selecting among four similar, but different, control methods.

These families are closely related and some are based on *model predictive control* (MPC) [2], which has received much attention lately. These policies aim at achieving an online smooth thermal control action that minimizes the performance loss as well as the computational and hardware overhead of embedding a thermal management system inside the MPSoC. Control models and

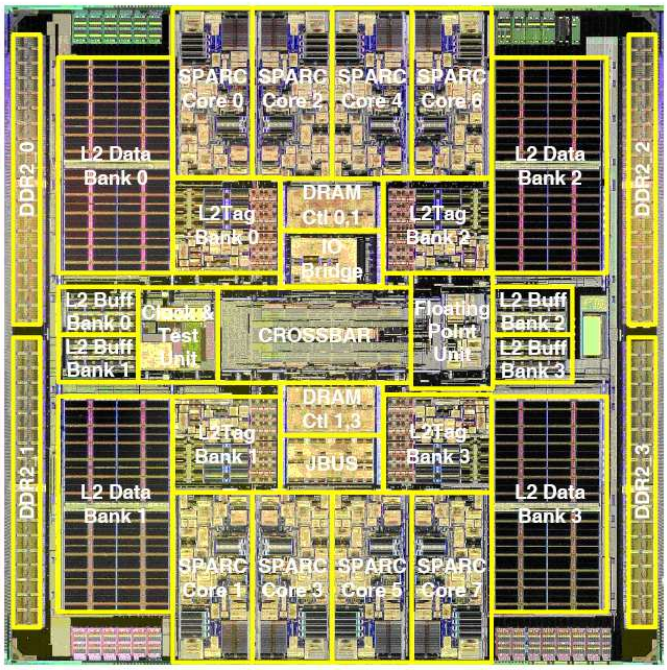

Figure 5.2: UltraSPARK T1 processor, die photograph by courtesy of SUN [49]

policies differ according to both the way details are included in the problem formulation and the way the solution is computed.

5.2 2D-MPSoC case study

5.2.1 The Niagara processor

The 64b Niagara SPARC processor (UltraSPARC T1) [49] is designed for power-efficient high-throughput commercial server applications where power, cooling, and space are major concerns. The *chip-multithreaded* (CMT) architecture achieves high throughput while optimizing performance/watt. Concurrent execution of 32 threads is implemented through 8 symmetrical 4-way multithreaded cores, supported by a high-bandwidth low-latency cache/memory system shown in Figure 5.2.

Features are: eight 64b Multithreaded SPARC Cores, shared 3MB L2 Cache, 16KB ICache per Core, 8KB DCache per Core, four 144b DDR-2 DRAM Interfaces (400 MTs), 3.2GB/s JBUS I/O.

About the technology: 90nm CMOS Process, 9LM Copper Interconnect, Power: 63 Watts @ 1.2GHz, Die Size: $378mm^2$, 279M Transistors, Package: Flip-chip ceramic LGA (1933 pins)

In our thermal model, the parameters are provided in Table 5.1. This table contains the thermal conductance and capacitance values of different materials used in modeling the MPSoC.

Table 5.1: Thermal and Floorplan parameters deployed in the model

Parameter	Value
Silicon conductivity	$130 W/(m \cdot K)$
Silicon capacitance	$1635660 J/(m^3 \cdot K)$
Wiring layer conductivity	$2.25 W/(m \cdot K)$
Wiring layer capacitance	$2174502 J/(m^3 \cdot K)$

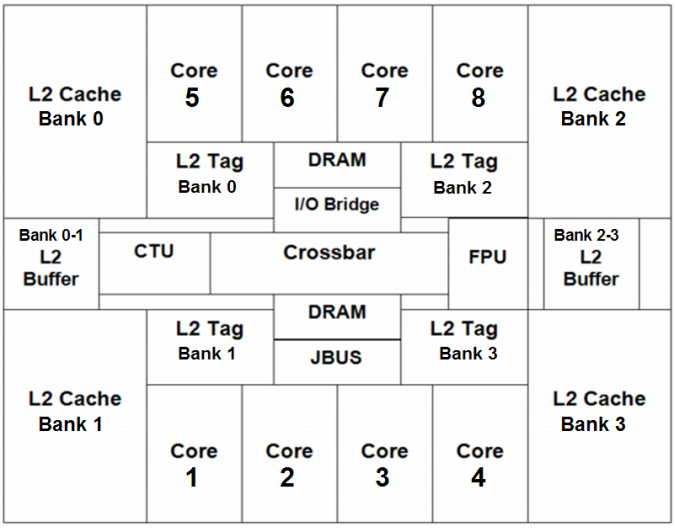

Figure 5.3: Floorplan of the Niagara-1 multicore case study

5.2.2 Layout

The MPSoC architecture we are considering resembles the 8-core Niagara-1 (UltraSparc T1) architecture from Sun Microsystems [43]. To model the Niagara floorplan, the chip has been divided into functional regions as depicted by yellow rectangles in the picture of 5.2. The resulting floorplan model is presented in Figure 5.3. The floorplan has been modelled using blocks of 3mm side each, and values of technological parameters and coefficients have been derived from [43].

5.2.3 Frequency Setting and DVFS

This architecture has a maximum operating frequency of 1.2 GHz and the maximum power consumption of each processor core at this frequency is 4 W [43]. To implement the voltage and frequency scaling techniques, we use frequencies ranging from 0 to 1.2GHz.

In this range, only specific values of frequencies are allowed, thus the values

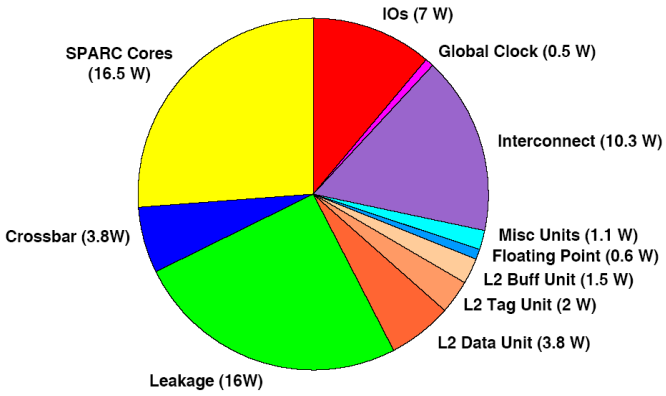

Figure 5.4: Niagara, chip power consumption.

for the frequencies different from 0 are expressed by Equation 5.1.

$$f = \frac{6 \cdot 10^9}{D_f} \quad \forall\ D_f,\ subj.\ to: \quad 5 \leq D_f \leq 35 \tag{5.1}$$

where D_f is the division factor needed by the clock tree generator. To simulate the system we used different benchmarks, ranging from web-accessing to playing multimedia [22], [23].

5.2.4 Power Consumption

To perform the thermal evaluation and calculate the temperature distribution among the chip, we need to have a power consumption trace for each yellow box on the die. If we know when each unit is active or idle, we can estimate the instantaneous power consumption using the average power values. For SPARC cores, the peak power consumption is similar to the average power values [49]. The power efficiency of this chip has been tested with the *Java Business Benchmark* (SPECJBB) [40] and the resulting power consumption of various components has been reported [49] in the pie-chart of Figure 5.4.

Figure 5.5 shows the ratio of the energy dissipated by each unit type in the chip with reference to the overall power consumption.

5.2.5 Benchmarks and Workload Statistics

We modelled the different workloads of a mix of different benchmarks, ranging from web-accessing to playing multimedia [22], [23]. Each job created by the scheduler defines and tests a different workload by varying the load of a set of large multiplication matrices and memory accesses intervals. Recorded execution characteristics are shown in Table 5.2. Using cpustat, we also recorded the cache misses and *floating point* (FP) instructions per $100K$ instructions to accurately model them in our MPSoC simulation framework.

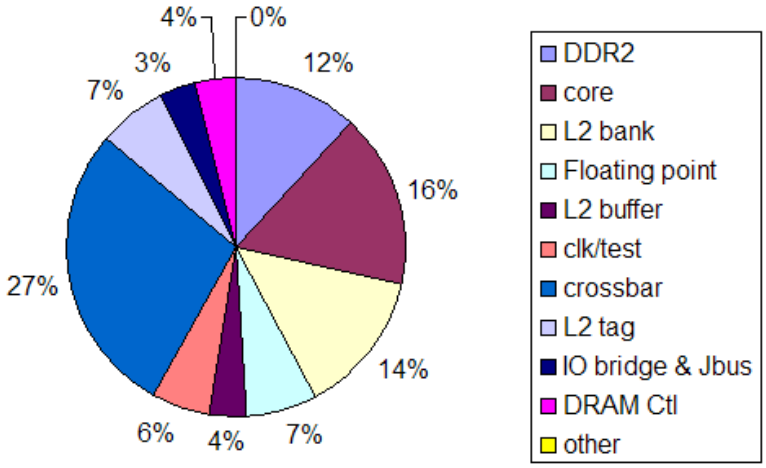

Figure 5.5: Niagara, chip power consumption by unit type.

	Benchmark	Average CPU Load	L2 I-Miss	L2 D-Miss	FP unit
1	Web-med	53.12%	12.9	167.7	31.2
2	Web high	92.87%	67.6	288.7	31.2
3	Database	17.75%	6.5	102.3	5.9
4	Web+DB	75.12%	21.5	115.3	24.1
5	gcc	15.25%	31.7	96.2	18.1
6	gzip	9	2	57	0.2
7	Mplayer	6.5%	9.6	136	1
8	Mplayer+Web	26.62%	9.1	66.8	29.9

Table 5.2: Comparative table of workload characteristics

The average CPU load is 62%, and ranges during the runtime execution from 10% to 100%. Jobs have an average duration of $2ms$, but oscillates from $0.1ms$ to $10ms$. The experiments are conducted using a large trace with around 60000 tasks, modeling several hundreds of seconds of actual system execution. We assume as initial condition, the system to be at room temperature set equal to $300°K$.

5.3 Policy Classification

In this paper we are considering and comparing four polices that are in our opinion representative. The thermal policy techniques we are considering are: the 'linear quadratic regulator' [96] (unconstrained MPC with horizon equal to infinity), the 'explicit/implicit model predictive control'-based approach [95] (traditional MPC), the 'approximated explicit model predictive control' policy [99] (approximated MPC) and finally the 'convex optimization'-based

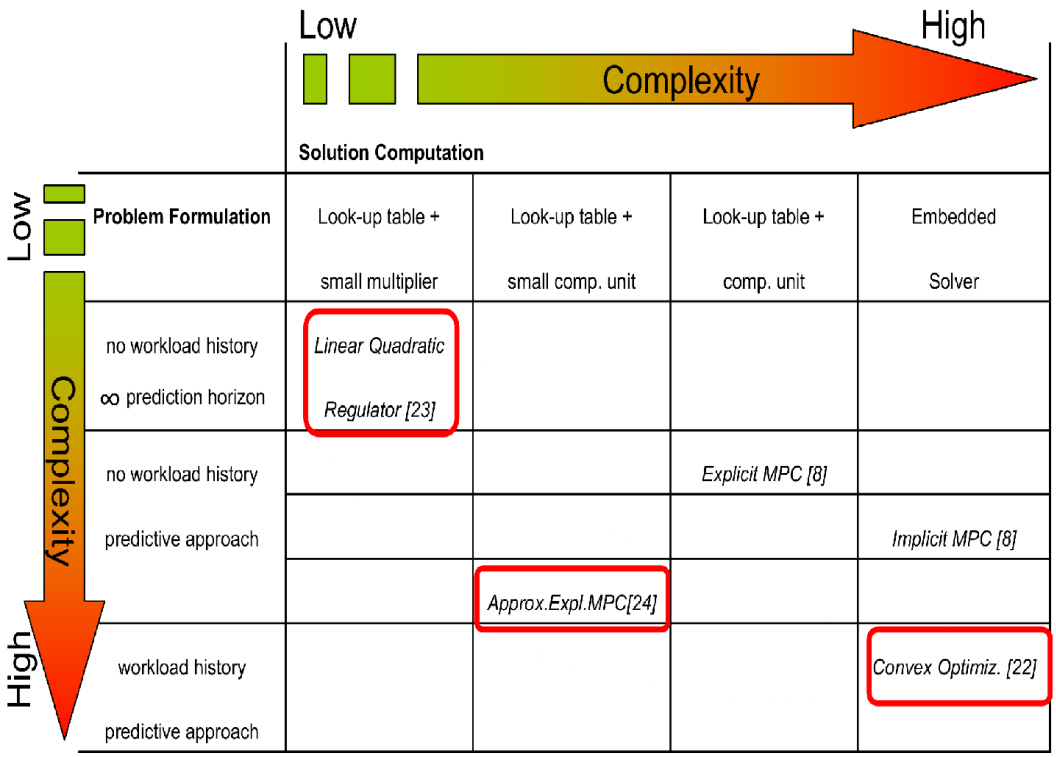

Figure 5.6: Classification of compared thermal management policies

approach [100] (joint workload & thermal profile prediction). This last technique is solved with a convex solver, however it is an MPC as well with a linear objective function.

We classify policies according to two main metrics. The first one is the complexity of the problem formulation and the second is the solution computational requirements. Figure 5.6 locates the compared policies graphically.

The first row identifies thermal policies that do not take into account the workload history. The entire formulation and solution of the *Linear Quadratic Regulator*(LQR) is based on the idea of prediction. However, the prediction horizon on the temperature profile of the system is fixed and equals to infinity. These policies only ensure that both power consumption and temperature spatial gradients are minimized. Their objective function targets:

- minimization of the difference between the target power consumption (required to get the work done) and the real one
- temperature spatial gradients minimization

The *linear quadratic regulator*(LQR) does not allow one to set any threshold constraint on the MPSoC maximum temperature. However a threshold-based mechanism is used to partially solve the problem.

The second row of Figure 5.6 identifies techniques that try to predict the future behavior of the MPSoC thermal profile. To this family of polices belong the approximated, the explicit and the implicit MPC. Their objective function targets:

- undone work minimization
- temperature gradient minimization
- power consumption minimization

Constraints are:

- frequency range
- power-frequency relation
- thermal profile prediction
- maximum temperature

The last row identifies techniques performing a prediction on the MPSoC thermal profile by additionally exploiting workload history information. An example is provided by convex optimization. The objective function targets:

- undone work minimization
- temperature gradient minimization
- power consumption minimization

Additional constraints with respect to the first and second row are:

- workload prediction
- workload prediction reliability

Each column of the table identifies a different level of computational requirement needed to on-line compute the solution to the problem formulation. To the first column belong policies that require only a look-up table and a small multiplier to be implemented. The second column identifies techniques needing a look-up table and a small computational unit to be implemented. Here the level of complexity is higher and higher capabilities are required. To the third column a look-up table and an average-to-large computational unit is required. To the last column belong algorithms requiring an online solver to be implemented.

Interesting trade-off policies are highlighted by the red squares and by the dotted lines. They correspond to techniques taking into account the highest number of MPSoC properties for a given computational requirement. They indeed perform the best in making the problem formulation easy to solve while exploiting the largest quantity of information in the determination of the MPSoC frequency assignment.

5.4 Comparison of Receding Horizon Algorithms

Common objectives fulfilled by all these class of polices are the following. First they ensure that the maximum MPSoC temperature never exceeds a pre-defined threshold. Second they avoid abrupt frequency and temperature variations both over time and over space. Finally they minimize the work that is requested from the scheduler and not executed. The control problem to be solved to fulfill previous requirements can be formalized in the following way:

$$J = \sum_{\tau=1}^{h} \left(\|\mathbf{Q}\mathbf{x}_\tau\|_{\mathbf{g}} + \|\mathbf{R}\mathbf{p}_\tau\|_{\mathbf{j}} + \|\mathbf{T}\mathbf{u}_\tau\|_{\mathbf{b}} \right) \tag{5.2}$$

$$min \;\; J \tag{5.3}$$

$$subject\ to: \quad 0 \preceq \mathbf{f}_\tau \preceq \mathbf{f}_{\max} \;\; \forall\, \tau \tag{5.4}$$

$$\mathbf{x}_{\tau+\mathbf{1}} = \tilde{\mathbf{A}}\mathbf{x}_\tau + \tilde{\mathbf{B}}\mathbf{p}_\tau \;\; \forall\, \tau \tag{5.5}$$

$$\tilde{\mathbf{C}}\mathbf{x}_{\tau+\mathbf{1}} \preceq \mathbf{t}_{\max} \;\; \forall\, \tau \tag{5.6}$$

$$\mathbf{u}_\tau \succeq \mathbf{0} \;\; \forall\, \tau \tag{5.7}$$

$$\mathbf{u}_\tau = \mathbf{w}_\tau - \mathbf{f}_\tau \;\; \forall\, \tau \tag{5.8}$$

$$\mathbf{p}_\tau \succeq \mu \mathbf{f}_\tau^\alpha \;\; \forall\, \tau \tag{5.9}$$

Function J is expressed by three sums where the summation index τ ranges from 1 to h. During these h future steps the system tries to minimize the cost function J and computes the frequency assignment for these steps.

The first term $\|\mathbf{Q}\mathbf{x}_\tau\|_{\mathbf{g}}$ is the g norm of the state vector x weighted by matrix $\mathbf{Q}$. Matrix $\mathbf{Q}$ is related to hotspot minimization and thermal balancing. The second term $\|\mathbf{R}\mathbf{p}_\tau\|_{\mathbf{j}}$ is the j norm of the input power vector p weighted by matrix $\mathbf{R}$. Matrix $\mathbf{R}$ quantifies the importance that power minimization has in the optimization process. The third term $\|\mathbf{T}\mathbf{u}_\tau\|_{\mathbf{b}}$ is the b norm of the amount of predicted required workload that has not been executed. The weight matrix $\mathbf{T}$ quantifies the importance that executing the workload required from the scheduler has in the optimization process.

Inequality 5.4 defines the range of working frequencies that can be used. It enables a continuous range of frequency settings but this does not prevent from adding in the optimization problem a limitation on the number of allowed frequency values. Equation 5.5 defines the evolution of the system according to the present state and inputs. Equation 5.6 states that temperature constraints should be respected at all times and in all specified locations. The maximum allowed temperature is $Tmax$. $Tmax \cdot \mathbf{1} = \mathbf{t}_{\max}$ and $\mathbf{1}$ is a vector of all ones. Since the system cannot execute jobs that have not arrived, every entry of $\mathbf{u}_\tau$ has to be greater than or equal to 0 as stated by Equation 5.7. The undone work at time τ, u_τ is defined by Equation 5.8. The relation between the power vector $\mathbf{p}$ and the working frequencies is expressed in previous section by Equation 4.13. In Equation 5.9 we relax Equation 4.13 to an inequality. It

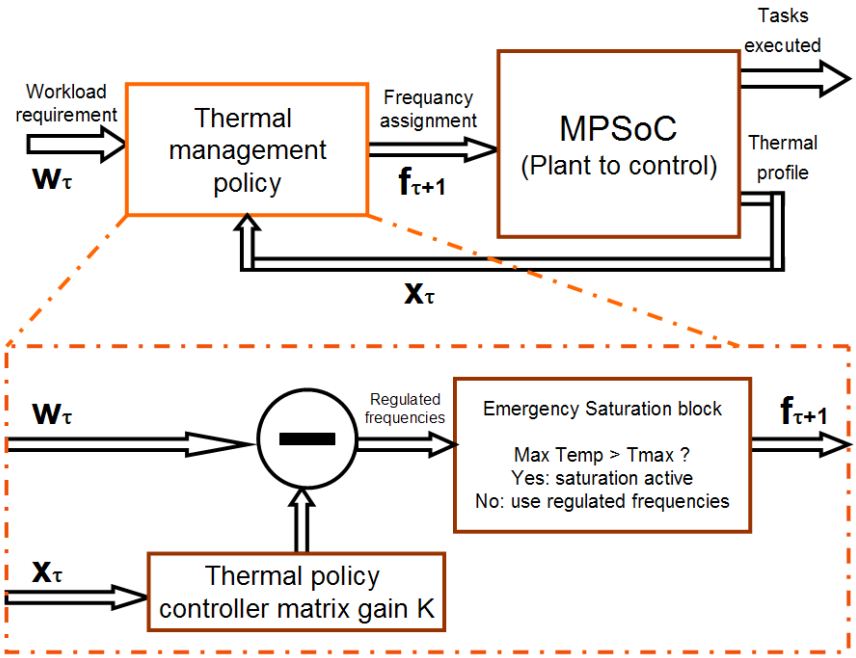

Figure 5.7: Linear Quadratic Regulator-based policy block diagram.

can be shown that the resulting relaxed convex problem is equivalent to the original problem with the quadratic equality constraint [[12], page 191].

The control problem is formulated over an interval of h time steps, which starts at current time τ. For this reason, the approach is said to be predictive. The result of the optimization is an optimal sequence of future control moves (i.e., frequency settings for the cores). Only the first sample of such a sequence is actually applied to the process; the remaining moves are discarded. At the next time step, a new optimal control problem based on new temperature measurements and required frequencies is solved over a shifted prediction horizon. Such a 'receding-horizon' [2] mechanism represents a way of transforming an open-loop design methodology into a feedback one, as at every time step the input applied to the process depends on the most recent measurements.

5.4.1 Linear quadratic regulator

The *linear quadratic regulator* targets temperature and power difference minimization using a linear discrete time system.

If $Tmax$, the maximum MPSoC temperature is less than a certain threshold, the overall system presented in Figure 5.7 is a linear feedback system, where MPSoC frequencies are calculated simply by subtracting from the workload requirement $\mathbf{w}_\tau$ the product of the state vector $\mathbf{x}_\tau$ and the controller matrix gain $\mathbf{K}$. The state vector $\mathbf{x}$ is related to the thermal profile by Equation 7.9. The emergency saturation block (in Figure 5.7) just saturates the regulated frequencies to a certain value when the maximum MPSoC temperature

is higher than the threshold $Tmax$. This enables the MPSoC to cool down and so to reduce its maximum temperature in case of overheating. This policy is based on the work of [96].

Problem formulation

By looking at the general model of Equations 5.2 - 5.9, the problem can be formulated in the following way.

The horizon is infinite and the reference (the requested workload $\mathbf{w}_\tau$) is assumed to be constant over all this period. Matrix $\mathbf{T}$ is a zero matrix and the norm $g = j = 2$. Thus, the objective function is a 2-norm. Because this method is linear, it is not possible to include any constraint inside the problem formulation. For this reason no frequency, temperature or undone work constraint can be added. However all these constraints are considered after the optimization. Frequencies outside the frequency range are discarded. A bias signal $\mathbf{w_t}$ is added to the control loop to force the system to execute the requested workload $\mathbf{w_t}$. Finally an emergency saturation block is added to avoid that the maximum MPSoC temperature is higher than the threshold $Tmax$.

Design phase

The state feedback controller gain $\mathbf{K}$ presented in [33] can be computed via standard matrix computations in negligible time using established tools. The solution of the previous equation exists only if matrix $\mathbf{Q}$ is positive semi-definite and matrix $\mathbf{R}$ is positive definite. However, while the original system is passive, the reduced system is not necessarily passive. However, the reduced system is stabilizable since it's passive and balanced reduction retains stabilizability. Consequently all LQR controllers are stabilizing [33].

Runtime phase

If the MPSoC maximum temperature is under a predefined threshold during the on-line optimization phase, the optimum frequency assignment to achieve thermal balancing is calculated using the following equation:

$$\mathbf{f_{\tau+1}} = \mathbf{w}_\tau - \mathbf{K} \cdot \mathbf{x}_\tau \tag{5.10}$$

where at time τ, $\mathbf{x}_\tau$ is the current state and $\mathbf{w}_\tau$ the workload required in order to fulfill performance requirements. The number of multiplication and additions N_{op} required every time the policy is applied at runtime, is given by the following equation:

$$N_{op} = l \cdot p \tag{5.11}$$

where l is the dimension of the state vector and p is the number of cores of the system.

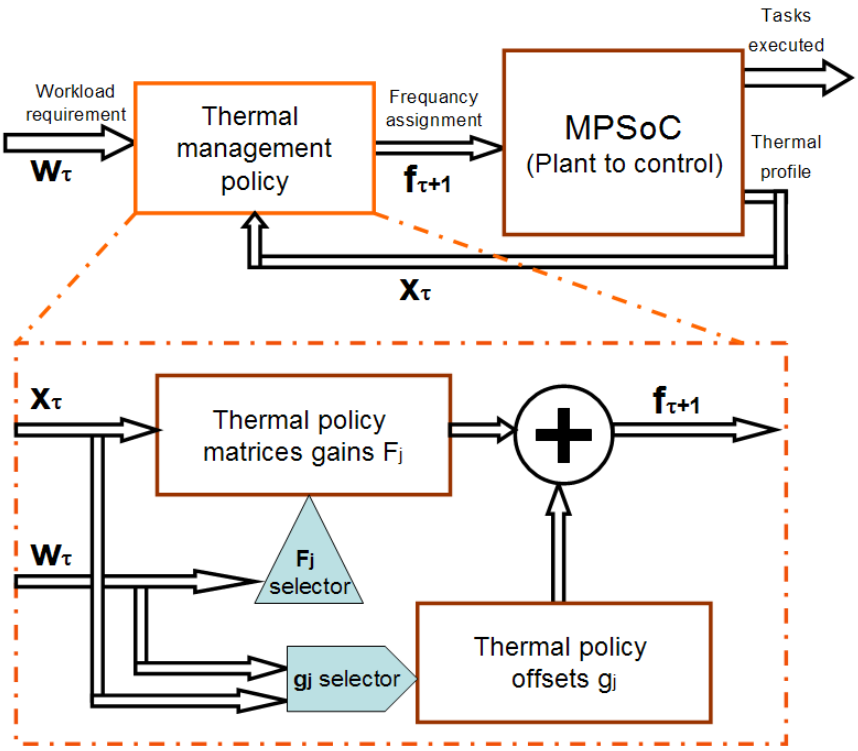

Figure 5.8: Explicit model predictive control-based policy block diagram.

5.4.2 Explicit/Implicit MPC

In general MPC is an optimal control approach aimed at maximizing a performance metric for a linear dynamic system under input/output constraints. The solution of the optimization problem provides the feedback control actions that will determine the frequency setting of the following clock cycles. The policy can be implemented by embedding a numerical solver in the real-time control code (implicit solver), or pre-computed off-line and evaluated through a look-up table of linear feedback gains (explicit solver). Figure 5.8 shows the block diagram of an explicit MPC-based policy. The 'plant to control' is the MPSoC that we want to control. In the case of implicit MPC, the orange dotted box is implemented using an embedded solver that online solves the frequency assignment problem. This policy is based on the work of [95]. The block diagram is shown in Figure 5.10.

Problem formulation

According to the general model of Equations 5.2-5.9, the problem formulation is the following.

The horizon is finite and equal to h and the reference (the requested workload $\mathbf{w}_\tau$) is assumed to be constant over all this period. The state and the power cost function matrices $\mathbf{Q}$ and $\mathbf{R}$ are set to be null matrices. Matrix $\mathbf{T}$

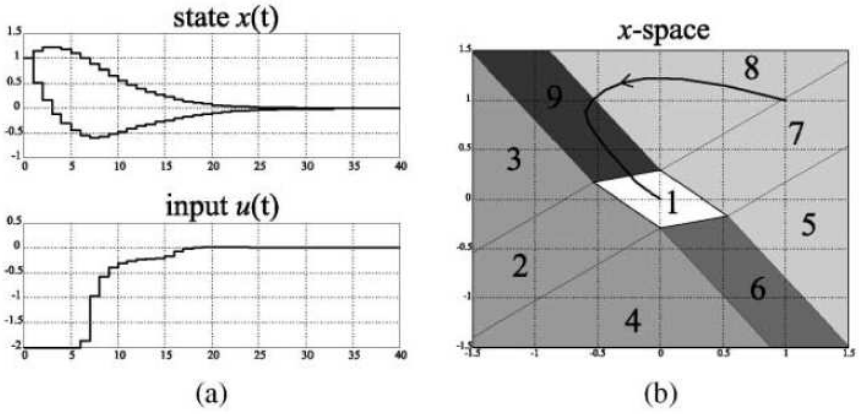

Figure 5.9: Explicit model predictive control example: (a) state values $x(t)$ and resulting input $u(t)$, (b) state-space partition and corresponding control trajectories.

is the identity matrix and the norm $b = 2$. Thus, the objective function is a quadratic form. All the others constraints expressed by Equations 5.4-5.9 are considered inside the problem formulation.

Implicit versus Explicit Regulator

The proposed control strategy can be implemented in two different ways. The first one is called implicit and requires to solve on-line the minimization problem every time the policy is applied. Thus, a significant amount of hardware resources are needed, since the result must be computed in a time frame shorter than the thermal time constants of the MPSoC.

An alternative approach has been proposed in [2]. In this case, the problem is solved off-line in a way that makes explicit the dependence of the solution of the frequency assignment problem $\mathbf{f}_\tau^\alpha$ on input vectors $\mathbf{f}_\tau^\alpha$, $\mathbf{w}_\tau^\alpha$ and $\mathbf{x}_\tau$. The state space can be divided into a set of regions, bounded by linear inequalities (i.e., a polytope), and in each region a different linear controller can be specified and computed off-line.

Then, the controller selection can be efficiently performed on-line by simply checking region boundaries. The resulting controller structure is defined in any of the M partitions as follows:

$$[\mathbf{f}_{\tau+1}^\alpha] = \mathbf{F_j} \begin{bmatrix} \mathbf{x}_\tau \\ \mathbf{f}_\tau^\alpha \\ \mathbf{w}_\tau^\alpha \end{bmatrix} + \mathbf{g_j} \quad \textbf{if} \quad \mathbf{H_j} \begin{bmatrix} \mathbf{x}_\tau \\ \mathbf{f}_\tau^\alpha \\ \mathbf{w}_\tau^\alpha \end{bmatrix} \leq \mathbf{k_j} \tag{5.12}$$

where matrix $\mathbf{F_j}$ and vector $\mathbf{g_j}$ are the gain and offset coefficients of the j^{th} region (see Figure 5.8). Each region is identified by affine inequalities defined by the matrix $\mathbf{H_j}$ and vector $\mathbf{k_j}$ in Equation 5.12 (see [2] for more details).

An example of explicit MPC region partitioning is shown in Figure 5.9. The task is to regulate the system to the origin while fulfilling the input constraint

$-2 \leq u(t) \leq 2$. The system to control consists of 2 entries for the state vector $x(t)$ and one input entry $u(t)$. The resulting explicit MPC controller is shown in Figure 5.9 (a). As it can be noted, the overall 2-dimensional state space is divided into 9 regions. For every region there is a specific controller that generates the optimum input value to control the system. In Figure 5.9, the controller starts from Region 7 and ends in Region 1 passing through Regions 8 and 9. Without loss of generality, in this example the region is determined only by the state value of the system.

If the partitions are properly stored, the number of operations depends logarithmically on the partitions [87]. Nonetheless, while the computer code for evaluating MPC in the explicit form is certainly simpler than the code embedding the QP solver, from the point of view of memory requirements, the explicit form may be more demanding, as M and the matrices to be stored in look-up tables may be large.

The length of the prediction horizon h affects the number of regions. There is indeed (worst-case scenario) an exponential relation between these two parameters. However the longer the prediction horizon is, the better the policy performance is. Thus, for large horizons an implicit implementation may be more convenient in terms of power dissipation and area consumption than an explicit one or vice-versa.

According to Equation 5.12, the number of coefficients N_c to implement the optimum explicit approach of our method is given by the following expression:

$$N_c = N_{reg} \cdot (\overbrace{2p(p+l/2)}^{\mathbf{F_j}} + \overbrace{p}^{\mathbf{g_j}} + \overbrace{2N_{avg} \cdot (p+l/2)}^{\mathbf{H_j}} + \overbrace{N_{avg}}^{\mathbf{k_j}}) \tag{5.13}$$

N_{reg} is the number of regions of the explicit controller, p is the number of processing cores and l the dimension of the state vector $\mathbf{x}$. N_{AVG} is the average number of affine inequalities that identifies each region. In this equation, the symbol $\overbrace{\ }$ highlights contributions for each of the matrices in Equation 5.12.

The computational effort of the evaluation can be computed from [87] and Equation 5.12 and, assuming an average-case scenario, is provided by the following equation:

$$N_{comp} = 1.7 \cdot log_2(N_{reg}) \tag{5.14}$$

$$N_{Mult-Add} = 2p(p+l/2) \tag{5.15}$$

where N_{comp} is the number of required comparisons and $N_{Mult-Add}$ is the number of required multiplications/additions.

5.4.3 Approximated explicit predictive policy

This method is similar to the explicit approach presented in Section 5.4.2 and both its block diagram and its problem formulation are presented in Figure 5.8. Compared with the optimum approach (see previous section), it provides a significant reduction in hardware requirements and computational cost at

the expense of a small loss in accuracy. This enables an increase of the prediction length and accuracy of the MPSoC model and so the performance of the method. This policy is based on the work of [100].

The solution of the optimization problem is computed off-line in a way that makes explicit the dependence of the solution of the frequency assignment problem $\mathbf{f_{\tau+1}}$ on input parameters $\mathbf{w}_\tau$ and $\mathbf{x}_\tau$. The resulting explicit controller is *piecewise polynomial*. In other words, the state space can be divided by a set of regions, bounded by linear inequalities (i.e., a polytope) and in each region a different polynomial controller can be specified and computed off-line [2]. Then, the controller selection can be efficiently performed on-line by simply checking region boundaries.

Approximation algorithm

The proposed method computes an approximate convex *Piece-Wise Affine* (PWA) lower bound of the optimal cost function J (Equations 5.2-5.9). Since this approach proceeds in an incremental greedy-optimal fashion, it is possible to stop the process when any desired level of complexity, or approximation accuracy, is reached. The control law is then derived from this lower bound using the barycentric technique proposed in [92]. The result is a nonlinear and smooth piecewise polynomial control law. In particular, the algorithm is divided into two main phases.

The first phase of the algorithm iterates two steps. In the first step, we compute the level of approximation and a point that obtains this level. In the second step, the approximation is updated such that the error is maximally reduced around this point. These two steps are iterated until the desired accuracy is achieved. It can be shown that any desired approximation error can be achieved in finite time for any convex function.

When the iteration is complete, we apply the second phase of the algorithm, where we define a polynomial for each region by interpolation of the optimal control law at the vertices of the region. The result is a smooth, piecewise polynomial control law.

5.4.4 Convex optimization based policies

Figure 5.10 shows the block diagram of this policy. In the case the problem formulation solver is implemented using an embedded solver that solves the frequency assignment problem online.

Problem formulation

According to the general model of Equations 5.2-5.9, the problem formulation is the following. The horizon is finite and equal to h. The state cost matrix $\mathbf{Q}$ is set to be a null matrix. Matrix $\mathbf{R}$, responsible for the power minimization

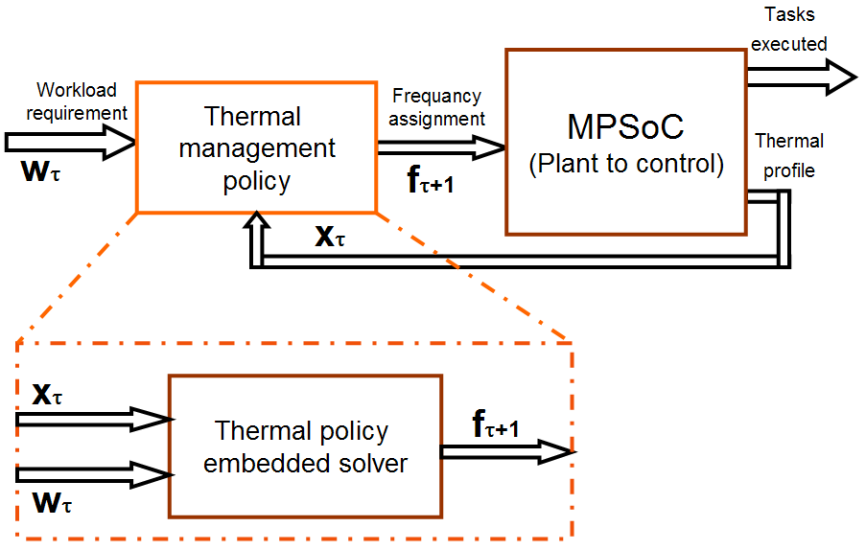

Figure 5.10: Embedded solver-based policy block diagram.

is an identity matrix and the norm $j = 1$. All the others constraints expressed by Equations 5.4-5.9 are considered inside the problem formulation.

The major differences with respect to previous problem formulations are the following. Matrix $\mathbf{T}$ is time varying, the norm $b = 1$ and the reference (the requested workload $\mathbf{w}_\tau$) is time-varying as well. The algorithm indeed dynamically predicts the future workload requirements $\mathbf{w}_\tau$ and the reliability of the estimation is embedded in the problem formulation by matrix $\mathbf{T}$. The larger the prediction error is, the smaller undone work is and vice-versa.

The accuracy of this prediction is embedded in the weighting matrix $\mathbf{T}$, which is chosen accordingly to the reliability of the workload prediction. We have chosen these parameters to achieve a good prediction, according to empirical studies performed on different benchmarks for representative examples of the MPSoCs under study in this thesis.

5.5 Experimental Results

5.5.1 Policies setup

In all our experiments the thermal management policies are applied every $T_{pol} = 10ms$, while the simulation step for the discrete time integration of the RC thermal model (Section III-c) has been set to 200μs. The maximum temperature limit is set to $370°K$. The room temperature is set to $300°K$. In the problem formulation, we used $\alpha = 2$ [62] to establish the relation between the frequency setting and the power consumption because we assumed that devices are working in velocity saturation [70].

Controller	Number of regions	Number of vertices	Computational effort (#operations)		
			search	control law	total
MPC-100	1	66	0	858	858
MPC-200	14	136	30	1025	1055
MPC-300	32	233	50	1184	1234
MPC-400	51	328	60	1286	1346
MPC-500	72	431	60	1335	1395
MPC-600	87	528	65	1354	1419
MPC-opt	3770		89552	16	89568

Controller	Storage Space (#coefficients)			Time design	Time run
	search	control law	total	[s]	[ms]
MPC-100	0	366	366	40.13	4.29
MPC-200	204	915	119	46.92	5.27
MPC-300	1560	1770	3330	50.75	6.17
MPC-400	3186	2778	5964	55.90	6.73
MPC-500	4986	3771	8757	63.61	6.97
MPC-600	7212	4683	11859	71.46	7.09
MPC-opt	89552	60320	149872	196.32	447.84

Table 5.3: Comparative table of MPC-based thermal approaches having different approximation complexities

Threshold based DVFS (TB-DVFS)

This policy sets the frequencies of the cores according to the requirements coming from the scheduler. If the maximum chip temperature goes above a specified threshold (in this case study set to $370°K$), the policy sets the frequency up to 62.5% of the maximum one [62].

Linear Quadratic Regulator (LQR)

In the setting, we focus more on keeping the thermal profile uniform rather than minimizing power consumption. Thus, in the problem formulation we minimize thermal unbalancing while respecting the power limit. According to our experimental model, where the number of cores p equals 8 and n equals 30, the number of required coefficients to store, multiplications and additions equal to $30 \cdot 8 = 240$. These operations need to be performed every T_{pol}.

Model predictive control (MPC)

To design the regulator, we used a Matlab-based development platform provided by [44]. Table 5.3 compares the approximated approach with the optimum approach proposed in [95]. The resulting polynomial control laws are

ranging from 100 to 600 vertices. The penultimate column reports the time in seconds that a MacBook Pro (2.8GHz, Core 2 Duo, 4GB ram) took to design the controllers. By comparing with the optimal controller, we get a reduction in the computation time ranging from 2.7× to 4.9×. The last column reports the time needed to run-time execute the policy assuming a processor able to execute $200K$ FLOPs. A reduction versus the optimum approach ranging from 63.1× to 104.4× can be achieved.

Convex optimization (Convex-opt)

The linear predictor has been designed using a 3^{rd} order polynomial equation [99], an observation window of $600ms$ and a prediction length equal to $50ms$ in the future. We assumed to have two frequency inputs controlling the MPSoC. The first frequency input controls cores 1, 4, 5 and 8. The second input sets the frequency value for cores 2, 3, 6 and 7. We suppose that the scheduler performs a workload balancing strategy on the cores to have all the cores running with potentially the same (or very similar) active frequency. By doing this we assume that the power consumption of the cores depends mostly on their frequency setting. To solve the problem formulation, we use CVX [34], an efficient convex optimization solver. C++ optimized implementations of this software take few microseconds (for the case study described in the experimental set-up section) to run on a state-of-the-art solver [27]-[13], which is at least 3 orders of magnitude less than the interval between two consequent applications of the policy (from 10ms to 100ms).

5.5.2 Executed workload and working temperature

Figure 5.11 compares the performance of the policies by plotting the normalized executed workload versus the thermal profile. The normalized executed workload is the workload executed by the policies normalized to the one that can be executed by the MPSoC running with the highest allowable frequency setting. The top plot analyzes the average maximum chip temperature while the bottom one shows the maximum MPSoC temperature peak. The average maximum chip temperature is the average of the maximum temperature over all the MPSoC.

As these figures show, in the case where no thermal policy is used (upper right circle), even for an average workload of less than 65%, the maximum chip temperature gets to almost 398^oK. All the thermal policies compared in this work avoid this problem. Indeed the maximum temperature peak of all the policies is less than 370.46^oK. The reason of these extra 0.46^oK (above the maximum temperature threshold $370^\circ K$) is because the simulated policies use an extended thermal model as compared to the policy computing method. In reality, considering the cache misses, pipeline stalls, and other events that could increase the CPI, this parameter cannot be estimated accurately. In

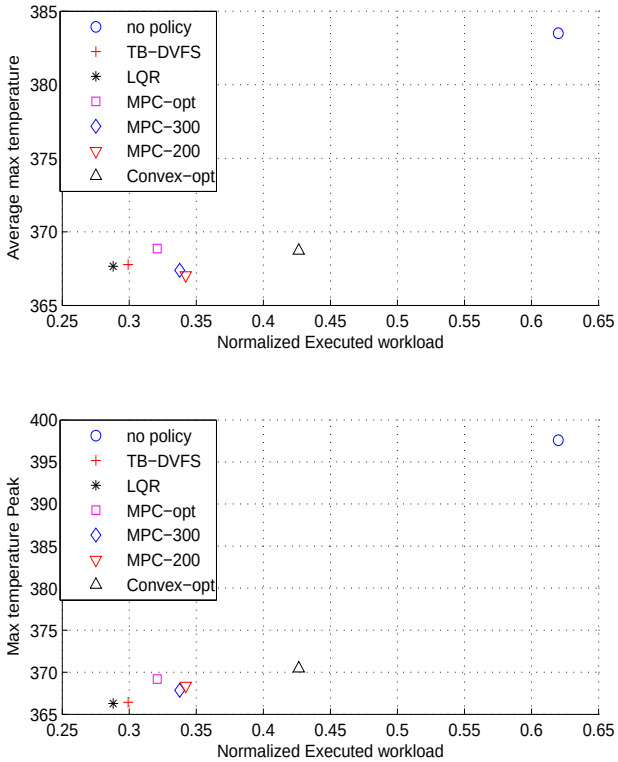

Figure 5.11: Temperature Vs executed workload normalized to the one that can be executed by the MPSoC running with the highest possible frequency setting.

practice, while the maximum temperature peak is slightly above the threshold by 0.46^oK, the average maximum temperature never exceeds the threshold.

The best performance in terms of executed workload is provided by convex optimization-based policy. This policy outperforms the TB-DVFS in terms of executed workload by a factor of 50% by having approximately the same average temperature. The approximated MPCs (with 200-300 vertices) provide almost the same performance with negligible differences in temperatures. The optimum MPC provides around 2% lower performance. The reason is given by the fact that the optimum MPC changes the frequency values more often than the approximated versions of it. The change in the frequency setting has an overhead in terms of additional power dissipation that is not considered in the problem formulation. However, our results show that this effect leads to a loss of performance less than 6% compared with the approximated MPC.

The best performance in terms of executed workload is provided by convex optimization-based policy. This policy outperforms the TB-DVFS in terms of executed workload by a factor of 50% by having approximately the same average temperature. The approximated MPCs (with 200-300 vertices) provide almost the same performance with negligible differences in temperatures. The optimum MPC provides around 2% lower performance. The reason is given by the fact that the optimum MPC changes the frequency values more often than the approximated versions of it. The change in the frequency setting has

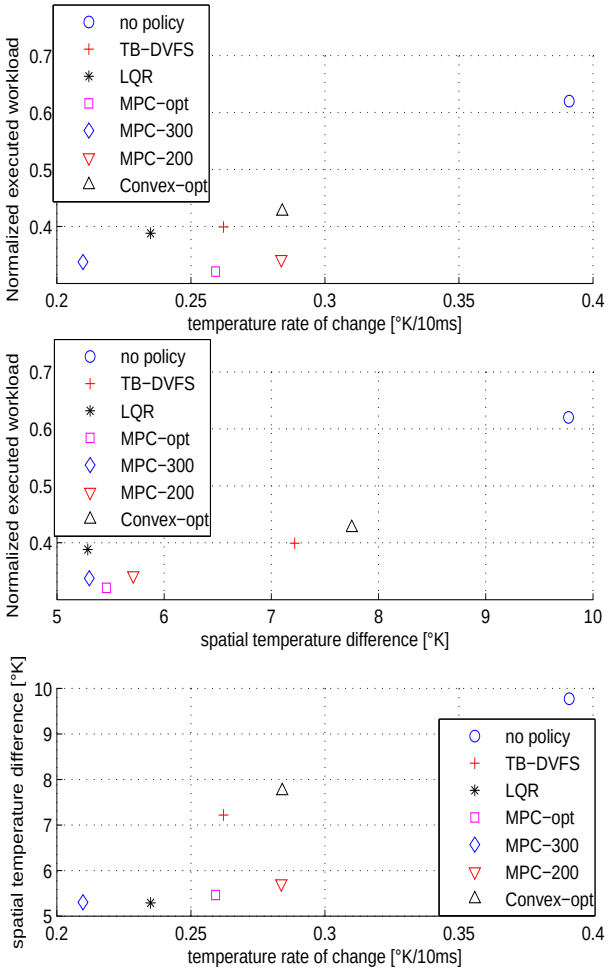

Figure 5.12: Temperature gradients analysis. Normalized executed workload Vs temporal temperature gradient (top); normalized executed workload Vs spatial temperature gradient (center); spatial Vs temporal temperature gradient (bottom).

an overhead in terms of additional power dissipation that is not considered in the problem formulation. However, our results show that this effect leads to a loss of performance less than 6% compared with the approximated MPC.

The TB-DVFS and the LQR policies show lower performance, as they provide 50% and 25% less executed workload than the convex-optimization approach and the approximated MPCs polices, respectively. The LQR policy shows an executed workload that is few percent lower the same compared with the TB-DVFS that is much simpler. Advantages of the LQR are shown in Figure 5.12. Temperature gradients are a concern not only in space, but also in time. The frequent abrupt change in working frequencies and voltages produces thermal cycles that raise the failure rate of the system [35]. In addition, abrupt power-mode transitions in voltage and frequencies scaling waste additional power [42].

Figure 5.12 consists of 3 plots. The first plot on the top compares the po-

lices according to the executed workload and the temperature rate of change. On the y axis there is the workload executed by the policies normalized to the one that can be executed by the MPSoC running with the highest allowable frequency setting. On the x axis there is the temperature rate of change. This metric represents the mean of the maximum absolute temperature difference in degrees on the same MPSoC location between two consecutive applications of the policy. For this reason its dimension is ${}^oK/(T_{pol} = 10ms)$. The second plot in the middle compares the polices according to the executed workload and the spatial temperature difference. This metric represents the mean of the maximum absolute temperature difference in degrees between two neighboring units. Its dimension is ${}^oK$. Both plots compare the polices according to the executed workload and temperature variations. The first one considers variations in the time domain, while the second one variations in the space domain. The third plot analyzes the correlation among time and space temperature gradients for all the compared policies.

The effect of thermal policies is shown in Figure 5.12. Both the spatial temperature difference and the temperature rate of change are reduced in all the policies by a factor up to 2×. Moreover, this is achieved by keeping the maximum temperature under a threshold that ensures the reliable operation of the MPSoC. In the setup of our case study, we worked in a worst case scenario by using a high-quality spreader. In the case of an embedded system, usually using a low-quality spreader, the advantage of using the compared policies is even more evident.

In all the plots of Figure 5.12 the convex optimization approach shows the highest variations in both the time and the space domain. The convex optimization based approach indeed does not target the minimization of space and time temperature gradient, but it targets only the minimization of the power consumption and the difference between the workload requested and the one executed by the MPSoC. Therefore it provides the best performance in terms of executed workload at the cost of a less uniform and a more rapidly changing thermal profile.

The policy with the smallest thermal variations (both in time and space) is the approximated MPC with 300 vertices and the LQR. In all graphs the LQR shows a performance comparable with the one of more complex techniques, like the MPC-300. The LQR, in terms of keeping the temperature under the predefined threshold, works with the same principle as the TB-DVFS. If we compare these two policies, a reduction up to 15% and 45% in time and space temperature variations is achieved. Moreover, the LQR is performing the best in minimizing spatial temperature gradients. The reason is because the LQR policy directly addresses thermal gradients as the main priority. This policy makes the thermal profile more uniform.

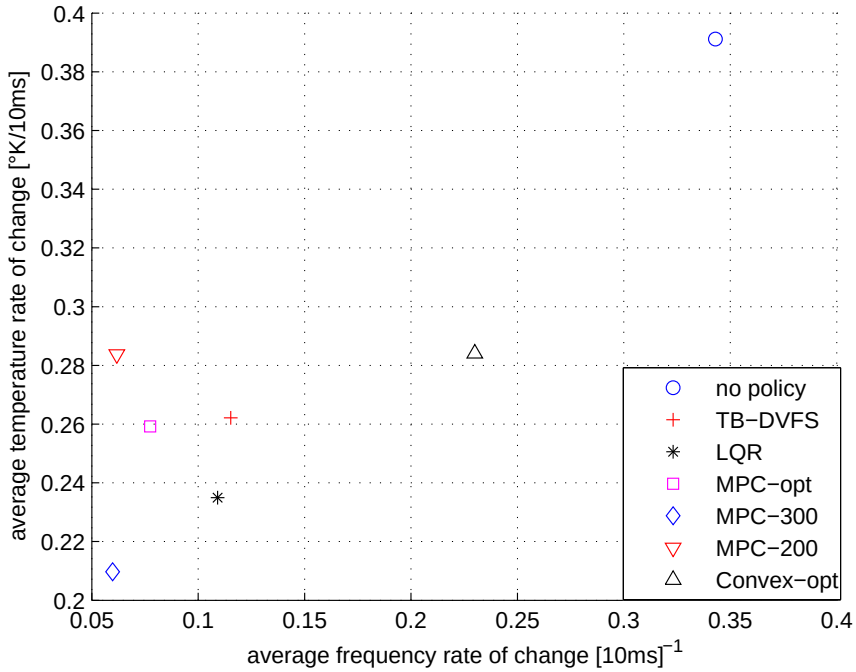

Figure 5.13: Temperature rate of change Vs frequency rate of change.

5.5.3 Thermal and frequency variations

This section makes a study about thermal and frequency variations that generated during the runtime execution of the compared thermal management policies. The more the policy acts smoothly while responding to workload demand, the more both these variations are small.

Figure 5.13 compares the policies according to the average frequency rate of change (x axis) and the average temperature rate of change (y axis). These two metrics represent respectively the average frequency and temperature variation that occurs between two consequent applications of the thermal control method. In our case study the policies are applied every $10ms$, and so the metrics dimensions are $[10ms^{-1}]$ and $[^{\circ}K/10ms]$, respectively.

Figure 5.13 shows that thermal management policies have reduced up to $6\times$ and $1.8\times$ frequency and temperature variations, respectively, compared with the case where no thermal management policies are used. The worst performance is provided by the convex optimization based approach policy. Figure 5.13 also outlines that the optimum MPC has an average frequency rate of change that is higher as compared to the approximated ones. The reason is because the fewer the number of regions are, the less often the frequency settings are changed.

5.6 Summary

In this section we analyzed and explored four control techniques for thermal management of MPSoCs based on MPC. The techniques that we have presented aim at achieving an online smooth thermal control action that minimizes the performance loss as well as the computational and hardware overhead of embedding a thermal management system inside the MPSoC. Our optimization problem considered the thermal profile of the system, its evolution over time and current time-varying workload requirements. We formulated this problem as a discrete-time control problem. The different techniques we presented in this work are characterized according to both the way details are included in the problem formulation and the method used to compute the solution.

We implemented the policies on a MPSoC-simulation platform, and performed experiments on a model of the 8-core Niagara-1 multicore architecture using benchmarks ranging from web-accessing to playing multimedia. Results show different trade-offs between the analyzed techniques. Strength and weaknesses about the compared thermal policies are also discussed.

Policies for Thermal Control with Liquid Cooling

6

In this chapter, we propose two novel online thermal management approaches for 3D multi-processors system on chip using microfluidic cooling. The controller uses dynamic voltage and frequency scaling for the computational cores and adjusts the liquid flow rate to meet the desired performance requirements and to minimize the overall MPSoC energy consumption.

6.1 Introduction

This chapter proposes two main contributions: a centralized and a distributed thermal management algorithm for 3D MPSoC. Despite techniques with air cooling, these techniques use liquid cooling technologies applied to 3D-MPSoCs.

Liquid cooling is performed by attaching a cold plate with built-in microchannels, and/or by fabricating microchannels in the silicon layers of the 3D-MPSoC architectures. Then, a coolant fluid is pumped through the microchannels to remove the heat. The flow rate of the pumps is altered dynamically, and the pump power consumption increases quadratically with the increase in flow rate [14]. Thus its contribution to the overall system energy is not negligible [77]. Details about models used for liquid cooling in 3D-MPSoCs are presented in Section 3.3.

6.1.1 Centralized thermal management

The first contribution of this chapter is a novel centralized thermal management approach for 3D stacks that controls both DVFS and a variable-flow liquid cooling using convex optimization to meet the desired performance and minimal energy requirements.

The optimization is centralized and performed online by an embedded solver. The process is applied at run-time using the convex-solver proposed by [34]. At this stage the convex solver finds the optimum frequency assignment for the inputs of the MPSoC system that will maximize performance under temperature constraints.

Results on the case study described in Section 6.2 show that the proposed method guarantees that scenarios with dangerous thermal profiles are avoided while satisfying the application performance requirements.

Moreover, cooling energy is reduced by up to 50%, as compared with state of the art liquid cooling policies. In addition, the proposed policy keeps the average thermal profile up to $18°C$ lower compared with state of the art polices using variable-flow liquid cooling, like [77].

6.1.2 Hierarchical thermal management

The second contribution of this chapter is an online hierarchical thermal management policy for high-performance 3D systems with liquid cooling. Our proposed controller uses an approach with a global controller regulating the active cooling and local controllers (on each layer) performing dynamic voltage and frequency scaling (DVFS) and interacting with the global controller.

Then, the on-line control is achieved by policies that are computed off-line by solving an optimization problem that considers the thermal profile of 3D-MPSoCs, its evolution over time and current time-varying workload requirements. The proposed hierarchical scheme is scalable to complex (and heterogeneous) 3D chip stacks.

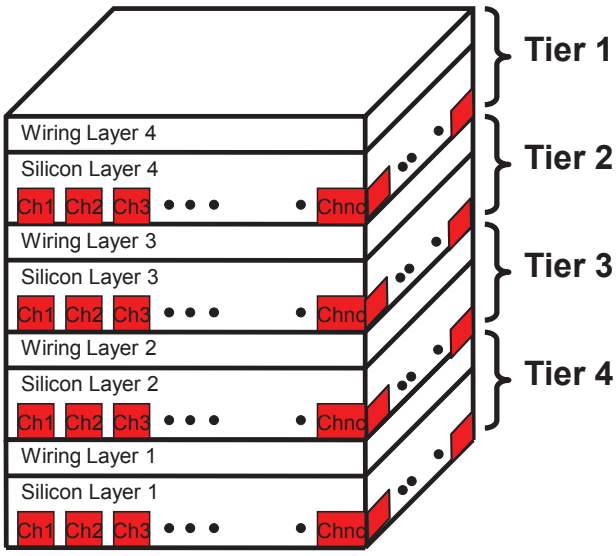

Figure 6.1: Structure of the 4-tier 3D-MPSoC model with interlayer liquid cooling.

Results on the case study described in Section 6.2 show significant advantages in terms of energy savings that reaches values up to 50% versus state-of-the-art thermal control techniques for liquid cooling, and thermal balance with differences of less than 10^oC per layer.

6.2 3D-MPSoC case study

6.2.1 Layout and Technology Specifications

A typical structure of 3D-MPSoC consists of two or more more silicon tiers, with the processing and storage elements of the system. Interlayer liquid cooling is realized by etching *microchannels* in silicon and creating porous structures of different form and shapes. Etching must take into account the *Through Silicon Vias* (TSVs) allocation and spacing requirements.

Figure 6.1 shows an example of a 4-tier 3D-MPSoC with multiple inlets and outlets in different parts of the tiers, as we target in this paper. In this figure the wiring layer is explicitly shown, as thermal properties of interconnect (usually copper) are different from silicon. In this example we show four different floorplans (A, B, C, D) (see Figure 6.2) in the silicon tiers with various processing cores (i.e., UltraSPARC Niagara T1 described by Kongetira et al. [43]), with independent clock frequency and voltage supplies, interconnects (`crossbar`) and memories (`scdata`).

In our thermal model of this 3D-MPSoC, the parameters are provided in Table 6.1. This table contains the thermal conductance and capacitance values of different materials used in modeling the stack.

Figure 6.2: Floorplan of the used silicon tiers in our 3D-MPSoC model.

Table 6.1: Thermal and Floorplan parameters deployed in the model

Parameter	Value
Silicon conductivity	$130 W/(m \cdot K)$
Silicon capacitance	$1635660 J/(m^3 \cdot K)$
Wiring layer conductivity	$2.25 W/(m \cdot K)$
Wiring layer capacitance	$2174502 J/(m^3 \cdot K)$
Water conductivity	$0.6 W/(m \cdot K)$
Water capacitance	$4183 J/(kg \cdot K)$
Die thickness (one stack)	$0.15 mm$
Area per core	$10 mm^2$
Area per L2 cache	$19 mm^2$
Total area of each layer	$115 mm^2$

6.2.2 Frequency Setting and DVFS

This architecture has a maximum operating frequency of 1.2 GHz. To implement the voltage and frequency scaling techniques, we use frequencies ranging from 0 to 1.2GHz.

In this range, only specific values of frequencies are allowed, thus the values for the frequencies different from 0 are expressed by Equation 6.1.

$$f = \frac{6 \cdot 10^9}{D_f} \quad \forall\ D_f,\ subj.\ to: \quad 5 \leq D_f \leq 35 \tag{6.1}$$

where D_f is the division factor needed by the clock tree generator. To simu-

Table 6.2: Microchannel-based parameters used in different topologies

Parameter	straight channel	Bent channel
Channel width	$50\mu m$	$50\mu m$
Channel height	$100\mu m$	$100\mu m$
Channel pitch	$150\mu m$	$150\mu m$
Channel length	$11mm$	$2.5 - 11mm$
Number of channels per tier	67	90
Max flow rate per tier	$0.0323l/min$	$0.0686l/min$

late the system we used different benchmarks, ranging from web-accessing to playing multimedia [22], [23].

6.2.3 Cooling Model

In our approach, we experiment with both straight and bent microchannels, having 2 and 4 ports respectively (see Figure 3.8). In both cases the microchannel cross section is constant. The straight microchannels have all equal length (i.e., they go from side to side of the chip). The length varies in bent channel structures.

The geometry of the cooling layer is related to the following factors: the microchannel topology and dimensions, and the TSVs sizes and spacing requirements. In our model, we use $50\mu m$ diameter TSVs with $100\mu m$ spacing requirements. The microchannel-related parameters is shown in Table 6.2 for both straight and bent channels. This table shows that the amount of injected fluid in the case of bent channels is more than that of straight channel. This increase implies better heat removal capabilities, but the increase in flow rate comes with an increase of the pumping power.

We assume that there is only one pump connected to all microchannels of all the layers, such as a centrifugal pump EMB MHIE [29], is responsible for the fluid injection to the whole system. This pump has the capability of producing large discharge rates at small pressure heads.

Liquid is injected to the stack from this pump via a pumping network. To enable using different flow rates for each stack, we control the fluid via control valves we include in the network. We assume *normally closed valves* (NCV) provided by Festo group [32]. NCVs use external power to reduce the pressure drop and to increase the flow rate. Without loss of generality, this configuration is scalable into different pumping networks, where different valves are used to control the fluid in every tier.

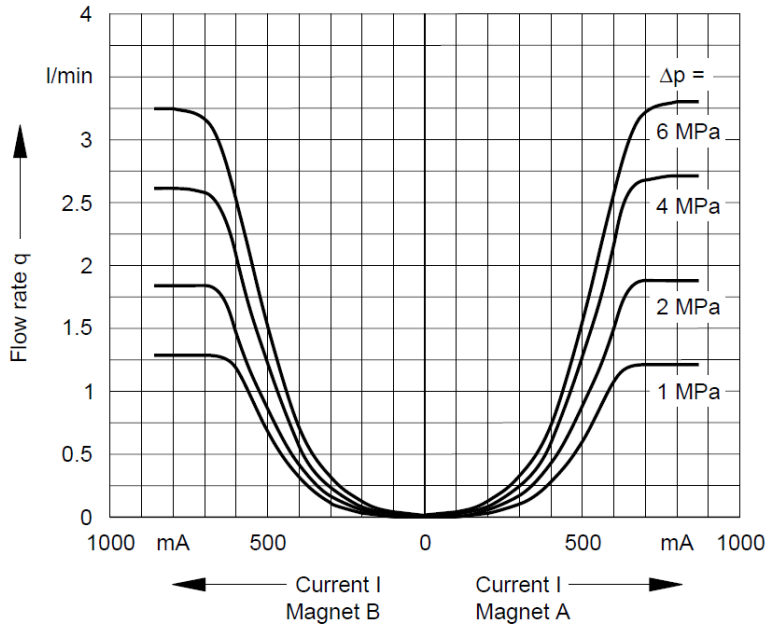

Figure 6.3: Electric current absorption (power consumption) and flow rates of the cooling infrastructure per one tier. Data from [32]

6.2.4 Cooling System Power Consumption

Figure 6.3 shows the relation between the increase of electric current (power) absorption consumed by the pump and the increase of injected fluid. As it can be noted there is a quadratic dependance between the current absorption and the liquid cooling flow rates.

Figure 6.4 shows the power consumed by the pump and valve per tier to inject the fluid from a single at a certain flow rate and pressure difference. In the case of straight channels, we use the same plotted values. However, in the case of bent channels, we increase the energy consumed by the pump only to account for the increased amount of injected fluid at the same pressure difference. Unlike the valve energy which is a function of the pressure difference, not the flow rate. Thus, the valve energy remains the same for both straight and bent channels.

6.2.5 3D-MPSoC Power Consumption

To perform the thermal evaluation and calculate the temperature distribution among the chip, we refer to power consumption values we got in Section 5.2.4. These values are derived from the work by Leon et al. [49] and are used for each of the units of the 4 tiers composing the 3D structure of Figure 6.2.

We dynamically calculate the leakage power of processing cores as a function of their area and actual run-time temperature. We use a base leakage power density of $0.25 W mm^2$ at $383°K$ for $90nm$ technology [9]. Thus, the

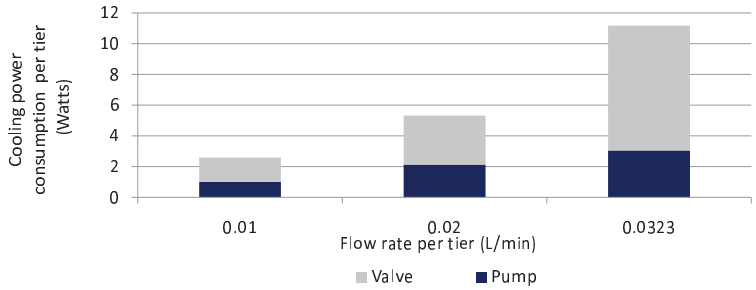

Figure 6.4: Power consumption and flow rates of the cooling infrastructure per one tier.

leakage power at a temperature $T°K$ is given by: $P(T) = P_o \cdot e^{\beta(T-383)}$, where P_o is the leakage power at $383°K$, and β is a technology dependent coefficient. We set $\beta = 0.017$ [77].

6.2.6 Benchmarks

We use workload traces collected from real applications running on an UltraSPARC T1. We record the utilization percentage for each hardware thread at every second using *mpstat* for several minutes for each benchmark.

We use various real-life benchmarks including web server, database management, and multimedia processing. The web server workload is generated by SLAMD [82] with 20 and 40 threads per client to achieve medium and high utilization, respectively.

For database applications, we experiment with MySQL using sysbench for a table with 1 million rows and 100 threads. Finally, we run several instances of the mplayer (integer) benchmark as typical examples of multimedia processing. The utilization ratios are averaged over all cores throughout the execution.

We assume as initial condition, the system to be at room temperature set equal to $300°K$.

6.3 Centralized Thermal Management

6.3.1 Policy computation

The proposed thermal management approach uses both DVFS and variable-flow liquid cooling to meet the desired requirements, which are represented by a two-term cost function. The first one is related to power minimization (3D-MPSoC power consumption and liquid cooling pumping system power consumption) and the second one to the performance loss (undone work). The solution of following minimization are the 3D MPSoC frequencies and cooling pumps speeds necessary to meet the desires requirements. The control problem

is formulated as the following convex optimization problem:

$$J = \sum_{\tau=1}^{h} \left(\|\mathbf{Rp}_\tau\|_\mathbf{j} + \|\mathbf{Tu}_\tau\|_\mathbf{b} \right) \tag{6.2}$$

$$min \;\; J \tag{6.3}$$

$$subject\ to: \quad \mathbf{f}_{\min} \preceq \mathbf{f}_\tau \preceq \mathbf{f}_{\max} \;\; \forall\, \tau \tag{6.4}$$

$$\mathbf{x}_{\tau+1} = \mathbf{Ax}_\tau + \mathbf{Bp}_\tau \;\; \forall\, \tau \tag{6.5}$$

$$\tilde{\mathbf{C}}\mathbf{x}_{\tau+1} \preceq \mathbf{t}_{\max} \;\; \forall\, \tau \tag{6.6}$$

$$\mathbf{u}_\tau \succeq \mathbf{0} \;\; \forall\, \tau \tag{6.7}$$

$$\mathbf{u}_\tau = \mathbf{w}_\tau - \mathbf{f}_\tau \;\; \forall\, \tau \tag{6.8}$$

$$\mathbf{l}_\tau \succeq \mu \mathbf{f}_\tau^\alpha \;\; \forall\, \tau \tag{6.9}$$

$$-\mathbf{w} \preceq \mathbf{m}_{\tau+1} - \mathbf{m}_\tau \preceq \mathbf{w} \;\; \forall\, \tau \tag{6.10}$$

$$0 \preceq \mathbf{m}_\tau \preceq \mathbf{1} \;\; \forall\, \tau \tag{6.11}$$

$$\mathbf{p}_\tau = [\mathbf{l}_\tau; \mathbf{m}_\tau] \;\; \forall\, \tau \tag{6.12}$$

It is important to highlight that the matrices $\mathbf{A}$, $\mathbf{B}$ used in previous equations are constant during the h time steps the system tries to minimize the cost function J, and are then updated every time the policy is applied. In our optimization problem formulation, h is the time horizon [2](or number of time steps) to minimize the cost function J. Then, matrices $\mathbf{A}$, $\mathbf{B}$ are constant during these next h time steps, and are then updated every time the predictive policy is applied.

Function J is expressed by a sum where the summation index τ ranges from 1 to h. The first term $\|\mathbf{Rp}_\tau\|_\mathbf{j}$ is the j norm (in our implementation $j = 1$) of the power input vector p weighted by matrix $\mathbf{R}$. Power consumption is generated here by two main sources: the voltage-frequency setting of the 3D MPSoC and the liquid cooling pumping power. Vector p is a vector containing normalized power consumption data of both the cores and the cooling pumps. Matrix $\mathbf{R}$ contains the maximum value of the power consumption of both the cores (first p diagonal entries) and the cooling pumps (last z diagonal entries).

The second term $\|\mathbf{Tu}_\tau\|_\mathbf{b}$ is the b norm (in our implementation $b = 1$) of the amount of predicted required workload that has not been executed. The weight matrix $\mathbf{T}$ quantifies the importance that executing the workload (required from the scheduler) has in the optimization process.

Inequality 6.4 defines the range of working frequencies that can be used. It enables a continuous range of frequency settings but this does not prevent from adding in the optimization problem a limitation on the number of allowed frequency values. Equation 6.5 defines the evolution of the system according to the present state and inputs. Equation 6.6 states that temperature constraints should be respected at all times and in all specified locations. Since the system cannot execute jobs that have not arrived, every entry of $\mathbf{u}_\tau$ has to be greater than or equal to 0 as stated by Equation 6.7. The undone work at time τ, u_τ is defined by Equation 6.8. Equation 6.9 defines the relation between the power

vector $\mathbf{l}$ and the working frequencies. μ is a technology-dependent constant. Since all constraints in the minimization problem must be convex functions, we relaxed the original power equation to the convex inequality of Equation 6.9. By doing this operation we changed the original minimization problem to the problem described by the convex Equations 6.2-6.9. It can be shown that the resulting relaxed convex problem is equivalent to the original problem with the equality constraint [12].

Equation 6.12 defines formally the structure of vector $\mathbf{p}$. Vector $\mathbf{l} \epsilon \Re^{\mathbf{p}}$ is the power input vector, where p is the number of frequency islands composing the 3D-MPSoC. Vector $\mathbf{m} \epsilon \Re^{\mathbf{z}}$ contains the normalized amount of cooling power for each of the z independent pumps. Equations 6.10-6.11 define constraints on the liquid cooling management. Equation 6.11 states that $\mathbf{m}$ is a normalized value and it can range from 0 to 1. Equation 6.10 defines the maximum increment/decrement that the normalized pump can have between two consequent applications of the policy. In other terms this value takes into account the mechanical time dynamics of the pump. Their values are stored in vector $\mathbf{w} \epsilon \Re^{\mathbf{z}}$.

The result of the optimization is an optimal sequence of future control moves (i.e., frequency settings for the cores of the 3D MPSoC which are stored in vector $\mathbf{f}$). To increase the performance of our proposed policy, history information about the task arrival process are exploited by the proposed algorithm. Matrix $\mathbf{T}$ is chosen accordingly to the reliability of the workload prediction. We have selected these parameters to achieve a good prediction, according to empirical studies performed on different benchmarks [22].

6.3.2 Policy setup

According to the general model of Equations 6.2-6.9, the problem formulation is the following. Matrix $\mathbf{T}$ is set to be an identity matrix while matrix $\mathbf{R}$ contains the maximum value of the power consumption of both the cores and the cooling pumps (power values from [20]). The policy minimizes the sum of all contributions to the 3D MPSoC power consumption as well as the undone workload. For this reason, we set both the norms b and j to 1.

All the others constraints expressed by Equations 6.4-6.9 are considered inside the problem formulation. The policy is applied every $T_{pol} = 10ms$, while the simulation step for the discrete time integration of the RC thermal model has been set to 200μs. The maximum temperature limit is set to $370°K$. The room temperature and t_{fluid} are set to $300°K$. In the problem formulation, we used $\alpha = 2$ to establish the relation between the frequency setting and the power consumption. The linear predictor has been designed using a 3^{rd} order polynomial equation, an observation window of $600ms$ and a prediction length equal to $50ms$ in the future. The optimization process is done online using the convex solver proposed in [34]. These operations, have been performed on standard processors (i.e., Core Duo @ 2GHz) in few tenth of microseconds. This time is 3 orders of magnitude smaller compared with the time the policy

is applied (i.e.10ms). The time constants needed by the mechanical dynamics of the cooling pumps to go from 0 to maximum power is set to $400ms$.

6.3.3 Experimental results

Policies under comparison

In our experiments, we compare the proposed 3D thermal management method with state-of-the-art thermal management techniques based on DVFS, load balancing and variable flow liquid cooling ([20], [77], [26]).

Dynamic load balancing (**LB**) [26] balances the workload by moving threads from a core's queue to another if the difference in queue lengths is over a threshold. Temperature-triggered task migration (**TTTM**) [26] moves tasks from a core if that core exceeds the threshold temperature. TTTM has an impact on performance resulting from the time overhead required to move tasks between the cores (e.g., context switch overhead and cold start effects). We assume a $1ms$ overhead when a thread is migrated to a new core [20], [77]. For previously mentioned polices, if the temperature goes higher than $420°K$, the system shuts down until the maximum MPSoC temperature returns below $250°K$. In temperature triggered DVFS (**TTDVFS**) [26] the voltage and frequency settings are reduced to the 10% of the maximum value when the maximum MPSoC temperature exceed the threshold value set to $370°K$. TTTM and TTDVFS can also be combined into a joint policy called (**TTTM_TTDVFS**) [26].

We experiment with both air-cooled (**AC**) and liquid-cooled (**LC**) systems for comparison purposes. In **LC_LB**, we apply 100% of the maximum flow rate (0.0323 l/min per cavity [77]). We also consider in the comparison state of the art liquid cooling methods recently proposed in [20] and [77]. These methods employ a variable-flow liquid cooling combined with DVFS. We refer to the first method as **LC_VF** and to the second one as **LC_Fuzzy**.

Hotspots prevention

Thermal impact of all the policies on the system is shown in Figure 6.5. This figure compares the percentage of time spent above the threshold temperature set to $370°K$. Thus shows each bar the area distribution of the dimension of the hotspot as percentage of the overall MPSoC area.

The first four policies are air cooled methods, while the last four are liquid cooled. The first ones are not able to avoid hot spots. **AC_LB** and **AC_TTTM** present hot spots for more than 67% of the execution time, and in addition to that, these hot spots affect more than 80% of the total MPSoC area. Methods using temperature-triggered DVFS show a better performance. This can be noted by looking at the results for **AC_TTDVFS** and **AC_TTTM_TTDVFS**. They indeded present hot spots for only 34% and

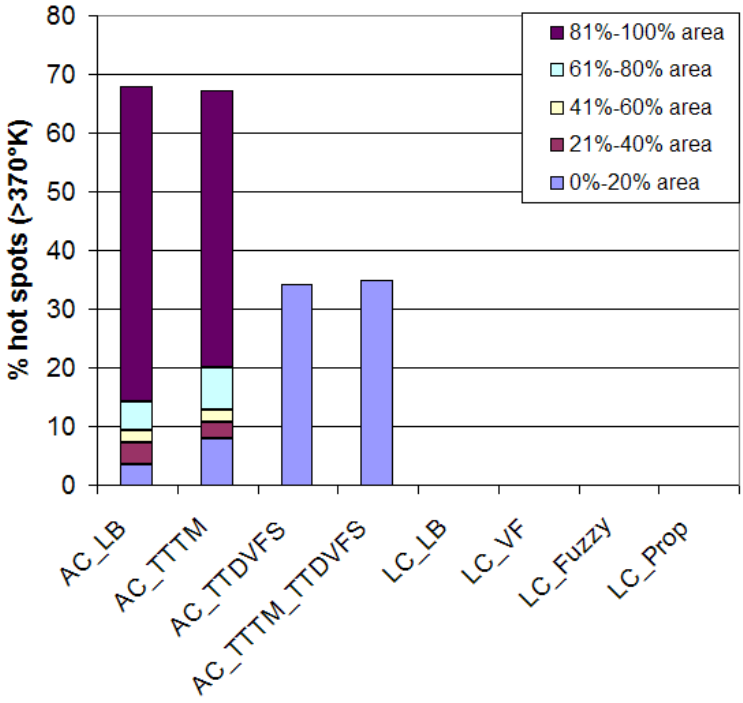

Figure 6.5: Percentage of run-time execution where the maximum MPSoC temperature is higher than the threshold($370°K$). The area of the hotspot is also provided as a percentage of the overall MPSoC area

35% of the execution time, respectively. In addition to that these hot spots cover less than 20% of the overall MPSoC area.

Air cooled policies do not completely avoid hotspots. The reason is because the 4-tier stacked architecture has problems in dissipating the heat of inner layers by using only a heat spreader. Liquid cooling techniques completely avoid any hotspots scenario. The reason is because of their capability to cool inner layers of the 3D-MPSoC of Figure 6.2.

Workload execution performance

Figure 6.6 shows the percentage of the workload requested from the MPSoC that has not been executed due to thermal problems. This is a direct measure of the performance of the system. Liquid cooling policies provides a value of undone workload that is less than 1% of the overall executed workload. Air cooled polices provide values ranging from 24%(**AC_LB**) to 31% (**AC_TTDVFS**).

Previous results show the reason why there is a need for liquid cooling for 3-D structures including multiple layers connected to each others. Because of the fact that we are interested in techniques that avoid hot spots while satisfying performance requirements, we restrict now our comparison to liquid

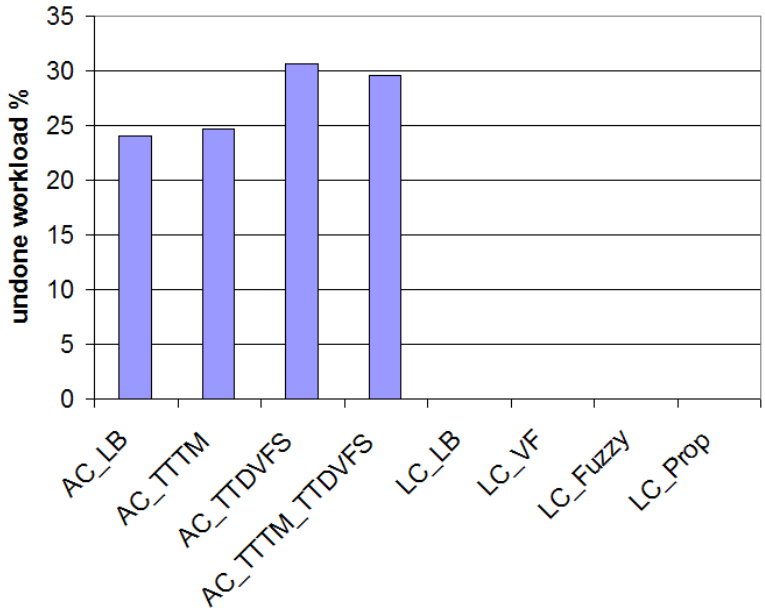

Figure 6.6: Undone work as a percentage of the overall requested workload

cooling methods.

Energy consumption and thermal profile

The left graph of Figure 6.7 shows the overall energy consumption of the 3D MPSoC. It is divided here into two contributions. The first one is the one absorbed by the cooling network (pumps+valves) while the second is the energy absorbed by the MPSoC activity (switching+leakage). The simplest policy **LC_LB** shows the highest energy consumption. The value of the cooling power here represents 24% the overall 3D MPSoC energy consumption. For this reason, **LC_VF** [20] and **LC_Fuzzy** [77] have been proposed. We tested these policies on our experimental setup. They show a reduction in the cooling power consumption by approximately 30% and 50% respectively, according to what mentioned in [77] and [20]. The proposed technique has a cooling and an overall 3D MPSoC power consumption that is respectively 50% and 7% lower compared with **LC_LB**. If we compare our policy with the recently presented technique **LC_Fuzzy** [77], we see approximately the same saving in terms of cooling power and a 3% saving in the overall MPSoC consumption.

To better emphasize advantages of this policy versus **LC_Fuzzy**, the graph on the right side in Figure 6.7 is presented. This graph shows the average maximum 3D MPSoC temperature for all the policies under comparison. It is important to emphasize here that from Figures 6.5 and 6.6 we know that all the policies completely avoid hot spots and execute the workload requested from the 3D MPSoC by the scheduler. The lowest thermal profile among the compared policies is generated by the **LC_LB**. In this case the maximum

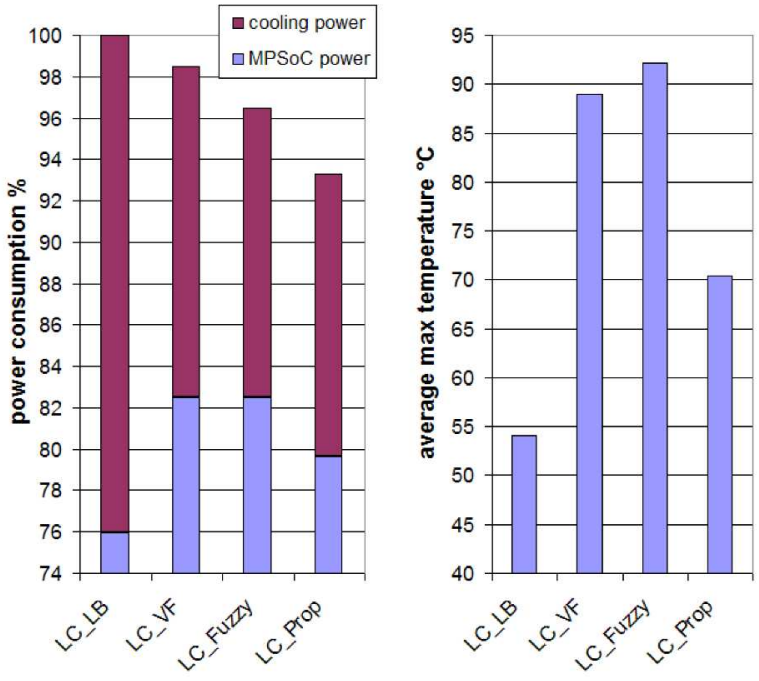

Figure 6.7: left graph: energy consumption of the overall system: 3D MPSoC power consumption and cooling network. Values are normalized to **LC_LB**; right graph: average maximum 3D MPSoC temperature [$^{\circ}C$]

MPSoC temperature has an average value of $54^{\circ}C$. **LC_LB** and **LC_Fuzzy** show a thermal profile having an average maximum temperature of $89^{\circ}C$ and $92^{\circ}C$, respectively. The reason is because both these systems save energy by reducing the cooling cost and by having the system working at a temperature close to the threshold set to $97^{\circ}C$. The proposed policy is able to keep the thermal profile $18^{\circ}C$ lower compared with **LC_Fuzzy**. The main reason is because the predictive problem formulation of the proposed method is able to satisfy performance requirements by acting in advance and this enables the policy to act slower on the system and so to save active power. This keeps the thermal profile colder and this turns also into a leakage power saving.

6.4 Distributed Hierarchical Thermal Policy

6.4.1 Hierarchical structure

The structure of the proposed hierarchical thermal management system is shown in Figure 6.8: the 3D-MPSoC architecture is partitioned into p tiers (or layers) where, without loss of generality, each tier is a subsystem of the 3D-MPSoC. In our exploration, we define a tier as a complete layer.

Moreover, any tier consists of several units. These units could be cores, memory storage units, or other computational units (e.g., ASIC or custom

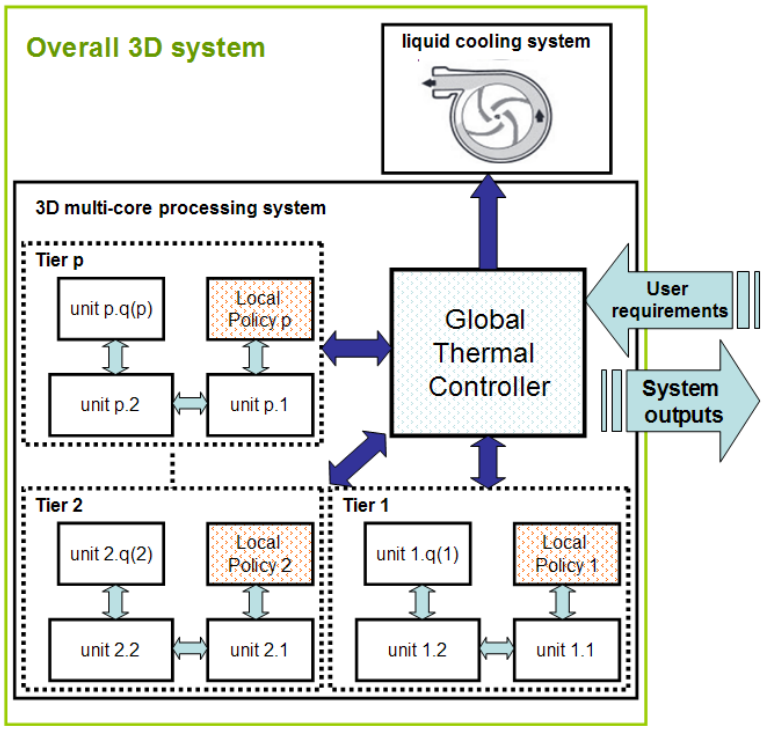

Figure 6.8: Structure of the proposed hierarchical thermal management system

hardware blocks). Then, the units inside each tier, say tier i, are partitioned into $q(i)$ frequency islands, and a local thermal controller manages the $q(i)$ islands, i.e., sets the frequencies and voltages to all (controllable) components inside the tier. Objectives of local controllers include preventing hot-spots and minimizing undone workload. Specific requirements (e.g. workload) come from a centralized unit (i.e., the *global thermal controller* in Figure 6.8), which is responsible for the holistic coordination of the p local thermal controllers, and which regulates the heat extraction of the cooling system by setting the pressure of the coolant liquid (by controlling the cooling pump and/or the controlling valve).

This hierarchical structure is crucial for scalability and feasibility of large MPSoCs [30]. Indeed by using this hierarchical approach, we can significantly simplify the function and overhead of the global controller by using local thermal controllers. Moreover, this structure enables the global and local controllers to be executed with different rates, e.g., the optimization of the global controller can be executed at least one order of magnitude less frequently as compared to the local regulators. The global controller manages the pumping flow rate, which is much slower process than DVFS.

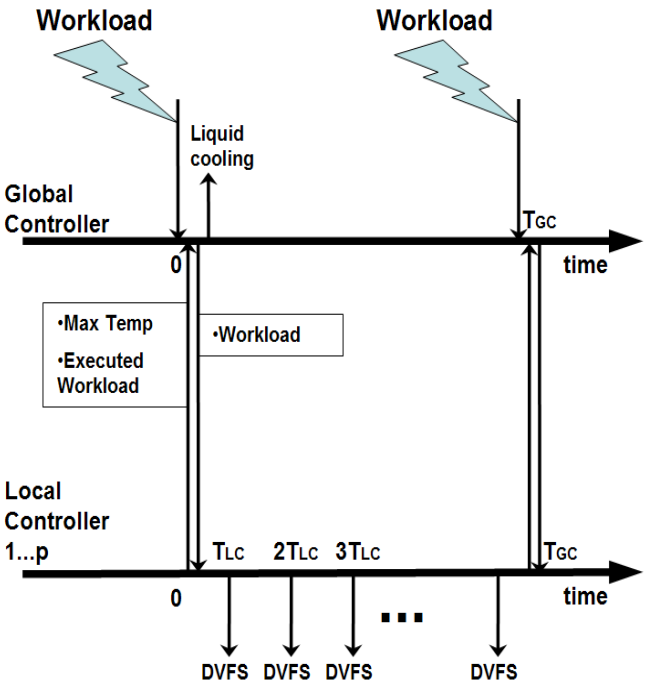

Figure 6.9: Communication protocol between the global and the local controllers of the proposed method

6.4.2 Run-time interaction: global and local controllers

The communication protocol between the local controllers and the global one is shown in Figure 6.9. Initially, the global controller receives a workload requirement from the scheduler as well as a data vector containing their workload fulfillment status in each specific tier from all the p local controllers. This data vector contains two pieces of information: i) the maximum temperature measured on line in the corresponding tier, ii) the already executed workload. Indeed this last information provides the global controller with an overview about how well the local controllers are performing in trying to fulfill overall requirements.

Moreover, as the workload fulfillment data from all the local controllers are collected and processed, the global unit splits the overall workload into p components. Hence, for each local controller, the global unit sets the amount of workload it has to execute. It is important to notice that the controller does not perform detailed task assignment, but just sets individual targets for each tier to satisfy the overall workload. The pressure of the coolant liquid is set during this process by the global controller, which performs this operation periodically, with a period of T_{GC}. Once these tasks are performed, the global controller stays still for the rest of the period T_{GC}. Concurrently each local controller sets periodically the DVFS value of all related islands, but with another period T_{LC}, such that $T_{GC} = n \cdot T_{LC},\ n \in Z^{+}$. The local controllers manage

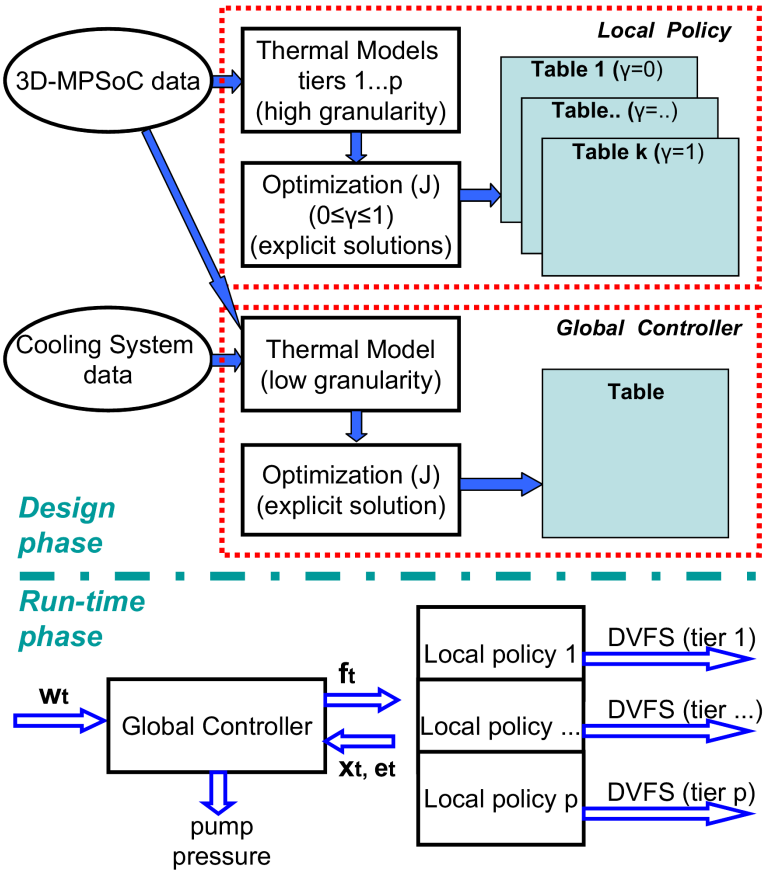

Figure 6.10: Design phase and run-time phase of the proposed hierarchical thermal management

independently the corresponding subsystems and they can communicate with the global thermal management unit only once in the period T_{GC}.

6.4.3 Design and implementation

The design and implementation of the proposed management scheme consists of two phases, i.e., *design phase* and *run-time phase*, which are shown in Figure 6.10. The *design phase* is performed off-line to compute and generate the optimized control decisions of both *local* and *global* controllers. Afterwards, these decisions are allocated in a look-up table-based implementation, at *design phase*, to be used by the global and local controllers at the *run-time phase*.

To compute the tables needed for the implementation of the local controllers, all design data related to the structure of the 3D-MPSoC (e.g., elements layout, thermal conductivity of materials,...) are used to create an accurate thermal model for each one of the p tiers composing the 3D-MPSoC (upper part of Figure 6.10). This model has a fine granularity and can be formulated as an optimization problem. Different explicit solutions (for various

values of the input parameters and optimization goals) are then stored into tables to be used at run-time. To compute the table needed for the implementation of the global controller, we build a coarse-grained thermal model of the 3D-MPSoC and of the cooling system (e.g., the available pumping power values, microchannel layout,...).

During *run-time*, both the global and the local controllers apply the rules stored in the aforementioned look-up tables. Each local controller generates the frequency setting for its tier elements, at the processing element-level granularity, while the global controller sets the pressure for the cooling system pump.

The overall system uses software-driven thermal management. That is, the control action is done by software routines (for both the local and global controller) that access the pre-computed data in the tables. These tables represent the control *policies*. Their computation is described in the following subsections.

Since the global controller performs the major decisions by frequent communication with all local controllers and performing the major control actions to them and the coolant system, a dedicated thread resembling the global controller routine is always active. Moreover, this thread is assigned to single dedicated processing unit that is fully utilized. Being fully utilized implies that, this element should be exposed to the maximum cooling ability available in the system. Thus, in our case of 3D-MPSoC with liquid cooling, a processing element that is nearest to the fluid inlet port(s) at any tier is a possible candidate to be allocated to the global control routine [77].

6.4.4 Policy computation: global thermal controller

The *global thermal controller* is the unit responsible for the global joint operation of all local controllers and for the pump control which sets the coolant pressure.

The workload to be dispatched to each local controller is stored in vector $\mathbf{f}_\tau$. The p entries of this vector contain the average frequency of operation at which each local controller has to work in order to execute the workload assigned to its controlled tier by the global unit.

The global controller policy minimizes power and undone workload. Furthermore, the performance requirements coming from the scheduler have to be fulfilled and the maximum temperature constraint satisfied. The problem can

be defined as follows:

$$J = \sum_{\tau=1}^{h} \left(\|\mathbf{R}\mathbf{p}_\tau\| + \|\mathbf{T}\mathbf{u}_\tau\| \right) \tag{6.13}$$

$$min \;\; J \tag{6.14}$$

$$subject\ to: \quad f_{\min} \preceq \mathbf{f}_\tau \preceq \mathbf{f}_{\max} \;\; \forall\, \tau \tag{6.15}$$

$$\mathbf{x}_{\tau+1} = \mathbf{A}\mathbf{x}_\tau + \mathbf{B}\mathbf{p}_\tau \;\; \forall\, \tau \tag{6.16}$$

$$\tilde{\mathbf{C}}\mathbf{x}_{\tau+1} \preceq \mathbf{t}_{\max} \;\; \forall\, \tau \tag{6.17}$$

$$\mathbf{u}_\tau \succeq \mathbf{0} \;\; \forall\, \tau \tag{6.18}$$

$$\mathbf{u}_\tau = \mathbf{w}_\tau - \mathbf{f}_\tau \;\; \forall\, \tau \tag{6.19}$$

$$\mathbf{l}_\tau \succeq \mu \mathbf{f}_\tau^{\mathbf{2}} \;\; \forall\, \tau \tag{6.20}$$

$$-\mathbf{w} \preceq \mathbf{m}_{\tau+1} - \mathbf{m}_\tau \preceq \mathbf{w} \;\; \forall\, \tau \tag{6.21}$$

$$0 \preceq \mathbf{m}_\tau \preceq \mathbf{1} \;\; \forall\, \tau \tag{6.22}$$

$$\mathbf{p}_\tau = [\mathbf{l}_\tau; \mathbf{m}_\tau] \;\; \forall\, \tau \tag{6.23}$$

where matrices $\mathbf{A}$, $\mathbf{B}$ are related to the overall 3D-MPSoC system description. These matrices represent the 3D-MPSoC system using a coarse granularity of the thermal cells and where the sampling time of the resulting discrete-time system is T_{GC}. We define the horizon of this predictive policy as h [2]. Then, the objective function J is expressed by a sum over the horizon.

In this equation, the first term $\|\mathbf{R}\mathbf{p}_\tau\|$ is the norm of the power input vector p weighted by matrix $\mathbf{R}$. Power consumption is generated here by two main sources: i) the workload setting and ii) the liquid cooling pumping power. Vector p is a vector containing normalized power consumption data the p tiers and the pumping power. Matrix $\mathbf{R}$ contains the maximum value of the power consumption of the tiers (first p diagonal entries) and the cooling system (last entry). The second term $\|\mathbf{T}\mathbf{u}_\tau\|$ is the norm of the required workload, but not yet executed. To this end, the weight matrix $\mathbf{T}$ quantifies the importance that executing the required workload from the scheduler has in the optimization process. Then, Inequality 6.15 defines a range of working frequencies to be used, but this does not prevent from adding in the optimization problem a limitation on the number of allowed frequency values.

Equation 6.16 defines the evolution of the 3D-MPSoC according to the present state and inputs. Equation 6.17 states that temperature constraints should be respected at all times and in all specified locations. Since the system cannot execute jobs that have not arrived, every entry of $\mathbf{u}_\tau$ has to be greater than or equal to 0 as stated by Equation 6.18. The undone work at time τ, u_τ is defined by Equation 6.19. Equation 6.20 defines the relation between the power vector $\mathbf{l}$ and the working frequencies. μ is a technology-dependent constant.

Then, Equations 6.21-6.22 define constraints on the liquid cooling management. The normalized pumping power value ($\mathbf{m}$) scales, and any time instance τ, from 0 (no liquid injection) to 1 (power at the maximum pressure difference

allowable), as shown in Equation 6.22. Moreover, we limit the maximum increment/decrement change in the pumping power value from time (τ) to ($\tau + 1$) by a another normalized value $\mathbf{w}$, as shown in Equation 6.21, which models the mechanical dynamics of the pump. Although we assume one pump in the target 3D-MPSoCs, since we use a vector notation for the pumping power and its constraints, our formulation is valid for multiple pumps as well.

Equation 6.23 defines formally the structure of vector $\mathbf{p}$. Vector $\mathbf{l} \epsilon \Re^{\mathbf{p}}$ is the power input vector, where p is the number of tiers of 3D-MPSoC.

Finally, we formulate the control problem over an interval of h time steps, which starts at current time τ. Therefore, our approach is predictive. Indeed the result of the optimization is an optimal sequence of future control moves (i.e., amount of workload to be executed in average for each tier of the 3D-MPSoC which is stored in vector $\mathbf{f}$). Then, we only apply to the target 3D-MPSoC the first samples of such a sequence; the remaining moves are discarded. Thus, at each next time step, a new optimal control problem based on new temperature measurements and required frequencies is solved over a shifted prediction horizon (e.g., the "receding-horizon" [2] mechanism), which represents a way of transforming an open-loop design methodology into a feedback one,as at every time step the input applied to the process depends on the most recent measurements.

As already presented in Section 5.4.2, this problem can be transformed so that the solution is given by the linear system

$$\mathbf{y_{\tau+1}} = \mathbf{F_j} \begin{bmatrix} \mathbf{x_{\tau+1}} \\ \mathbf{f^2}\tau \\ \mathbf{w_\tau^2} \end{bmatrix} + \mathbf{g_j} \tag{6.24}$$

where $\mathbf{y}$ is the desired solution as a vector containing the workloads and the pump power, matrix $\mathbf{F_j}$ is a suitable matrix, and $\mathbf{g_j}$ a suitable vector defined over subregions of the solution space indexed by j. We refer the reader to [2] and [99] for details. In [99] an approximate computation method of the regions shows a consistent reduction in the number of storage space with a negligible performance loss.

6.4.5 Policy computation: local controllers

The p local controllers are responsible for the thermal management (e.g., DVFS) of the p tiers of the target 3D-MPSoC. Then, for each tier i the local controller sets frequency and voltage for the $q(i)$ frequency islands (cf. Figure 6.11).

In our hierarchical design, the local controller i receives as input the vector $\mathbf{f_{t+1}}$, which is the average frequency at which island i has to run to execute all the workload assigned to it by the global unit. As a second input data, some thermal sensors provide the thermal profile of the 3D-MPSoC island. We assume that the thermal sensors are optimally allocated as shown in previous work [97]. Thus, the impact of thermal sensor quality and allocation on

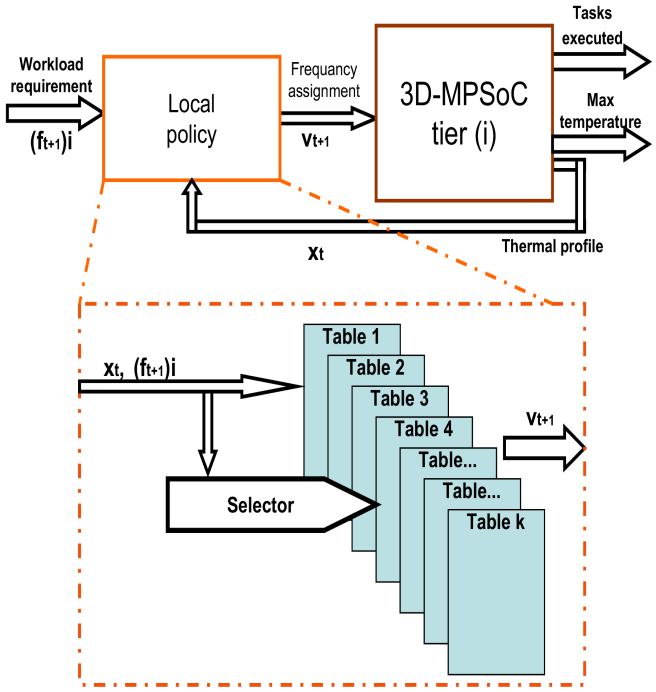

Figure 6.11: Local policy controller block diagram.

the management policy is beyond our scope. The local policy computes the frequencies and voltages for all the $q(i)$ units inside island i, as sketched in the dotted box of Figure 6.11. Input data are used as both computing and selection parameters to choose one of the k functions stored in pre-computed look-up tables.

The local controller decides on the type of optimization to perform: either performance or power-oriented optimization and the related policies are stored in the corresponding tables. Specifically, the control policies optimize power and undone workload. We use an optimization parameter γ that weights these two objectives. At the same time, performance requirement coming from the global controller has to be fulfilled and the maximum temperature constraint satisfied.

The control function is expressed by a policy that is the solution of the following optimization problem:

$$J = \sum_{\tau=1}^{h} \Big(\|\mathbf{Rp}_\tau\| + \gamma \|\mathbf{Tu}_\tau\| \Big) \tag{6.25}$$

$$min \;\; J \tag{6.26}$$

$$subject\ to: \quad 0 \preceq \mathbf{v}_\tau \preceq \mathbf{v}_{\max} \;\; \forall\, \tau \tag{6.27}$$

$$\mathbf{x_{\tau+1}} = \mathbf{Ax}_\tau + \mathbf{Bp}_\tau \;\; \forall\, \tau \tag{6.28}$$

$$\tilde{\mathbf{C}}\mathbf{x_{\tau+1}} \preceq \mathbf{t}_{\max} \;\; \forall\, \tau \tag{6.29}$$

$$\mathbf{u}_\tau \succeq \mathbf{0} \;\; \forall\, \tau \tag{6.30}$$

$$\mathbf{u}_\tau = (\mathbf{f}_\tau)_\mathbf{i} - \sum \mathbf{v}_\tau \;\; \forall\, \tau \tag{6.31}$$

$$\mathbf{p}_\tau \succeq \mu \mathbf{v}_\tau^\mathbf{2} \;\; \forall\, \tau \tag{6.32}$$

where matrices $\mathbf{A}$, $\mathbf{B}$ are related to the thermal modeling of the specific tier that the local controller is supervising. The objective function J expresses the minimization problem by a weighted sum of two terms, in a similar vein as the global policy is computed, except for the tuning parameter γ. Parameter γ changes according to the specific type of optimization criteria for each tier. It ranges from 0 to 1 in steps of 0.1. We set this parameter at run-time based on the maximum temperature recorded according to a heuristic rule: the hotter the thermal profile, the lower γ is, and vice versa. Thus, the controller performs performance-oriented optimization in the case of cold thermal profile, but power saving oriented optimization in case of a hot thermal profile. γ is used at run-time to choose from a set of tables, as shown in Figure 6.11. In the next subsection, we present the generated design space by the parameter γ, and quantify how it significantly affects the power and performance trade-offs of the local policy design.

As in the previous case, this problem can be transformed so that the solution is given by the linear system:

$$\mathbf{y_{\tau+1}} = \mathbf{F_j} \begin{bmatrix} \mathbf{x_{\tau+1}} \\ \mathbf{v^2}\tau \\ \mathbf{f_\tau^2} \end{bmatrix} + \mathbf{g_j} \tag{6.33}$$

where $\mathbf{y}$ is the desired solution as a vector containing the frequencies of the various islands for the tier under consideration.

6.4.6 Policy setup

The *global thermal controller* activation period is $T_{GC} = 1s$, while the local policies are applied every $T_{LC} = 10ms$. The simulation step for the discrete time integration of the RC thermal model has been set to 200μs. The maximum temperature limit is set to $370°K$. The room temperature and fluid temperature (T_{fluid}) are set to $300°K$. In the problem formulation, to establish the relation between the frequency setting and the power consumption,

we use a quadratic relation as in Murali et al. [62]. The time constants needed by the mechanical dynamics of the cooling pumps to go from 0 to maximum power is set to $400ms$.

We vary the parameter γ in the local policies from 0 to 1 with steps of 0.1. Since every value of this parameter is associated with a different look-up table, there are 11 tables in the local controller.

6.4.7 Compared 3D-MPSoC thermal management policies

In our evaluation of the proposed hierarchical management policy thermal and energy efficiency, we implement different state-of-the-art thermal management techniques that we elaborate on them as follows:

- **Liquid cooling with load balancing (LC_LB)** [26, 21] (the default implementation in most operating systems): it applies the maximum flow rate (0.0323 l/min per tier), while balancing the workload by moving threads from a core's queue to another if the difference in queue lengths is over a threshold.

- **LUT-based varying flow rate with TALB (LC_VAR)** [20]: it changes the flow rate (between 0.01-0.0323 l/min per tier) based on the predicted maximum temperature, but the jobs are scheduled with *temperature-aware 3D load balancing* [20].

- **Fuzzy control-based thermal management (LC_FUZZY)** [77]: it uses a run-time fuzzy control to alter the flow rate (between 0.01-0.0323 l/min per tier) and tunes the voltage and frequency values of the processing elements.

In our evaluation, we use the straight microchannel-based cooling layer with all policies, while we use the bent microchannel-based cooling layer with LC_LB and our proposed hierarchical policy.

6.4.8 Results

In our evaluation of different thermal management policies, we compare our proposed policy with respect to the other management techniques mentioned above based on the:

- Maximum and average temperatures.

- Thermal gradients.

- Power consumption and performance degradation.

In the following subsections, we elaborate on each of the aforementioned metrics.

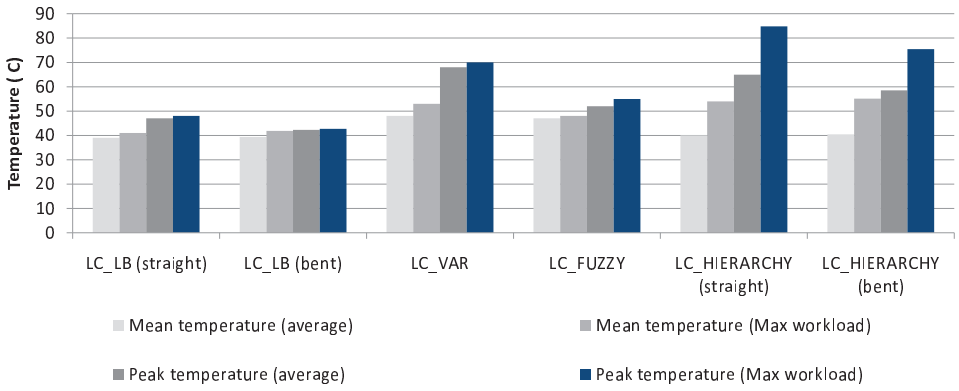

Figure 6.12: Peak and average temperatures observed using all the policies, both for the average case across all workloads and maximum workload on 4-tier 3D-MPSoC.

Maximum and average temperatures

Thermal impact of all the policies the 4-tier 3D-MPSoC is shown in Figure 6.12. In this figure, we show in the peak and average temperature recordings. Interlayer liquid cooling has the ability to absorb the heat flux between different tiers surrounding the cooling layer, regardless the used structure. LC_LB reduces the peak temperature to 47^oC, whereas LC_FUZZY and LC_VAR push the system into a higher peak of 52^oC and 67^oC, respectively, but still avoids any hot-spots. This is the similar case in our proposed hierarchical policy, where the peak temperature reaches 84^oC. The variation from the peak temperature comes from the fact that main target is to reduce the peak temperature to any value below 85^oC. However, since each technique has a different management policy, with different control elements, the peak and average temperatures are affected.

Thermal gradients

We compute thermal gradients in the stack in addition to computing the peak/average temperatures. We calculate the maximum thermal gradient in the whole stack as well as the average intralayer thermal gradient of the different source layers in the stack. We define the thermal gradient threshold by 15^oC, hence the policy objective is to minimize the maximum thermal gradient to any value below 15^oC.

Figure 6.13 and Figure 6.14 show the maximum thermal gradient and the intralayer thermal gradient of the 3D-MPSoC with different management policies. Although interlayer liquid cooling diminishes the thermal hot-spots, it increases both the intralayer and the maximum thermal gradient of the stack. This is based on the fact that the fluid grows thermally from the inlet to the outlet such that the elements near the inlet have more heat removed than

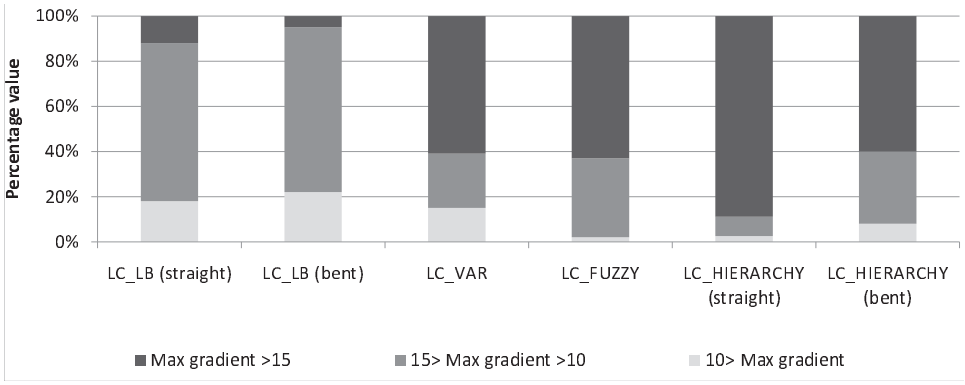

Figure 6.13: Maximum thermal gradient of the whole 3D-MPSoC stack, using the average case of all workloads.

the ones at the outlet. Moreover, varying the flow rate, as in LC_FUZZY and LC_VAR, increases the thermal gradient, since reducing the flow rate increases the thermal gradient of the system.

Our hierarchical policy manages to reduce intralayer thermal gradients below 10^oC per layer. This is due to that fact that the local controllers distributed the assigned workload among the controlled elements, taking into consideration their thermal state. Thus, elements with lower temperature gets more load, while high temperature elements are assigned lower workload.

While using straight microchannels has an impact on the thermal gradient, the usage of bent microchannels aids in diminishing the maximum thermal gradient. The peak thermal gradient in LC_LB with bent channels, is reduced, by 58% with respect to using the same policy with straight channels. This enhancement is based on the fact that there is more fluid pumped to the bent structure than the straight structure. Moreover, the fluid path is in the bent channels is relatively shorter than that of straight channels. Thus, the fluid thermal growth is lower in the bent channel case. Furthermore, the use of bent channels with our policy aids in reducing the maximum gradient by an additional 32% with respect to using straight channels.

System and cooling power consumption

Figure 6.15 shows the total consumed power when running the various policies on the 4-tier MPSoC for the average workload. Energy consumption values are normalized with respect to the load balancing policy on a system with LC_LB. In this figure, we show that our proposed policy manages to reduce the cooling power and the overall system power by 60% and 23%, respectively with respect to LC_LB. Moreover, our policy even reduced the cooling energy more than LC_VAR and LC_FUZZY by 40% and 22%, respectively.

When the bent channels are used with LC_LB, the pumping power consumed is higher than the case with straight channels. This is based on the same fact that more fluid is pumped in this case, hence more power is needed.

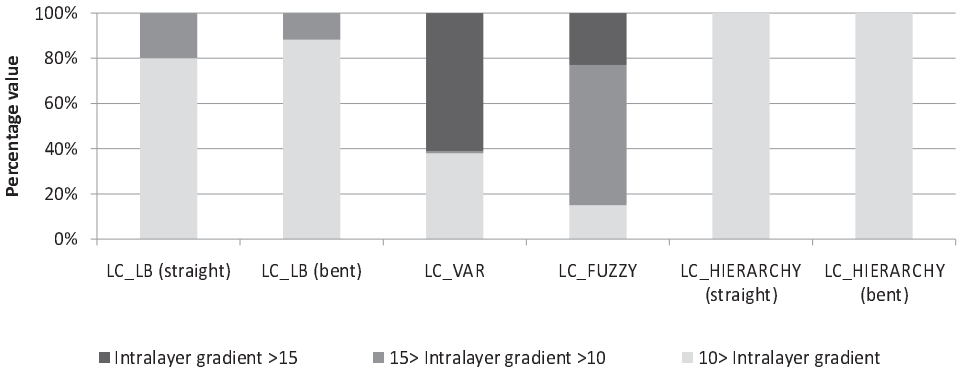

Figure 6.14: Average intralayer thermal gradient of the whole 3D-MPSoC stack, using the average case of all workloads.

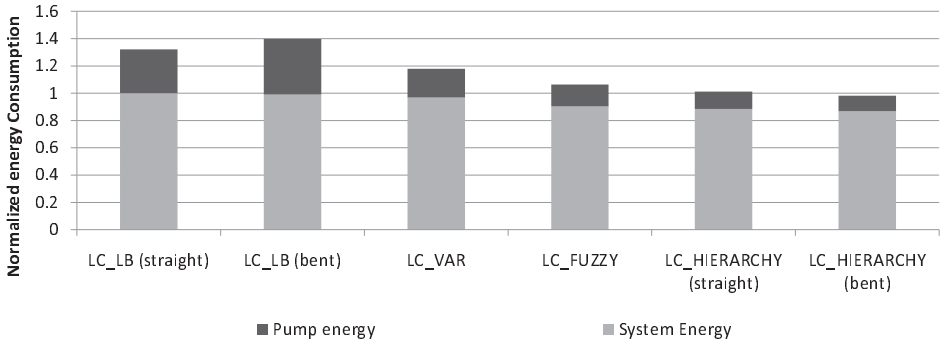

Figure 6.15: The normalized energy consumption in the whole system (chip and cooling network) averaged per stack.

However, when the bent channels are used, our policy does not apply the maximum flow rate since the objective goal is achievable with lower flow rates. Thus, the consumed pumping power in the bent channel case is of the same order as the case with straight channels.

6.5 Summary

This chapter has presented two main contributions: a centralized and a distributed thermal management algorithm for 3D MPSoC. These techniques use liquid cooling technologies applied to 3D-MPSoCs.

The first contribution of this chapter is a novel centralized thermal management approach for 3D stacks that controls both DVFS and a variable-flow liquid cooling using convex optimization to meet the desired performance and minimal energy requirements. The optimization problem considers the thermal profile of the system, its evolution over time and current time-varying workload requirements. Our experimental results illustrate that our policy satisfies performance requirements, maintains the temperature below the spec-

ified threshold, while reducing cooling energy by up to 50%. The policy also keeps the thermal profile approximately $18°C$ lower compared with state of the art polices using liquid cooling.

The second contribution of this chapter is an online hierarchical thermal management policy for high-performance 3D systems with liquid cooling. The proposed controller has an innovative hierarchical structure that enables the policy to be both effective in terms of achieving its performance requirements and simple in terms of hardware implementation. Moreover, the hierarchical structure of the policy enables the thermal management system to be easily scalable to any 3D systems. The optimization problem is executed and it considers the thermal profile of the system, its evolution over time and current time-varying workload requirements. The implementation is done using look-up tables. Results show significant advantages in terms of energy savings that reach values up to 50% with respect to state-of-the-art thermal control techniques for 3D stacks with liquid cooling, and a thermal balance with differences of less than 10^oC per layer.

7 Sensor Placement

Any thermal management algorithm to control the system must know the overall state (or thermal profile) of the MPSoC. This means that the temperature of every single cell in which the floorplan has been divided must be known. Thermal sensors are silicon devices able to measure the temperature of the material surrounding them, that is indeed silicon.

In this chapter we present three approaches to estimate the thermal profile from sensors located in some specific locations on the MPSoC floorplan.

7.1 Thermal Profile Estimation

7.1.1 Temperature estimation by sensing devices

This approach derives the thermal profile by means of many sensing devices placed on the silicon layer. It can be proved both theoretically and experimentally that with a small approximation error (of the order of few percent) the temperature of a certain copper cell can be considered similar to the one located in the same position of the silicon layer. The derivation is presented here by working on the different heat propagation dynamic between the cell and its neighbors.

Silicon layer

The cell S_{i-0} transfers heat to cells of the same layer with a rate depending on the constant K_{si-si}. The heat transfer between cell S_{i-0} and C_{i-0} depends on K_{si-cu}. The ratio between these two constants expresses the dominance relation between the two mechanism of heat transmission. By analyzing the RC model behind Figure 3.1, we have that:

$$\frac{K_{si-si}}{K_{si-cu}} = \frac{1}{2} \cdot \left(\frac{h_{si}}{l}\right)^2 \tag{7.1}$$

where h_{si} is the height of the silicon cell and l is its width. By assuming $h_{si} << l$ (i.e. a factor of 10 using a cell size of 3mm as shown in the experimental setup chapter) we have that the dominant dynamic is the one that exchange the heat between the silicon and the copper layer.

Copper layer

We now analyze the ratio between K_{cu-si} and two other constants K_{cu-cu} and K_{cu-RT}. The first one is expressed by the following equation:

$$\frac{K_{cu-cu}}{K_{cu-si}} = \frac{1}{2} \cdot \frac{K_{cu} \cdot h_{cu} \cdot h_{si}}{K_{si} \cdot l^2} \tag{7.2}$$

where h_{si} is the height of the silicon cell and l is its width. h_{cu} is the height of the copper cell. K_{cu} and K_{si} are respectively copper and silicon thermal conductivity coefficients. Because the last two coefficient have the same order of magnitude, by assuming $h_{si} << l$ and $h_{cu} << l$ (from the experimental setup chapter: $h_{si} = 300\mu m, h_{cu} = 850\mu m, l = 3000\mu m$) we have that the dominant dynamic is the one exchanging heat with S_{i-0}.

The second one is expressed by:

$$\frac{K_{cu-RT}}{K_{cu-si}} = \frac{K_{cu} \cdot h_{si} \cdot S_{pck}}{K_{si} \cdot (K_{cu} \cdot l^2 + S_{pck})} \tag{7.3}$$

where h_{si} is the height of the silicon cell and l is its width. S_{pck} is the environmental superficial conductance of each cell composing the copper layer. K_{cu}

and K_{si} are respectively copper and silicon thermal conductivity coefficients. From Equation 7.3, if we assume that $h_{si} >> K_{si}/K_{cu}$ (true for $h_{si} > 1\mu m$) and that $l >> \sqrt{S_{pck} \cdot h_{si}/K_{si}}$ (from the experimental setup chapter: for $S_{pck} = 16.67m, thanl > 133\mu m$, true in our case since $l = 3000\mu m$), we have that $K_{cu-RT} << K_{cu-si}$.

When all previous assumptions hold, in the heat exchange process (Figure 3.1), the most important dynamics is the one between cells S_{i-0} and C_{u-0}. This means that temperature difference between a silicon cell and the copper cell above it is negligible compared with temperature differences between these cells and their respective neighbors. The most important consequence of previous statement is that the copper layer thermal profile can be assumed to be like the silicon one with a small error. This way, a full state estimation can be performed by simply measuring temperatures on the silicon layer.

Simulation Results

The plot of Figure 7.1 shows the percentage error between every cell of the silicon layer and the one of the copper layer on it normalized to the difference between the silicon temperature and the ambient one ($300°C$).

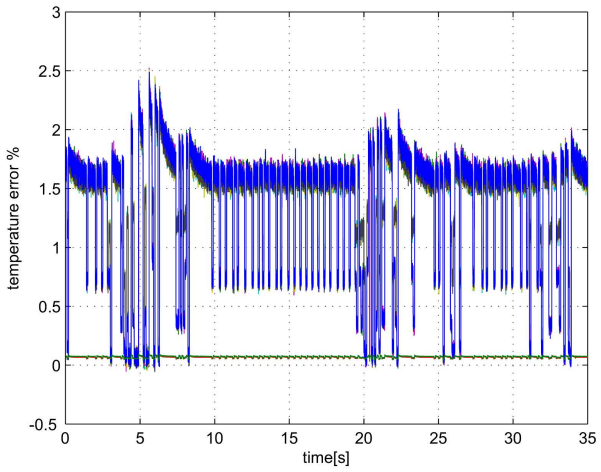

Figure 7.1: Percentage error between every cell of the silicon layer and the one of the copper layer on it normalized to the difference between the silicon temperature and the ambient one ($300°C$).

Simulation results show that the temperature is always less than 2.5 % and the approximation error is around 1.5% for most of the time. This error is the experimental validation of the approximation explained in Section 7.1.

Another possible source of inaccuracy is represented by the presence of noise affecting measurements done by thermal sensors. This noise is called measurement noise. The following simulation shows the amplitude of this noise and the impact that it has on the estimation of the thermal profile.

The plot of Figure 7.2 compares the maximum chip temperature with the one estimated by thermal sensors measurements. As it can be noted, errors up to

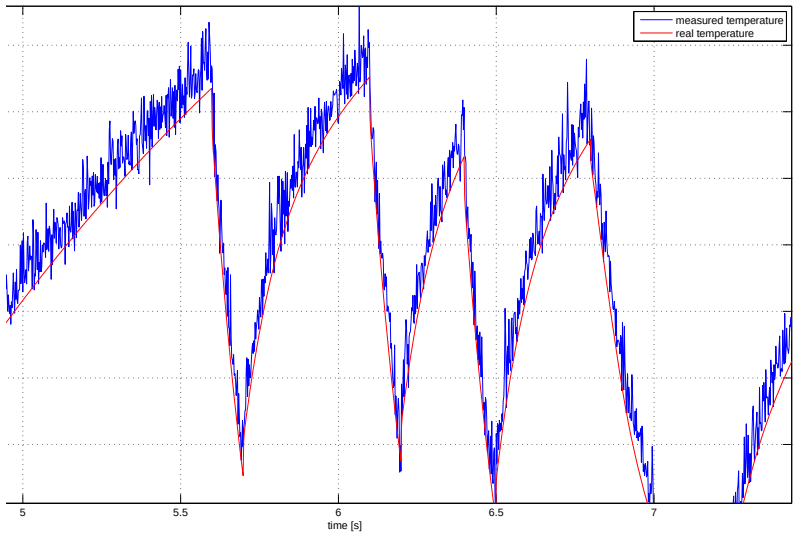

Figure 7.2: Maximum chip temperature estimation (in Kelvin degrees): real versus thermal sensors.

$2°C$ can be made using thermal sensors. The method presented in following section will solve this problem.

7.1.2 Temperature estimation by observability

A better approach to estimate these temperatures is to use a state estimator [33]. A state or thermal profile estimator is an algorithm able to derive the current thermal profile based on measurements in some specific locations on the chip with a specific rate. This method enables a reduction in the number of sensors and also the sensors sampling noise. Equation 7.4 models the temperature measurement process.

$$\tilde{\mathbf{t}}_\tau = \mathbf{C}\mathbf{t}_\tau \tag{7.4}$$

The output $\tilde{\mathbf{t}}_\tau \; \epsilon \; \Re^s$ of our system is the temperature observed by the s on-chip thermal sensors placed in the silicon layer. Matrix $C \; \epsilon \; \mathcal{B}^{s \times n}$, $\mathcal{B} = \{0, 1\}$, represents a selection matrix that models the placement of a sensor on the silicon die. Since we are assuming to have distinct measurements coming from distinct sensors, $\mathbf{C}$ has only 1 nonzero element per row and each column can have at most 1 nonzero element. Figure 7.3 shows a graphical representation of selection matrix $\mathbf{C}$ for the case study floorplan described in the experimental setup chapter. Cells of the floorplan are numbered starting from the bottom left corner and ending on the top right corner of the floorplan. Namely $c_{i,j}$

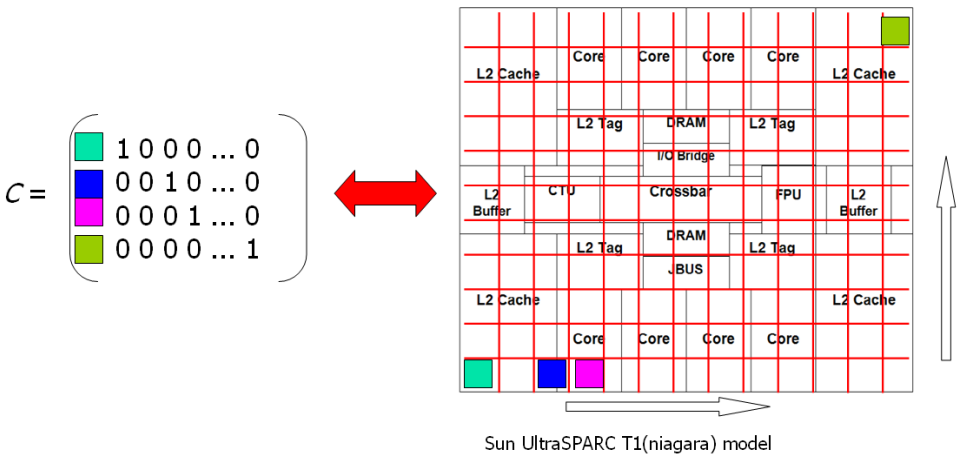

Figure 7.3: Graphical representation of selection matrix $\mathbf{C}$ for the case study floorplan described in the experimental setup chapter.

is equal to 1 if thermal sensor i is located inside the cell j. For technological reasons thermal sensors can be placed only on the silicon layer.

The parameter that measures how much a system is observable is called observability. Observability refers to the property of a system that enables the reconstruction of the state variables given the inputs [33]. It means that we are able to reconstruct completely the thermal profile of the chip given the inputs only by looking at the measurements coming from the sensors, placed in locations specified by the matrix $\mathbf{C}$. This means that we are assuming to have in the output vector s distinct temperature measurements coming from s distinct cells. The rank of the observability matrix $\mathbf{Q}$ expresses the number of states that can be reconstructed from the measurement vector $\tilde{\mathbf{t}}_\tau$. The observability matrix $\mathbf{Q}$ is expressed by the following equation (see [33]):

$$\mathbf{Q} = [\mathbf{C}; \mathbf{CA}; \ldots; \mathbf{CA^{n-1}}] \tag{7.5}$$

If matrix $\mathbf{Q}$ is full-rank, the state vector can be reconstructed completely from the measurements, the input vector $\mathbf{p}$ and matrices $\mathbf{A}$ and $\mathbf{B}$ identifying the thermal dynamics of the system (see Equation 3.2).

The problem of selecting the right placement of thermal sensors to both minimize the number of sensors and maximize observability is the problem of choosing the matrix $\mathbf{C}$ with the minimum number of rows that maximize the rank of the observability matrix $\mathbf{Q}$. Given an MPSoC model, this problem depends on the location and the number of sensors inside floorplan (matrix C). The sensor sampling frequency (f) has as well an influence on the observability. In next paragraphs we propose a placement algorithm to deal with just mentioned problem.

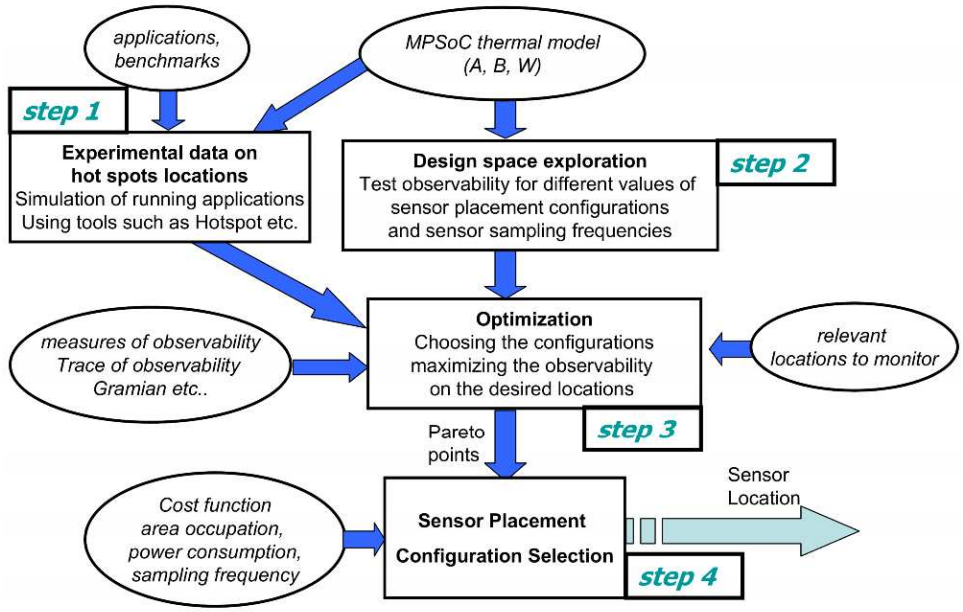

Figure 7.4: Proposed method block diagram

7.2 Full Model Placement Algorithm

This methodology is based on a complete design space exploration of all possible sensor placements. Then, according to user defined constraints, the configuration that maximizes the observability of the system is chosen. This approach is a method performing a complete design exploration on the full model without any model order reduction techniques.

7.2.1 Methodology

The block diagram of the proposed procedure is presented in Figure 7.4. The method is consists of four steps.

In the first step, experimental data of hot spots locations are recorded during the runtime execution of the system. Data can be obtained from real chip temperature measurements or from simulations using tools such as Hotspot. These data will point out in which locations an accurate monitoring is needed in order to identify the rising of potential hotspots. It is important to notice that these data are not used to define the placement for the sensors. They are used by the algorithm as a criterion to rank among different sensor placements having the same observability properties.

In the second step the design space exploration is done on all the possible sensor placement configurations. First the model is sampled using a frequency f that ranges from F_{min} to F_{max}. After that, the number of sensors employed in the placement is varied from 1 to n. A value of 1 means having only 1 sensor in the whole MPSoC floorplan. At this stage for every value of s and

f, a total of $\binom{n}{s}$ sensor placement configurations are generated. The possible configurations have only one sensor per cell. This leads to have a matrix C having one nonzero element per row and a total of s rows. The rank of the observability matrix Q is computed for each configuration. In the case of the experimental setup we are considering described in Section 5.2, the floorplan consists of 24 cells and the overall computational effort requested by this approach is feasible. For larger floorplans the model order reduction method proposed in Section 7.3 may result more appropriate.

The third step performs an optimization based on data collected on both previous steps plus some additional data. This step performs a selection of the number of all analyzed sensor placement configurations. First, configurations that do not allow the estimation of area of the chip that are relevant to the designer are discarded. If a full profile estimation is needed by the designer, then, placements leading to observability matrices with rank less than $2n$ are discarded. As a second criterion, configurations where sensors are placed on experimental hot-spots locations (see step 1 of the algorithm) are preferred to other ones. Remaining placements are ranked according to metrics to measure the observability of a system (i.e.the observability Gramian [86]-[10]). Finally according to aforementioned metrics, Pareto point placements are computed. A specific sensor placement is corresponding to every Pareto point in the plane sensor number s versus sensor sampling frequency f.

The last step selects the best placement according to the designer defined criteria based on area occupation(related to s), power consumption(related to s and f) and sensor sampling frequency (related to f).

7.2.2 Placement results

We applied the proposed algorithm to the case study described in Section 5.2. The overall computation of the proposed algorithm on a INTEL Core^{TM}2 duo laptop having a frequency of 2GHz (T7200) in the case of a modelling performed using 24 states took 3.44 minutes. After the first step, we obtain the design space exploration results of all possible sensor placement configurations. By plotting the percentage of the observable states over the overall states of the MPSoC versus the number of sensors used and their sampling frequency, we obtain the plot of Figure 7.5. As it can be noted from the graph, for a fixed percentage of the observable states, there are many options. The graph shows that the number of sensors can be reduced by increasing the sensors sampling frequency and vice-versa. It is important to notice also that there are many possible sensor placement configurations associated with any point in previous graph.

At this stage, among all possible placement configurations we identify Pareto points inside the design space. Trade-offs are between the number of sensors and their sampling frequency. The reason is because for a given observability target, the lower bound of thermal sensors employed depends on the thermal sensor sampling frequency (see Figure 7.5). According to sim-

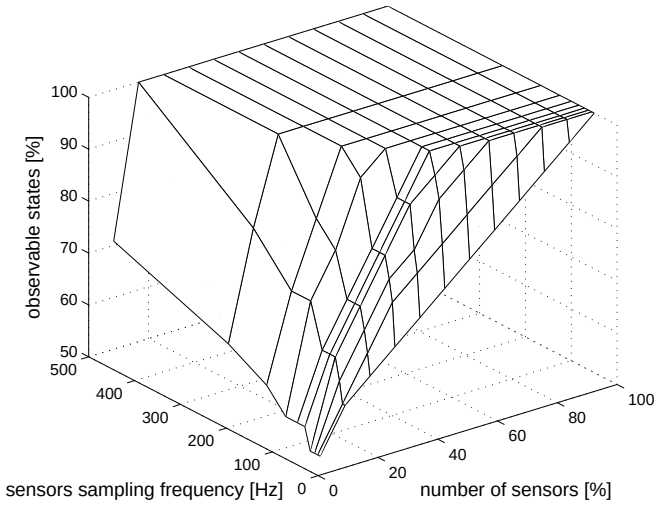

Figure 7.5: Design space exploration of the case study (step 2)

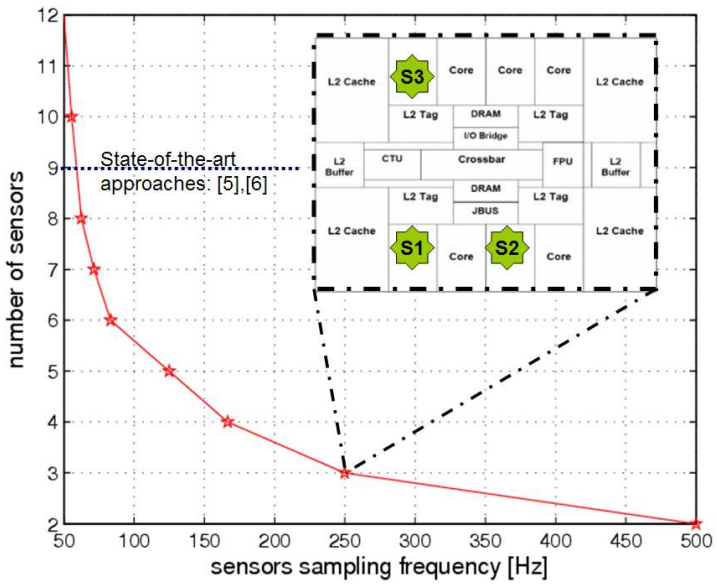

Figure 7.6: Pareto points (steps 3+4) and comparison with [5],[6].

ulations and results from [24] and [43], in our case study, critical areas for hotspots are the cores and the crossbar located in the central part of the chip. Moreover, we are interested in monitoring 100% of the states of our MPSoC. In this case study, the overall Pareto points computation step takes few seconds. The graph identifying the resulting Pareto points in the plane sensors number versus sensors sampling frequency is shown in Figure 7.6.

In the last step of the proposed methodology (see Figure 7.4), we assume a maximum of 3 thermal sensors as possible design constraint on the MPSoC.

Moreover we want to have a sensor sampling frequency as low as possible. According to Figure 7.6, the corresponding Pareto point is a 3 sensor configuration with a sampling frequency of $250Hz$. This means that if we want to make the system observable with only 3 sensors, we need to sample them every $4ms$ and we need to place them in a specific configuration. This specific placement is shown in Figure 7.6. This sensor configuration supports a complete estimation of the system and so a complete reconstruction of the thermal profile of the MPSoC. This operation can be implemented by using conventional estimation techniques (i.e. Kalman filter [33]). This will also correct for potential noise sources present in thermal sensors.

7.2.3 Comparison with state-of-the-art methods

We compare the method with the approaches proposed by Memik et al. [56] and Sharifi and Rosing [80]. The algorithm proposed by Memik et al. [56] finds the sensor placements that provides the best estimation accuracy for a certain number of sensors, according to experimental hot-spots locations. The algorithm is based on an interpolation technique based on experimental data derived from simulations. The algorithm proposed by Sharifi and Rosing [80] minimizes the temperature differences among the hot-spot temperature and the one detected by the thermal sensor. Both these algorithms are based on techniques trying to catch hot spots by using the minimum number possible of sensors for a certain accuracy.

Our method is not targeting hot spots but the observability of the system. Once the system is observable, hot spots are automatically detected. The reason is because the portion of the thermal profile of the chip that is relevant to the designer can be completely reconstructed from sensors measurements. The advantage is a strong reduction in the number of required sensors. In our case study, the cores may run with independent frequencies. This imply that hot spots are uncorrelated from core to core. Consequently, according to the works proposed by Memik et al. [56] and Sharifi and Rosing [80] at least one sensor per independent unit is required to monitor those elements and detect potential hot spots. Thus, a minimum of 9 sensors (8 sensors for the cores plus 1 sensor for the crossbar) are needed by both techniques to detect all possible hot spots. Conversely our method needs only 3 sensors, i.e. a reduction of 3×. Moreover Figure 7.6 shows that the gain can reach 4.5× at a sampling frequency of 500Hz. The proposed method enables the full thermal profile estimation of the MPSoC by using a number of sensors that is smaller than the number of the location of potential hot-spots formation regions.

7.3 Model Order Reduction Placement Algorithm

This method finds the sensor placement configuration that maximizes the observability of the system with a greedy algorithm. The algorithm is based on a model order reduction performed on the thermal model of the MPSoC.

7.3.1 Introduction

In the model described by Equations 3.2 and 7.4, a state is required for every block of the floorplan, because we need n states to store n temperatures values. This requirement is expensive in terms of computational cost. The higher the number of states modelling the MPSoC is, the higher the number of sensors required for its state estimation is. This could be a problem in case of a detailed model of a complex 3D-MPSoC including liquid cooling.

This approach finds the best locations inside the 3D-MPSoC where thermal sensors can be placed using a greedy technique. The advantage is an efficient method to solve both the sensor placement and the model order reduction of the MPSoC. Complex MPSoC models are indeed reduced in complexity with an inaccuracy in the order of few percents. As a result the thermal model is simpler that the original one and the computational cost of the thermal management algorithm will be reduced.

The block diagram of the proposed algorithm is presented in Figure 7.7. The proposed methodology consists of two phases: a design-time phase and a run-time phase.

During the design phase the thermal management system model is defined. The reduced order MPSoC thermal model and the sensor placement are the outputs of this off-line phase. The concept behind the proposed sensor placement technique is based on an analysis of the the balanced state-space realization of the 3D-MPSoC system and its Hankel singular values decay rate (see [46], [33] for more details). The number of states of the reduced order model is fixed according to user designer accuracy requirements, and a specific location is assigned to each sensor.

During the run-time phase the reduced order system state vector $\mathbf{x}$ is estimated thanks to a simple state estimator (i.e. Kalman filter) and measurements coming from thermal sensors. Then, this information is used by the thermal model to perform the optimization on the reduced-order 3D-MPSoC model pre-defined in the design-time phase.

7.3.2 Model: from structure-centric to energy-centric

First, an accurate 3D-MPSoC thermal model is created according to the model presented in previous section. This will determine matrices $\mathbf{A}$ and $\mathbf{B}$ according to Equation 7.6. We assume here that all temperatures are offsets from room

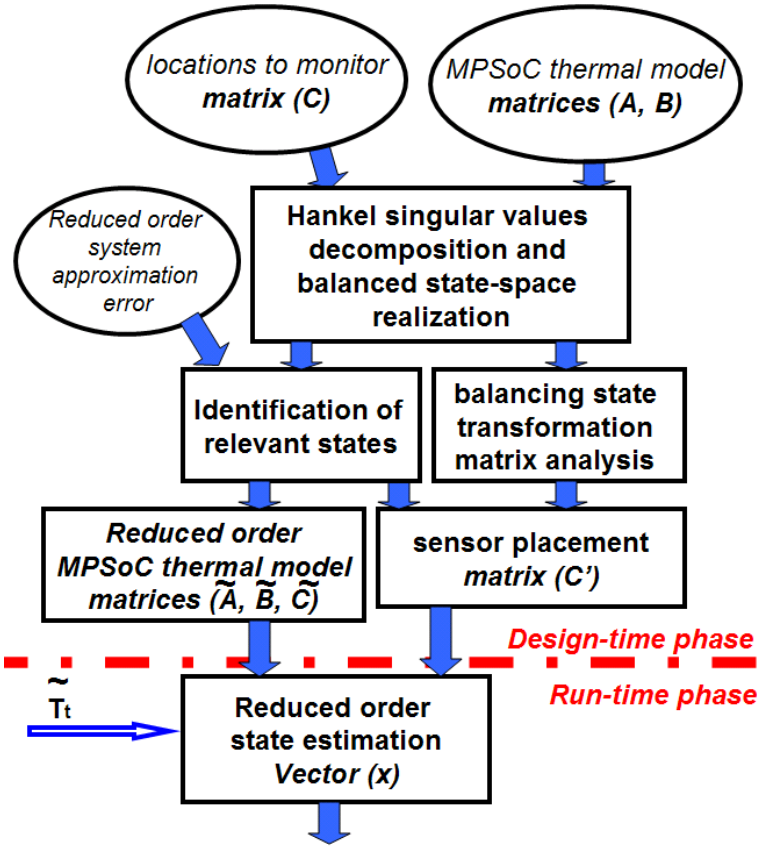

Figure 7.7: Proposed method block diagram

temperature. This way we can omit vector $\mathbf{w}$ from Equation 3.2.

$$\mathbf{t_{\tau+1}} = \mathbf{A}\mathbf{t}_\tau + \mathbf{B}\mathbf{p}_\tau \tag{7.6}$$

Locations that the policy needs to monitor to ensure safe working conditions are determined by the following relation:

$$\tilde{\mathbf{t}}_\tau = \mathbf{C}\mathbf{t}_\tau \tag{7.7}$$

Equation 7.7 describes the choice of relevant locations to monitor inside the MPSoC. Matrix $\mathbf{C}\ \epsilon\ \mathcal{B} = \{\mathbf{0},\mathbf{1}\}^{\mathbf{s}\times\mathbf{n}}$ is a selection matrix. In this model we assume that we want to control locations on the silicon layer of each tier. We do this to ensure a full MPSoC temperature control in every location containing an active device on the silicon layer. We assume that s is the total number of those locations. Namely $c_{i,j}$ is equal to 1 if thermal sensor i is located inside the cell j.

In the case study of the 3D-MPSoC described in Section 6.2, the number of states that is also the the number of temperature values for each cell composing the 3D-MPSoC floorplan is 200. This means that the number of cells composing the silicon layers as well as the ones composing the copper layer is 100. Thus matrix $\mathbf{A}$ consists of 10^4 entries and matrix $\mathbf{C}$ has 100 rows.

To determine the states with negligible contribution to the MPSoC thermal dynamics, the system is balanced using a Gramian-based balancing of state-space realizations [46]. This technique computes a balanced state-space realization for the stable portion of the system. For stable systems, the output is an equivalent system for which the controllability and observability Gramians are equal and diagonal, their diagonal entries forming the vector $\mathbf{g} \epsilon \Re^{\mathbf{n}}$ of Hankel singular values. These values provide a measure of energy for each state in the system. If the corresponding Hankel singular value for a certain state is a relatively small number, this means that that state has a small influence in the dynamic of the system. The second output of the Gramian-based balancing [46] is the balancing state transformation matrix $\mathbf{T} \epsilon \Re^{\mathbf{n}\times\mathbf{n}}$ that converts the original system into the balanced one.

The rationale behind this operation is to change the 3D-MPSoC thermal model system perspective. The original model belongs to a geometric and physical view of the 3D-MPSoC where states are related to physical properties. The new model generated by the Gramian-based transformation is energy centric and every states is a heat propagation dynamic. This representation emphasizes how much the thermal dynamic represented by that specific state is relevant to the heat propagation response of the system. The i^{th} row of the conversion matrix $\mathbf{T}$ describes the contribution that the temperature of each thermal cell in the original model gives to the i^{th} most important (in terms of energy) thermal dynamic of the new generated system.

7.3.3 Identification of relevant states

Here we elaborate information related to Hankel singular values vector $\mathbf{g}$. Figure 7.8 shows the state energy distribution for our case study. As Figure 7.8 shows, the energy magnitude drops quite fast and most of the states gives almost negligible contributions to the input-output response of the system. To define a threshold level to distinguish between relevant and not relevant states, we look at the rate of decay of the state energy.

Figure 7.8 shows the decay rate for the normalized energy related to Hankel singular values for our case study. Red arrows points to change in the rate of the decay rate. To identify transition points we look at peaks in the third derivative of the function defined by vector $\mathbf{g}$. In Figure 7.8 they are highlighted with red circles. All these points represent a set of possible threshold point to distinguish between relevant states and negligible states. This means that by adding points after these transition points the advantage of adding states would be smaller in terms of reducing the approximation error.

Figure 7.9 shows the model approximation error in percentage versus number of states in the reduced model for our case study. As Figure 7.9 shows, the decay rate is fast. It is important to notice that only threshold points have been considered in this plot. They are marked with '*'. Results show that an approximation error of $6 \cdot 10^{-2}$ in percentage can be achieved with only 20 states and 180 can be easily discarded with a reduction factor of $10\times$

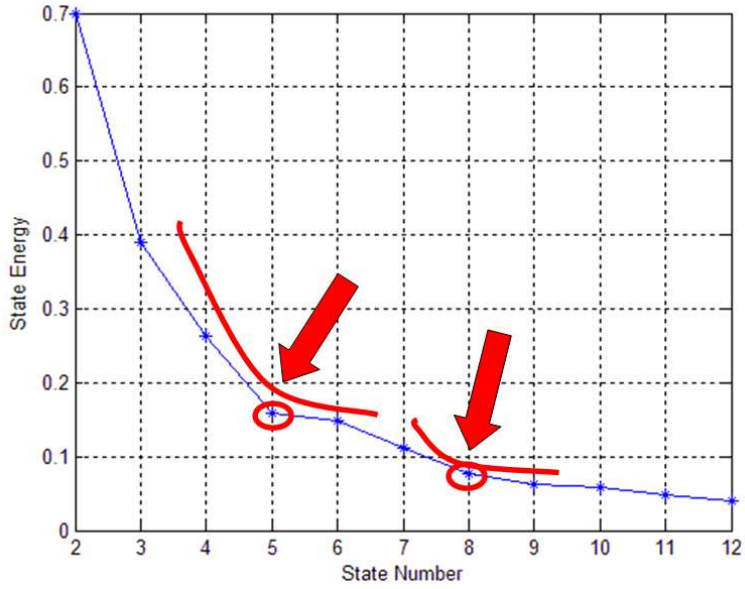

Figure 7.8: Decay rate analysis for the normalized energy related to Hankel singular values for our case study. Red arrows points to change in the decay rate.

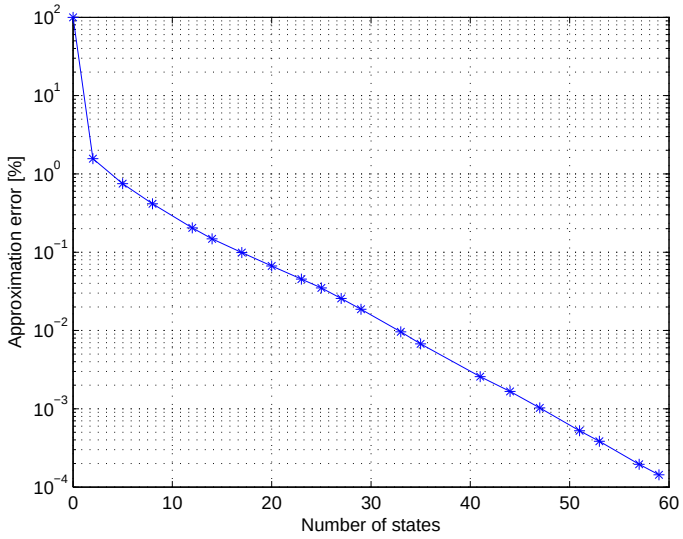

Figure 7.9: Model approximation error [%] versus number of states in the reduced model for our case study.

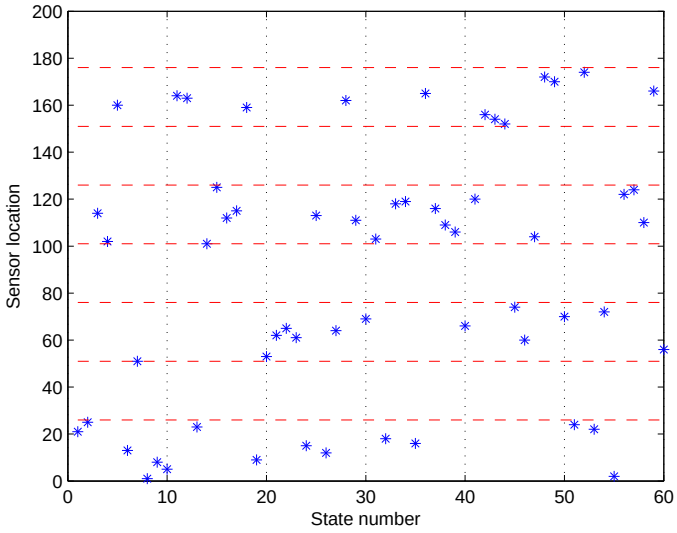

Figure 7.10: Sensor location according to the most relevant component identifying each state of the new thermal model. Horizontal lines delimits one layer from another

for the given accuracy. It is also important to notice that it does not make much sense to go for higher accuracies because inaccuracies in the silicon, in the power model or in thermal sensors will add uncertainty in the results.

7.3.4 Balanced state transformation analysis

Here we elaborate information related to conversion matrix $\mathbf{T}$. The i^{th} row of $\mathbf{T}$ describes the contribution that the temperature of each thermal cell in the original model gives to the i^{th} most important (in terms of energy) thermal dynamics of the new generated system. For this reason at this stage for each row i of $\mathbf{T}$, we identify the most relevant component in absolute value. We call this component j. This means that if we place a sensor in the j^{th} cell in the original model, among all the possible sensor locations, this position would be the one that will contribute more in terms of energy to the i^{th} most important thermal dynamic of the new generated system.

Figure 7.10 shows the most relevant component identifying each state of the new thermal model. Horizontal lines delimit one layer from another. They are placed at multiples of 25 because each layer according to our case study consists of 25 cells. As Figure 7.10 shows, there are no sensors in the copper layer. The reason is because we decided to add this technological limitation as a constraint in the definition of the most relevant sensor location for each state.

7.3.5 Reduced order model and sensor placement

At this stage the user-defined parameter that is missing to complete the sensor placement is the desired accuracy of the reduced order model. If we accept an approximation error of $6 \cdot 10^{-2}$ in percentage, we fix the number of states to 20. By doing this operation we reduce by a factor of 10 the number of states in the model and so the computational complexity of Equation 7.6.

At this point a new reduced order model is obtained from the original one after the balancing using a Gramian-based balancing of state-space realizations [46]. States corresponding to Hankel singular values smaller than a predefined threshold (in our case we selected the 20^{th}) are discarded. Thus the full MPSoC thermal model is now described by the following system of equations:

$$\mathbf{x}_{\tau+1} = \tilde{\mathbf{A}}\mathbf{x}_{\tau} + \tilde{\mathbf{B}}\mathbf{p}_{\tau} \tag{7.8}$$

$$\tilde{\mathbf{t}_{\tau}} = \tilde{\mathbf{C}}\mathbf{x}_{\tau} \tag{7.9}$$

Where matrix $\tilde{\mathbf{A}} \; \epsilon \; \Re^{l \times l}$ and matrix $\tilde{\mathbf{B}} \; \epsilon \; \Re^{l \times p}$. The number of states of the new thermal model is l and p is the number of inputs in the MPSoC model. Equation 7.8 describes the state update for the reduced order model of the MPSoC. This equation is analogous to Equation 7.6. The only difference is that, in this case, the states do not represent directly temperature values inside each cell. Matrix $\tilde{\mathbf{C}} \; \epsilon \; \Re^{s \times l}$ in Equation 7.9 relates the value of the states to temperature in s specific locations (every cell in all silicon layers) inside the MPSoC. This equation is analogous to Equation 7.7 and describes how the temperature measurements can be derived from the state vector $\mathbf{x}$. In our case we were interested in knowing all the cells temperatures of every silicon layer, for this reason $s = 100$.

The purpose of sensor placement is to get reliable information on the 3D-MPSoC thermal profile. The reason is because every time any policy is applied, it operates on reliable thermal profile temperature values. The key for this is to obtain the state vector $\mathbf{x}$. In Section 7.3.3, the balancing state transformation matrix $\mathbf{T}$ converts the original system into the balanced one. Thanks to this matrix, to obtain the estimate of the reduced state vector $\mathbf{x}$, it is sufficient to multiply the thermal profile by matrix $\mathbf{T}$.

For the system identified by Equation 7.6, it means that we are able to reconstruct completely the thermal profile of the chip given the inputs only by looking at the measurements coming from the sensors, placed in locations specified by the matrix C'.

$$\tilde{\mathbf{t}_{\tau}} = \mathbf{C}'\mathbf{t}_{\tau} \tag{7.10}$$

Matrix $\mathbf{C}' \; \epsilon \; \Re^{\mathbf{s}' \times \mathbf{l}}$ in Equation 7.10 is a selection matrix that describes the sensor placement inside the 3D-MPSoC. This means that we are assuming to have in the output vector s' distinct temperature measurements coming from s' distinct cells every T_s seconds where T_s is the sensors sampling period. The rank of the observability matrix $\mathbf{Q}$ expresses the number of states that can be

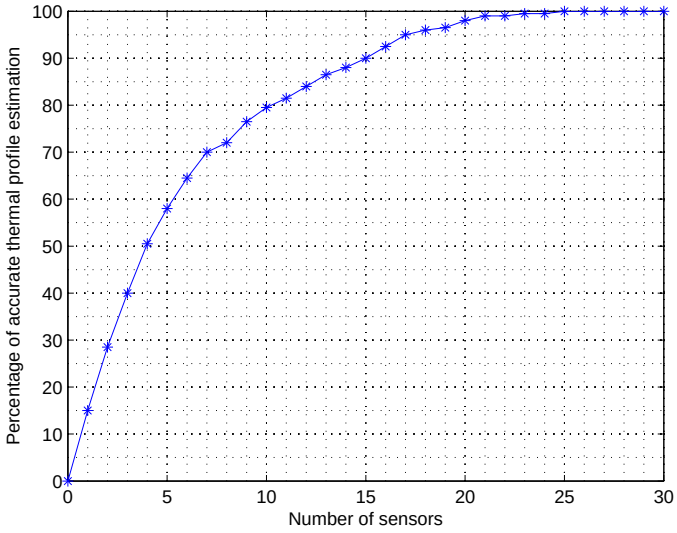

Figure 7.11: Sensor placement algorithm: percentage of accurate temperature estimation according to the number of sensors placed with the proposed methodology

reconstructed from the measurement vector $\tilde{\mathbf{t}_\tau}$. The observability matrix $\mathbf{Q}$ is expressed by the following equation (see [33]):

$$\mathbf{Q} = [\mathbf{C}'; \mathbf{C}'\mathbf{A}; \ldots; \mathbf{C}'\mathbf{A}^{\mathbf{n-1}}] \tag{7.11}$$

If the rank of matrix $\mathbf{Q}$ equals n, the state vector $\mathbf{x}$ can be reconstructed completely from the measurements, the input vector $\mathbf{p}$ and matrices $\mathbf{A}$ and $\mathbf{B}$ identifying the thermal dynamics of the system (see Equation 7.6).

The problem of selecting the right placement of thermal sensors to both minimize the number of sensors and maximize observability is the problem of choosing the matrix $\mathbf{C}'$ with the minimum number of rows that makes the rank of the observability matrix $\mathbf{Q}$ equal n. Given an MPSoC model, this problem depends on the location and the number of sensors inside floorplan (matrix $\mathbf{C}'$) and the sensor sampling period T_s.

To choose the sensor placement we used the information about the locations that contribute most to each of the states in the balanced model. The algorithm is a greedy technique that adds a sensor position according to the placement suggested in Section 7.3.4. Figure 7.10 shows the most relevant component identifying each state of the new thermal model and the algorithm starts from the most relevant state (state number 1) and goes on adding sensors until the rank of the observability matrix equals the rank of $\mathbf{A}$. Figure 7.11 shows the percentage of states that can be estimated versus the number of sensors placed. As it can be noted, for each sensor that is placed as suggested by Figure 7.10, there is an increase in the number of states of the reduced order model that can be estimated. The complete estimation is achieved with

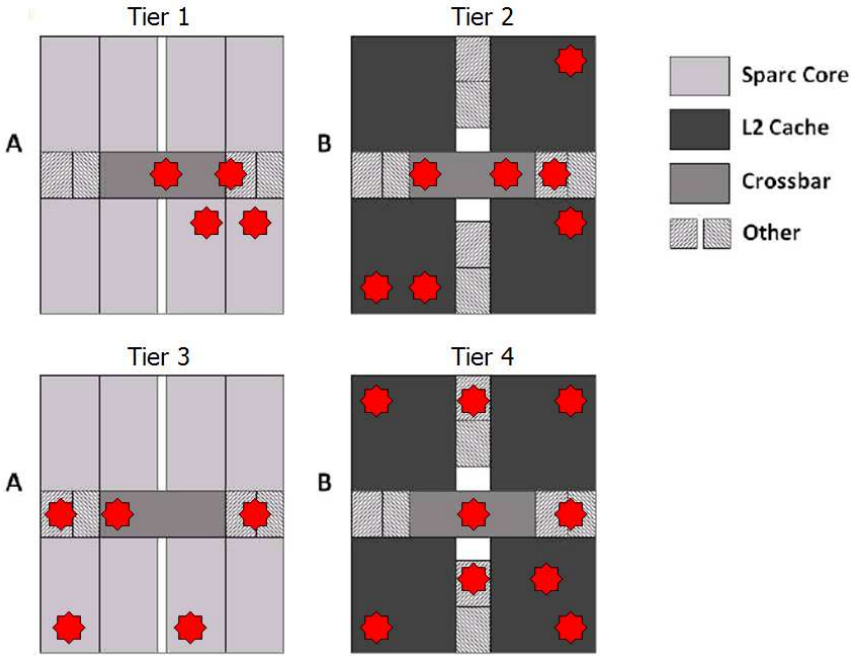

Figure 7.12: Sensor placement for our case study with sensors (marked as red stars on the floorplan) sampling frequency T_s of $1ms$

25 sensors for our case study. This means that with only 25 sensors it is possible to estimate the thermal profile of the 3D-MPSoC. This means that we achieved a reduction of a factor 8 in the number of required sensors. Figure 7.12 shows the resulting placement assuming a sensors sampling frequency T_s of $1ms$.

7.4 Summary

In this chapter we present three approaches to estimate the thermal profile from sensors located in some specific locations on the MPSoC floorplan.

The first method derives the thermal profile by means of many sensing devices placed on the silicon layer. The derivation is presented here by working on the different heat propagation dynamic between the cell and its neighbors.

The second approach is based on a complete design space exploration of all possible sensor placements. Then, according to user defined constraints, the configuration that maximizes the observability of the system is chosen. This approach is a method performing a complete design exploration on the full model without any model order reduction techniques.

The third technique finds the sensor placement configuration that maximizes the observability of the system with a greedy algorithm. The algorithm is based on a model order reduction performed on the thermal model of the MPSoC.

8 Experimental Framework

In this chapter we describe the simulation infrastructure that we used to validate and test thermal management systems, models and algorithms proposed in this thesis.

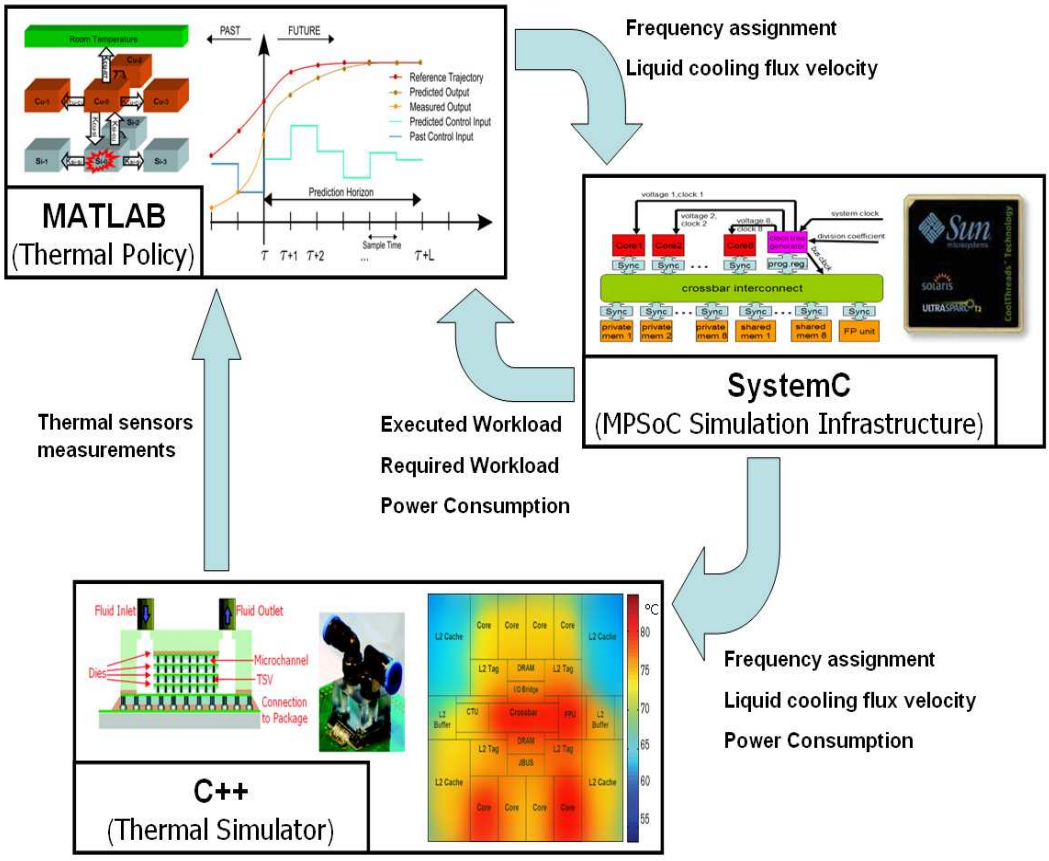

Figure 8.1: Global infrastructure block diagram

8.1 Global Infrastructure Overview

To simulate thermal management policies in a realistic environment, I created an infrastructure based on many tools interacting each others. The architecture consists of three main blocks. The block diagram is shown in Figure 8.1.

8.1.1 Thermal policy computation

The first block is written in MATLAB and it is the responsible for the thermal management policy. This code receives information from a file and produces another file containing the frequency assignment. If the system has liquid cooling, it generates also the velocities for the fluid circulating in each layer of the 3D-MPSoC.

The policy is an algorithm that runs at runtime and it is based on the receding horizon philosophy described in previous chapters. This algorithm needs from the MPSoC simulation infrastructure current power consumption data and both the requested and the executed workload.

Temperature sensors temperature measurements are also provided from the

thermal simulator block. These data are temperature measurements performed in some specific locations on the thermal profile. These data are used by the MATLAB block to generate the thermal profile of the MPSoC every time the thermal management optimization is performed.

During the runtime execution of the MATLAB code, the simulation is frozen until the completion of the algorithm that usually takes few milliseconds on the CORE2 DUO computer we used in our setup.

8.1.2 MPSoC simulation infrastructure

The second block is written in SystemC and it is the responsible for the MPSoC simulation infrastructure. This unit simulates in an accurate way the MPSoC system. This design space exploration engine supports hardware abstraction level and continuity between architectural and hardware design, and at the same time it fully supports multiprocessing. This block models the 2D-MPSoC and 3D-MPSoC of our simulation setup described in Sections 5.2 and 6.2 respectively.

It collects information related to current run-time MPSoC setting, freezes the SystemC simulation and starts the MATLAB block that implements the thermal management policy. Just collected information is passed to this function. The thermal policy is applied every T_{pol} seconds. Once the solution is computed, this unit unfreezes the simulation and waits a specific amount of time T_{cp} before setting frequencies and voltages among the cores and the interconnect. In the case of a 3D-MPSoC with liquid cooling, the flux velocity is also set for each layer. T_{cp} models the time needed by the actual hardware implementation of the policy to solve the frequency assignment problem. In our case we assumed a worse case scenario by setting T_{cp} equal to $50\mu s$.

This algorithm needs from the thermal management algorithm the frequency setting and in the case of a 3D-MPSoC with liquid cooling, the flux velocity for each layer. This blocks produces as outputs the power consumption and the requested and executed workloads. This information is sent to the thermal policy to compute the frequency setting for the next T_{pol} seconds. The frequency setting along with the flux velocities and power consumption data are sent to the thermal simulator to compute the thermal profile of the MPSoC.

8.1.3 Thermal simulator

This block is written in C++ and it is the responsible for the thermal profile generation of the overall MPSoC structure. This code receives information from a file and produces at runtime the evolution of the thermal profile of the MPSoC. These information are the frequency setting along with the flux velocities and power consumption data.

The output of this block is used for two reasons. The first one is to validate and compare the performance of the thermal policies under comparison.

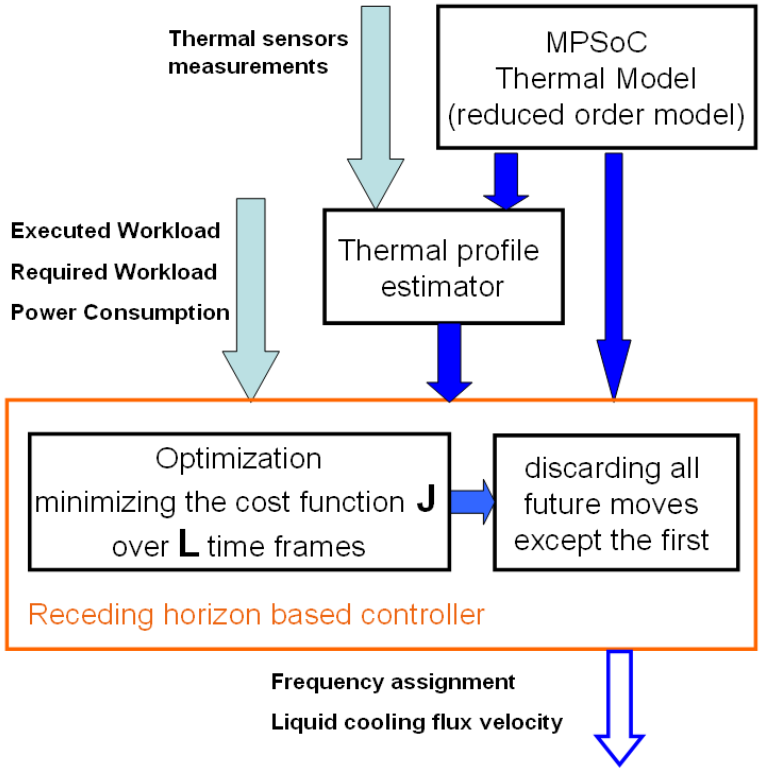

Figure 8.2: Global infrastructure block diagram

The second one is used to simulate thermal sensors placed in some specific locations on the MPSoC. These data are used by the MATLAB block to generate the thermal profile of the MPSoC every time the thermal management optimization is performed.

The thermal simulator is the one described in Section 8.5 and 8.4 of this thesis. This simulation tool models with high accuracy also all the cooling system infrastructure such as microchannels, the cooling liquid temperature and the heat spreading layer in the case of air cooling.

8.2 Thermal policy: Matlab code architecture

To implement the thermal management policy algorithm, we created a software architecture based on many functional blocks interacting each others. The block diagram is shown in Figure 8.2.

8.2.1 Optimization algorithm

The main optimization algorithm is surrounded by the orange box called *receding horizon based controller*. The algorithm minimizes a pre-specified cost function **J** over a prediction horizon of **L** time frames. The cost function

is different for every policy, however, all cost functions have the same input parameters and the all generates the same outputs. The outputs are the frequency assignment and if the system has liquid cooling, also the velocities for the fluid.

The receding horizon based controller receives many input parameters. It receives the executed workload, the required workload and the power consumption directly from the SystemC simulation framework. It receives also the thermal profile of the MPSoC and its reduced order thermal model from two distinct MATLAB functions.

8.2.2 Reduced order thermal model

Information about the thermal model is used in the MATLAB code for two specific purposes. The model is used by the optimization algorithm to predict the thermal response of the MPSoC to control. This model indeed computes the thermal profile according to power consumption values and a specific frequency setting. The second purpose is to estimate the thermal profile of the MPSoC from thermal sensors measurements coming from specific locations on the MPSoC.

8.2.3 Thermal profile estimator

The MATLAB function responsible for the thermal profile estimation can be implemented in many ways. In this thesis we used three methods. The first one is based on a least squares minimization (see the work by Boyd et al. [11] for more details). The second one is based on a Kalman filter (see the work by Franklin et al. [33] for more details). The last method methods requires a matrix multiplication and it is proposed by Zanini et al. [98]. The first two methods are more computationally expensive but they are less sensitive to thermal noise affecting thermal sensors measurements. The last method requires a small number of operations to estimate the thermal profile, however, it is more sensitive to thermal noise.

8.3 MPSoC emulator: SystemC code architecture

8.3.1 Virtual Platform Environment

The block diagram of our MPSoC simulation infrastructure is shown in Figure 8.3. The simulation architecture is a SystemC-based simulation platform based on MPARM (see Benini et al. [3] for more details). The main device consists of eight 32-bit cores, and 4 shared memories (L2 Cache Banks(0..3)). Private memories are assumed to be in side each core (as core we refer to a processing unit and a private memory). All these units communicate among each others

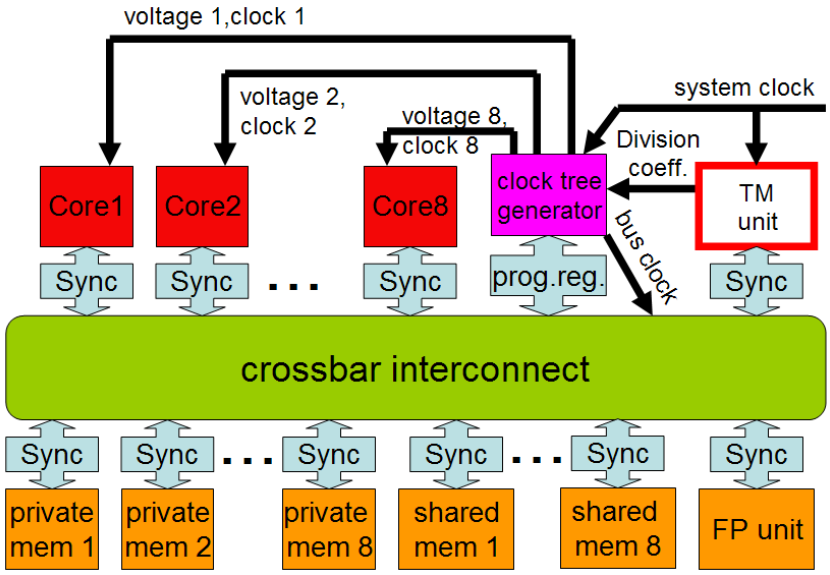

Figure 8.3: Simulation infrastructure block diagram

by a crossbar interconnect. A floating point unit is also connected to it. A DRAM interface connects to main memory. The virtual platform environment provides also power statistics for the several hardware modules in the simulated platform. The simulation is based on applications generating functional data traffic on the target architecture.

8.3.2 DVFS Support

The virtual platform supports different working frequencies and voltages for each processor core. For this purpose, a variable clock tree generator (see Equation 5.1), programmable registers and a synchronization module have been integrated in the simulation platform. The clock tree generator feeds the hardware modules of the platform (processors, memories, etc.) with independent and frequency scaled clock trees. The frequency scaled clock trees are generated by means of frequency dividers (shift counters), whose delay can be configured by users at design-time.

Scaling the clock frequency of the processors creates a synchronization issue with the system bus, which is assumed to work at the maximum frequency. The processing cores and the bus interface communicate by means of a handshaking protocol which assumes the same working frequency at both sides. Therefore, a synchronization module was designed, containing two dual-ported FIFOs wherein data and addresses sent by the bus interface to the processor and vice versa are stored. This module works with a dual clock: one feeding the core side and one feeding the bus interface side. Finally, as shown in Figure 8.3 the module also takes care of properly interfacing processor to bus signals, and a dedicated sub-module is implemented for this purpose.

8.3.3 Support for Thermal Management Policies

To simulate thermal management policies, we added a *thermal management* (TM) unit to the network of Figure 8.3. This unit is a SystemC module that supports an intercommunication between the thermal policies written in Matlab with the systemC-based Platform. This collects information related to current run-time MPSoC setting, freezes the SystemC simulation and starts a Matlab-based function that implements the thermal management policies. Just collected information are passed to this function. The Matlab function analyzes these data and solves the frequency assignment problem (see previous sections).

The solution represents the frequency setting for the next cycles, until the thermal policy is applied again after T_{pol} seconds. Once the solution is computed, the TM unit unfreezes the simulation and waits a specific amount of time T_{cp} before setting frequencies and voltages among the cores and the interconnect. In the case of a 3D-MPSoC with liquid cooling, the flux velocity is also set for each layer. T_{cp} models the time needed by the actual hardware implementation of the policy to solve the frequency assignment problem. In our case we assumed a worse case scenario by setting T_{cp} equal to $50\mu s$.

8.4 Thermal Simulator

8.4.1 Challenges in thermal simulators design

To study the thermal profile of a processor, many tools have been proposed. The most important ones (in relation to hotspot computation) have been described by Skadron et al. [81] and Paci et al. [64]. In this section we analyze limitations that these tools show from a thermal management designer operating viewpoint.

In Hotspot, the first limitation is that, to change the position of an element in the floorplan, the user has to measure the exact position of all the other elements contained in it. This way of describing the system is very time consuming and increases the probability of making a mistake. Another consequence of this way of measuring is that it is quite complicate to rearrange the blocks inside the floorplan since it's quite difficult to establish the new position of them.

A limitation in the thermal simulator proposed by Paci et al. [64] is that to simulate rectangular blocks, a big matrix must be filled and its dimension is inversely proportional to the size of the smallest square element inside the floorplan. A further problem is that, to simulate the system, power values of each component for each simulation step must be specified into a file by hand. This is an additional overhead that the user has to face to have the simulation done.

Another problem is that output information of both toolboxes is a file in which temperature values of all toolboxes for every simulation step are

```
# Floorplan of the niagara processor: UltraSPARC T1
# comment lines begin with a '#'
# comments and empty lines are ignored

cell:
300E-6

dimensions:
a,      3,      30,     2.044
c,      8,      18,     2.933
d,      11,     22,     2.412
f,      8,      12,     1.181
g,      8,      5,      0.617
h,      8,      10,     0.098
i,      19,     10,     4.948
t,      11,     10,     1.165
x,      10,     5,      0.802
y,      10,     6,      0.913
u,      1,      1,      0.006
end

positions:
a,      cc,     0,      4
d,      b-,     2,      2
c,      bc,     0,      0
c,      b-,     0,      0
c,      b-,     0,      0
c,      b-,     0,      0
d,      b-,     0,      6
a,      bc,     2,      4
t,      cc,     16,     18
x,      b-,     0,      0
y,      -b,     0,      0
t,      b-,     0,      -5
f,      -b,     3,      0
g,      b-,     0,      1
g,      -b,     0,      0
g,      c-,     5,      -5
g,      -b,     0,      0
h,      b-,     0,      -5
```

Figure 8.4: First example of a description using the new description language.

reported. This is a very non-intuitive way of showing results. Other representations of outputs must be generated by the tool.

8.4.2 Proposed syntax and output types

A new language to describe layouts has been introduced. We call this language *"rearranging notation"* and it is based on the fact that blocks positions on the floorplan are described in a way that makes the floorplan reconfiguration and power traces generation easier to reconfigure as compared to preceding systems. The following picture of Figure 8.4 gives an example of this new floorplan description language.

Rearranging Notation Semantics

Using this new language to describe floorplans it is possible to insert comments and notes in the text by using the symbol "#". After that a command called

"**cell**" specifies the cell dimension. This number specifies the resolution with whom the floorplan is specified. In this case it is equal to $300\mu m$.

The keyword "**dimensions** :" denotes the starting of a new session in which all the basic components of the floorplan are specified. The first column is an ascii character that specifies the specific component on the floorplan. The second and the third column specifies respectively the component width and length in cell numbers. The last column specifies the power in Watt consumed by the specific component. The last element type called "**u**" in the table specifies the null element used for wiring. The empty spaces in the floorplan description will be automatically filled by this "**u**" elements by the tool.

The keyword "**positions** :" denotes the starting of the last session in which positions of all the chip components are located in the floorplan. To the floorplan in fact has been overlapped a coordinate system having the x axis on the bottom side of the chip. The y axis is located on the left side of the chip. They both have the 0 located on the bottom left edge of the chip. For every line in this section, the first column is an ascii character that specifies the component type.

The following columns specify the position of the bottom left corner of the block. The second one is composed by 2 ascii characters. They both could be any of the following ones: "**c**", "**b**" or "−", leading to a total of 9 possible combinations. The meaning is the following: the first one describes how the x coordinate system is derived, while the second one the y one. The letter "**c**" means that the value of the coordinate will be directly specified. The letter "**b**" means that the position will be the one of the previous block plus the dimension of it. This enables to place very easily blocks close to each others without caring about their coordinate. The symbol "−" means that the position is the same as the one of the previous block.

The third and fourth columns are related respectively to the x and y axis and their meaning is related to the type of the characters respectively present at the first and second place of second column. If the symbol is a "**c**", the number expresses directly the coordinate. If the symbol is a "−" or a "**b**" it describes offset in relation to the starting position specified by the symbol.

Layout Description Example

Just to make an example, by looking at Figure 8.4 in the first line of section **positions** :, a block of type "**a**" is explicitly set in position (0,4). The second line states that a block of type "**d**" is placed with an offset of 2 in the vertical position in relation to preceding position, so at the location 4+2=6. It says also that the block is placed with an offset of 2 in the horizontal position starting from the end of block "**a**". This means that the block is located in the horizontal position 0+3+2=5.

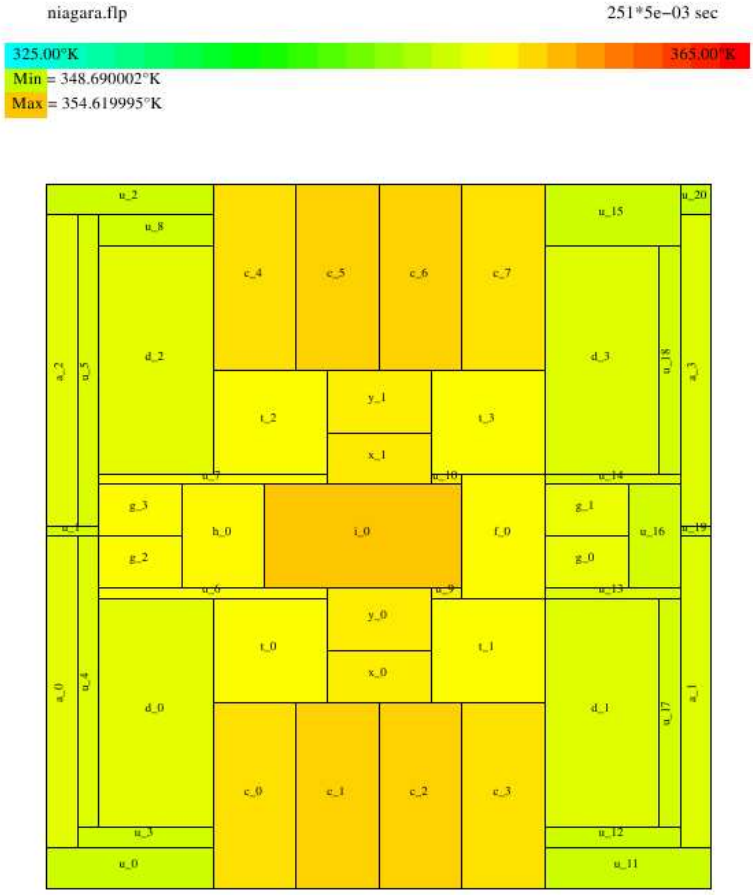

Figure 8.5: Niagara, chip thermal profile at time 1.255ms.

Output Types

This way of describing the floorplan adapts the Cartesian reference system to the one that our perceptual system in our brain uses to describe things that work by proximity. This notation is very compact and supports a complete description with very few lines of code.

The way the output is shown to the user makes use of all the system that our brain uses to receive information, that are numeric, video and images.

This enables the user to get all the needed information in the fastest way possible without loosing details. Figure 8.5 gives an example of possible output of this new temperature simulation tool. This figure shows the thermal profile of our case study resembling the Niagara chip at time $1.255ms$. This is an example of picture output. A movie of the MPSoC warming up is also provided as output as well as the same information using a 3D matrix. In Figure 8.5 all temperature are shown in terms of colors. Cold colors (i.e. blue, green) correspond to low temperatures while hot colors (i.e. orange, red) correspond to hot temperatures. On the top of Figure 8.5 there is a small legend showing the minimum and the maximum MPSoC temperature with the corresponding

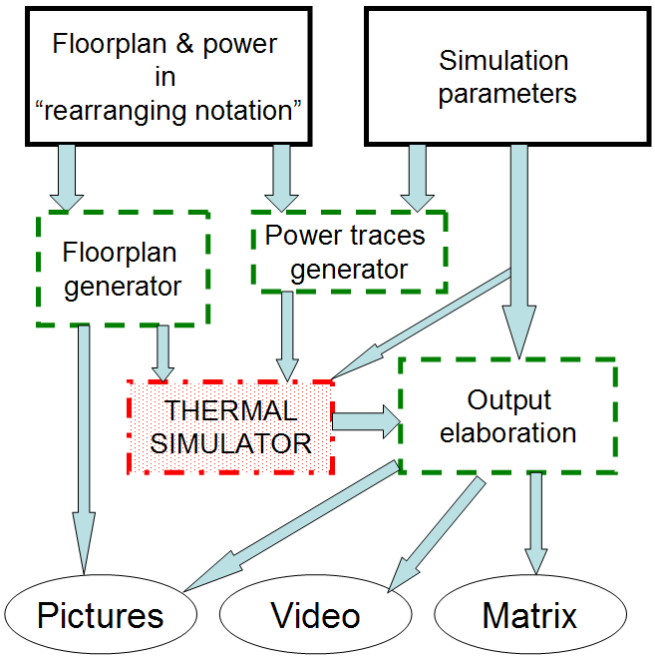

Figure 8.6: Functional block diagram of the developed framework

color. Each block is labelled with a string. The first letter identifies the block type (i.e. **a**, **b**, **c** ...). The last number of the string identifies the occurrence of that block in the layout. For example, the string "c_1" means that that is the block number 1 of type c.

8.4.3 Functional diagram of the simulator

The block functional diagram of the new developed framework is shown in Figure 8.6.

Input data needed to run the system are the black solid boxes on the top of Figure 8.6. Needed information are the floorplan using the new notation, power data for each block composing the MPSoC and the user-defined simulation parameters.

Output results are represented by the round black boxes on the bottom of Figure 8.6. Snapshots of the thermal profile are provided, as well as video outputs as well as a matrix containing the temperature of each cell composing the floorplan at every simulation time step.

The floorplan generator block generates a snapshot of the floorplan using colors to highlight blocks of the same type. It generates also the state-space model of the MPSoC.

The power traces generator provides the power traces needed by the thermal simulator using benchmarks and user-defined simulation parameters (i.e.: the average duty cycle of the generated workload).

The thermal simulator is the one described in Section 8.5 of this thesis. The

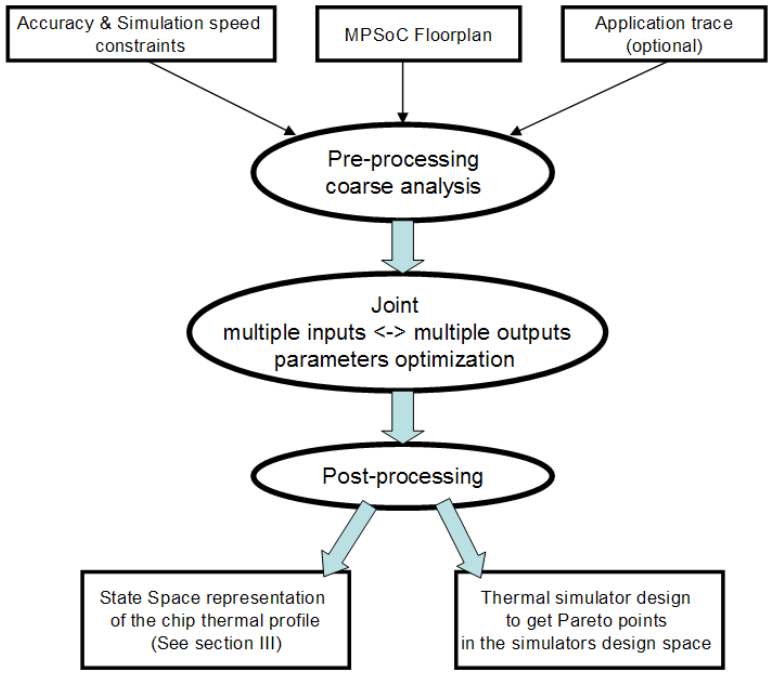

Figure 8.7: Thermal simulator design flow

precision of the proposed differential equations integration solver is shown by looking at the digits after the dot in the min and max temperature measurements. In this case it's 10^{-6}.

The output elaboration box transforms the simulation output in a form that is easier to the human mind to understand than a numeric form such as video and thermal profile pictures. A Matlab-compatible matrix notation is also generated as output.

8.5 Adaptive Thermal Simulation Algorithm

In Chapter 3 we analyzed some integration methods for state-of-the-art thermal simulators and we have proposed a generic way to represent them using a matrix-based state space representation. Parameters affecting the accuracy and the speed of the simulation have been analyzed both theoretically and experimentally. This section presents a design flow that determines the thermal simulator that shows the better speed/accuracy trade-off for the desired simulation constraints. The proposed methodology exploits the previously derived results for the various simulation methods.

8.5.1 Methodology

The block diagram of the design flow for thermal simulation is shown in Figure 8.7. The design and simulation flow has the following as input parameters:

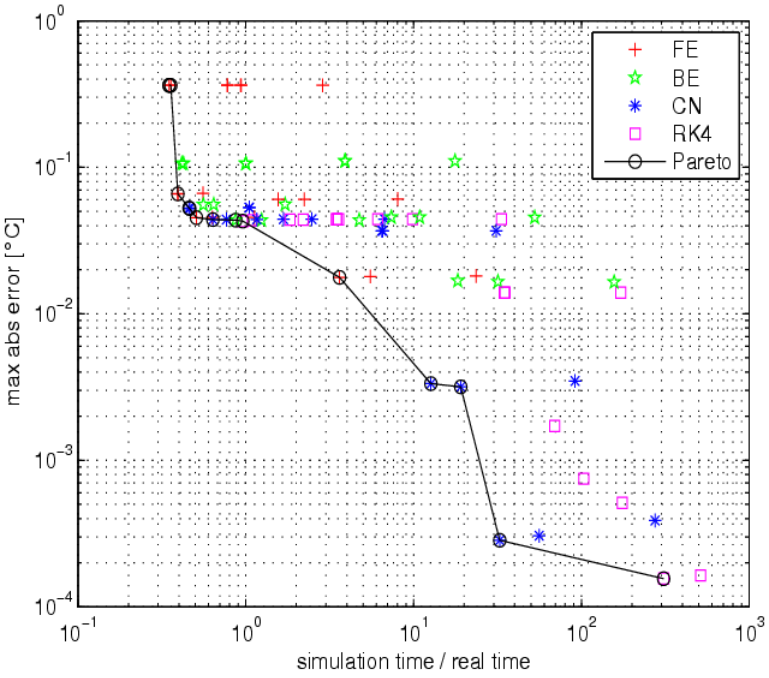

Figure 8.8: Design space exploration using our adaptive thermal simulator

MPSoC floorplan, power traces for the functional units (optional), desired simulation accuracy and simulation speed. The flow produces two outputs: 1) The state-space representation of the chip thermal profile (see Section 3). This output is also useful for the thermal management policies that require a model of the MPSoC to perform optimizations. 2) The design of a thermal simulator, including a set of choices regarding the order of the ODE solver, the simulation time step, and the parameters described in Section 3.2 to maximize the simulation speed for a given accuracy.

The thermal simulator does three main steps. The first one performs a pre-processing of input parameters. It computes the maximum allowable Δ_{max} by using Eqn. 3.45, 3.43, 3.41 and 3.40. Then, it simulates the thermal behavior of the MPSoC by using randomly generated power traces or real-life power traces. This simulation takes a short time, and is executed by using a coarse-grid granularity. More specifically, Δ_{max} is used as simulation time step and the selected grid granularity is equal to the dimension of the smallest functional block of the floorplan. In our case study, we simulated the system for a total simulation time of 100ms, and the simulation was repeated for all the solvers (FE, BE, CN and RK4). The goal of this pre-processing phase is to identify the order of the solver and the order of magnitude of the parameters that will maximize the performance of our solver for the given constraints. For this reason there is no need to perform thermal simulations with higher accuracy (and longer simulation time).

The second phase performs a design space exploration in the range of values

identified by the pre-processing phase. This phase varies all parameters by a few multiplicative factors and simulates the MPSoC for a longer time. Since matrices used by the solvers are temperature dependent, the simulation has to be long enough to generate temperature differences in the MPSoC thermal profile, to make the position of each Pareto point (shown in Figure 8.8) more reliable. More specifically, in the 8-core case study, we simulated the overall system for 10 seconds for all the combinations of the following parameters: two grid granularity values (block size/cell size = 1; 2), three step size values ($\Delta\tau = \Delta_{max}; \Delta_{max}/2; \Delta_{max}/4$), and three values of matrix calculation period ($T_{MC} = \Delta\tau; 5\Delta\tau; 9\Delta\tau$).

To measure the accuracy error, the 4th order ODE solver with a small simulation time step and a high grid granularity is used as the baseline. The last post-processing step of the design phase creates the two outputs in Figure 8.7 with the optimum parameters, as discussed in the next section.

8.5.2 Experimental validation results

This section presents an experimental validation of the method proposed in the previous section. The calculation of Δ_{max} for different grid resolutions has already been shown in Figure 3.4. Following the proposed design method, we simulate the system for a very short time (100ms) with $\Delta\tau = 3E-3$s and a coarse-grid resolution with the block size to cell size ratio equal to 1. At this point we identify the order of the solver, and the proposed simulator design flow automatically refines our simulation until we obtain the desired optimum point in the design space. We simulated the case study for the parameters described in previous sections for all the simulators (FE, BE, CN, RK4). Results are shown in Figure 8.8.

The optimum configurations of parameters (or Pareto points) provide the highest accuracy for a given simulation speed or vice versa. These configurations are automatically selected by our simulation framework according to the required accuracy or simulation speed for the given MPSoC design. Once these Pareto points in the design space are computed, our thermal simulator design can select the one closer to the input constraints specified. Figure 8.9 compares the speed and the accuracy of the proposed method with state-of-the-art-thermal simulators like HotSpot [81] and the FE-based thermal simulator presented in [64]. All three groups of results (Proposed method, HotSpot, FE-based simulator) have inaccuracy values with respect to a 4th order ODE solver with a small simulation time step and a high grid granularity. The setup time has been taken into account. The simulation time step is constant in both implementations. These results indicate that our adaptive simulation framework, which utilizes various ODE methods, can improve the simulation speed up to 70% with respect to HotSpot, while resulting in 6× higher accuracy.

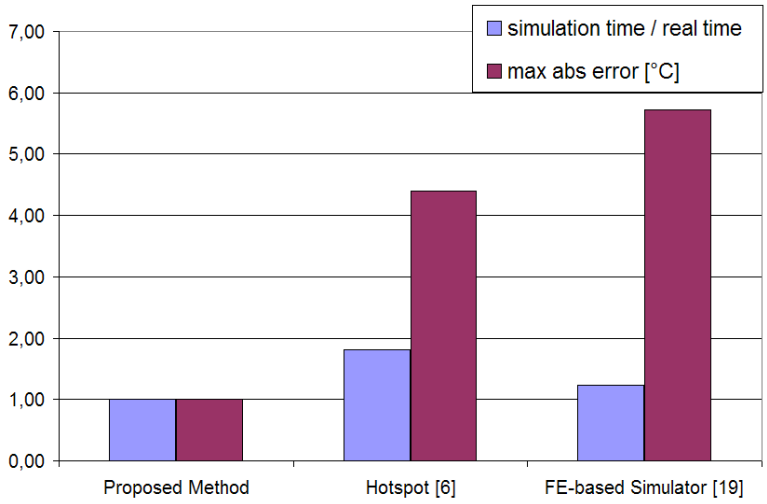

Figure 8.9: Normalized comparison of the proposed method (accuracy=$3 \cdot 10^{-3}\ ^{\circ}C$) with RK4-based thermal simulators (as HotSpot) and (FE)-based thermal simulators.

8.6 Summary

In this chapter we describe the simulation infrastructure that we used to validate and test thermal management systems, models and algorithms proposed in this thesis.

The first three sections give an overview of the overall simulation infrastructure. A functional description of each one of its components is also provided.

Section 4 and Section 5 describe in detail the thermal simulator we proposed to test, validate and compare the thermal policies proposed in this thesis.

9 Conclusions

9.1 Thesis Summary and Contributions

In this thesis, I have proposed improvements in three major fields of thermal management for MPSoCs. Chapter 2 gives a background on state-of-the-art techniques about the fields related to this work. Modeling, algorithms and system design are the main areas described by this survey.

The first field is Modeling. Chapter 3 introduces mathematical models used in this thesis. Models are related to the heat transfer model of the MPSoC with special emphasis to the way the system cools down (i.e. micro-cooling, natural heat dissipation etc.) and the heat propagates inside the MPSoC. A liquid cooling model of a 3d-MPSoC is also provided. Chapter 4 introduces the concepts developed to model the workload of an MPSoC system. Moreover some considerations are made about the system energy models used in this thesis. Workload prediction is also introduced and two estimation techniques are presented.

The second contribution is related to thermal management policies. New algorithms based on model predictive control have been proposed to maximize performance, increase reliability and minimize MPSoC power consumption. The proposed policies manage MPSoC working frequencies and micro-cooling systems to reach their goals in the most effective possible way and consuming the lowest possible amount of resources. Chapter 5 introduces air cooling algorithms. Four families of policies are analyzed and compared by both theoretical studies and experimental tests. We also provide a classification of the policies according to their problem formulation complexity and their computational effort requirements. The 2D-MPSoC case study used to run the simulations is also presented in detail. Chapter 6 introduces liquid cooling algorithms. Two novel algorithms are proposed here. The first one is based on a

centralized controller based on convex optimization. The second controller is a distributed structure based on the interaction between a global unit and many small controllers. The 3D-MPSoC case study used to run the simulations is also presented in detail.

Finally we implemented the proposed policies and methods with an innovative hardware simulation platform. Contributions in this field deal with constraints and problems related to the implementation of the proposed policies in 3D MPSOC systems. Chapter 7 introduces techniques to perform a detailed thermal profile estimation of the MPSoC structure. Two techniques are presented here to achieve a temperature estimation by using few thermal sensors placed in specific locations on the MPSoC. Chapter 8 presents the thermal simulation infrastructure used to test and compare policies presented in this thesis. The infrastructure consists of many parts written with three different programming languages and simulated on different simulation platforms.

9.2 Future Work

Although much research has been devoted to thermal management systems design, this area has not yet reached complete maturity. Contributions proposed in this thesis can be extended in various interesting directions. In the sequel I will point some interesting points.

In this thesis, the MPSoC thermal model has been studied and modelled by linearizing the thermal properties of materials. An interesting point could be to define the model in a way that could take into account nonlinearities of coefficients.

In the thermal model I developed, the integration method can be made more accurate and possibly with a variable integration step. Moreover the density of the wiring on the metal layer has been assumed as constant over the overall floorplan. This assumption can be removed by allowing the thermal model to consider different wiring densities in the metal layer.

The power management algorithms I presented assume workloads that change over time and are non-stationary. However, the workload prediction methods I have proposed are reliable only for a horizon that is shorter than 50ms-100ms. New innovative solutions can be proposed to make the prediction more accurate for longer time horizons.

System designers become more conscious of power dissipation issues and an increasing number of power-optimized commodity components is made available. The software design methodology I presented is user driven and would greatly benefit from automation. A new generation of power optimization tools is needed to choose and manage such components. Some optimizations can be automated at the design time. A system can be developed that can guide the designer in selection and implementation of appropriate design criteria.

Energy-efficient design and utilization at the system level will continue to

be a critical research topic in the next few years as there are still many unsolved problems and open issues to be solved.

Bibliography

[1] Ims research report. http://www.imsresearch.com/, 2009.

[2] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos. The explicit linear quadratic regulator for constrained systems. In *Automatica*, 2002.

[3] L. Benini, D. Bertozzi, A. Bogliolo, F. Menichelli, and M. Olivieri. "Mparm: Exploring the multi-processor soc design space with systemc". *Journal of VLSI Signal Processing, Springer*, 2005.

[4] L. Benini, A. Bogliolo, G. A. Paleologo, and G. De Micheli. "Policy optimization for dynamic power management". *TCAD*, 1999.

[5] L. Benini and G. De Micheli. *Dynamic Power Management: Design Techniques and CAD Tools*. Kluwer Academic Publishers, 1998.

[6] A. Bhunia, K. Boutros, and C. Chung-Lung. High heat flux cooling solutions for thermal management of high power density gallium nitride hemt. In *ISCTP*, 2004.

[7] W. L. Bircher and L. John. Complete system power estimation using processor performance events. In *Transactions on Computers*, 2011.

[8] B. Black, A. Murali, N. Brekelbaum, J. DeVale, L. Jiang, G. H. Loh1, D. McCauley, P. Morrow, D. W. Nelson, D. Pantuso, P. Reed, J. Rupley, S. Shankar, J. Shen, and C. Webb. Die stacking (3d) microarchitecture. In *IEEE MICRO*, 2006.

[9] P. Bose. Power-efficient microarchitectural choices at the early design stage. In *PACS*, 2003.

[10] T. Boukhobza and F. Hamelin. State and input observability recovering by additional sensor implementation: a graph theoretic approach. In *Automatica*, 2009.

[11] S. Boyd, , and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[12] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[13] S. Boyd and B. Wegbreit. Fast computation of optimal contact forces. In *IEEE Transactions on Robotics*, 2007.

[14] T. Brunschwiler, B. Michel, H. Rothuizen, U. Kloter, B. Wunderle, H. Oppermann, and H. Reichl. Interlayer cooling potential in vertically integrated packages. In *Microsystem Technologies*, 2008.

[15] T. Brunschwiler, S. Paredes, U. Drechsler, B. Michel, W. Cesar, G. Toral, Y. Temiz, and Y. Leblebici. Validation of the porous-medium approach to model interlayer-cooled 3d chip stacks. In *3DIC*, 2009.

[16] T. Brunschwiler, H. Rothuizen, U. Kloter, H. Reichl, B. Wunderle, H. Oppermann, and B. Michel. Forced convective interlayer cooling potential in vertically integrated packages. In *ITHERM*, 2008.

[17] R. L. Burden and J. D. Faires. *Numerical Analysis*. Brooks Cole, 2000.

[18] K. Choi, R. Soma, and M. Pedram. Dynamic voltage and frequency scaling based on workload decomposition. In *Proceedings of the 2004 international symposium on Low power electronics*, 2004.

[19] R. J. Cochran and S. Reda. Consistent runtime thermal prediction and control through workload phase detection. In *DAC*, 2010.

[20] A. K. Coskun, D. Atienza, T. Rosing, T. Brunschwiler, and B. Michel. Energy-efficient variable-flow liquid cooling in 3d stacked architectures. In *DATE*, 2010.

[21] A. K. Coskun, J. L. Ayala, D. Atienza, and T. S. Rosing. Modeling and dynamic management of 3d multicore systems with liquid cooling. In *VLSI-SoC*, 2009.

[22] A. K. Coskun, T. S. Rosing, and K. C. Gross. Proactive temperature balancing for low cost thermal management in mpsocs. In *ICCAD*, 2008.

[23] A. K. Coskun, T. S. Rosing, and K. C. Gross. Temperature management in multiprocessor socs using online learning. In *DAC*, 2008.

[24] A. K. Coskun, T. S. Rosing, and K. Whisnant. Temperature-aware task scheduling. In *DATE*, 2007.

[25] CoWare. N2c. http://www.coware.com/cowareN2C.

[26] J. Donald and M. Martonosi. Techniques for multi-core thermal management: Classification and new exploration. In *ISCA*, 2006.

[27] Y. Eldar and D. Palomar. *Convex Optimization in Signal Processing and Communications*. Cambridge University Press, 2010.

[28] M. J. Ellsworth and M. K. Iyengar. Energy efficiency analyses and comparison of air and water cooled high performance servers. In *InterPACK*, 2009.

[29] emb. WILO MHIE centrifugal pump. http://www.wilo.com/cps/rde/xchg/en/layout.xsl/3707.htm.

[30] T. Emi, M. A. Al Faruque, and J. Henkel. Tape: Thermal-aware agent-based power economy for multi/many-core architectures. In *ICCAD*, 2009.

[31] B. Falsafi and D. A. Wood. Modeling cost/performance of a parallel computer simulator. In *TOMACS*, 1997.

[32] festo. Festo electric automation technology. http://www.festo.com.

[33] G.F. Franklin, J. D. Powell, and M. L. Workman. *Digital Control of Dynamic Systems*. McGraw Hill, 2011.

[34] M. Grant and S. Boyd. Cvx: Matlab software for disciplined convex programming. www.stanford.edu/ boyd/cvx/.

[35] Jan Haase, Markus Damm, Dennis Hauser, and Klaus Waldschmidt. Reliability-aware power management of multi-core processors. In *DIPES*, 2006.

[36] T. R. Halfhill. Transmeta breaks x86 low power barrier. In *Microprocessor Report*, 2000.

[37] C. J. Hughes, J. Srinivasan, and S. V. Adve. Saving energy with architectural and frequency adaptations for multimedia applications. In *IEEE Micro*, 2001.

[38] C.J. Hughes, V. S. Pai, P. Ranganathan, and S. V. Adve. Rsim: Simulating shared-memory multiprocessors with ilp processors. In *IEEE Trans. on Computer*, 2002.

[39] W.-L. Hung, G.M. Link, Yuan Xie, N. Vijaykrishnan, and M. J. Irwin. Interconnect and thermal-aware floorplanning for 3d microprocessors. In *ISQED*, 2006.

[40] JBB. http://www.spec.org/jbb2005/.

[41] S. Joshi and S. Boyd. Sensor selection via convex optimization. In *transaction on signal processing*, 2009.

[42] H. Jung and M. Pedram. Continuous frequency adjustment technique based on dynamic workload prediction. In *VLSI Design*, 2008.

[43] P. Kongetira, K. Aingaran, and K. Olukotun. Niagara: A 32-way multithreaded sparc processor. In *IEEE MICRO*, 2005.

[44] M. Kvasnica, P. Grieder, M. Baotic, and F. J. Christophersen. Multiparametric toolbox (mpt). 2006.

[45] laing. Laing 12 volt DC pumps datasheets. http://www.lainginc.com/pdf/DDC3_LTI_USletter_BR23.pdf.

[46] A. J. Laub, M. Heath, C. Paige, and R. Ward. Computation of system balancing transformations and other applications of simultaneous diagonalization algorithms. In *IEEE Trans. Automatic Control*, 1987.

[47] H. Lee, Y. Jeonga, J. Shinb, J. Baekc, M. Kanga, and K. Chuna. Package embedded heat exchanger for stacked multichip module. In *Transducers, Solid-State Sensors, Actuators and Microsystems*, 2003.

[48] J. S. Lee, K. Skadron, and S. W. Chung. Predictive temperature-aware dvfs. In *Transactions on Computers*, 2010.

[49] A. S. Leon, K. W. Tam, J. L. Shin, D. Weisner, and F. Schumacher. A power-efficient high-throughput 32-thread sparc processor. In *ISSCC*, 2006.

[50] P. P. P. M. Lerou, H. J. M. Ter Brake, H. J. Holland, J. F. Burger, and H. Rogalla. Insight in clogging of mems based micro cryogenic coolers. In *Applied Physics Letters, 90*, .

[51] P. P. P. M. Lerou, G. C. F. Venhorst, C. F. Berends, T. T. Veenstra, M. Blom, J. F. Burger, H. J. M. Ter Brake, and H. Rogalla. Fabrication of a micro cryogenic cooler using mems-technology. In *Journal of Micromechanics and Microengineering*, .

[52] Y. Lu, T. S. Rosing, and G. De Micheli. Software controlled power management. In *CODES*, 1999.

[53] M. Dales. Swarm. http://www.dcs.gla.ac.uk/michael/phd/swarm.html.

[54] M. Magno, D. Brunelli, L. Thiele, and L. Benini. Adaptive power control for solar harvesting multimodal wireless smart camera. In *ICDSC*, 2009.

[55] P.S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hllberg, J. Hgberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A full system simulation platform. In *IEEE Trans. on Computer*, 2002.

[56] S.O. Memik, R. Mukherjee, N. Min, and L. Jieyi. Optimizing thermal sensor allocation for microprocessors. In *IEEE TCAD*, 2008.

[57] Mentor Graphics. Seamless hardware/software co-verification. http://www.mentor.com/seamless/products.html.

[58] A. Merchant, B. Melamed, E. Schenfeld, and B. Sengupta. Analysis of a control mechanism for a variable speed processor. In *Transactions on Computers*, 1996.

[59] A. Milenkovic and V. Milutinovic. A quantitative analysis of wiring lengths in 2d and 3d vlsi. In *Microelectronics Journal, Elsevier*, 1998.

[60] R. Mukherjee and S. O. Memik. Physical aware frequency selection for dynamic thermal management in multi-core systems. In *ICCAD*, 2006.

[61] S.S. Mukherjee, S. K. Reinhardt, B. Falsafi, M. Litzkow, S. Huss-Lederman, M. D. Hill, J. R. Larus, and D. A. Wood. Wisconsin wind tunnel ii: A fast and portable parallel architecture. In *PAID*, 1997.

[62] S. Murali, A. Mutapcic, D. Atienza, R. Gupta, S. Boyd, L. Benini, and G. De Micheli. Temperature control of high performance multicore platforms using convex optimization. In *DATE*, 2008.

[63] A.V. Oppenheim and R. W. Schafer. *Discrete-Time Signal Processing*. Prentice-Hall, 1989.

[64] G. Paci, F. Poletti, and L. Benini. Exploring temperature-aware design in low-power mpsocs. In *DATE*, 2006.

[65] G. Paleologo, L. Benini, A. Bogliolo, and G. De Micheli. Policy optimization for dynamic power management. In *Proceedings of the Design Automation Conference*, 1998.

[66] A. Papoulis and S. U. Pillai. *Probability, Random Variables and Stochastic Processes*. Mc Graw Hill, 2002.

[67] E. Pop. Energy dissipation and transport in nanoscale devices. In *Springer*, 2010.

[68] K. Puttaswamy and G. H. Loh. Thermal analysis of a 3d die-stacked highperformance microprocessor. In *GLSVLSI*, 2006.

[69] Q. Qiu, Q. Wu, and M. Pedram. "Stochastic modeling of a power-managed system: construction and optimization". *TCAD*, 2001.

[70] J. M. Rabaey. *Low Power Design Essentials*. Springer, 2009.

[71] A. Ramalingam, F. Liu, S. R. Nassif, and D. Z. Pan. Accurate thermal analysis considering nonlinear thermal conductivity. In *ISQED 2006*, 2006.

[72] R. Rao, S. Vrudhula, C. Chakrabarti, and N. Chang. An optimal analytical solution for processor speed control with thermal constraints. In *Proceedings of the 2006 international symposium on Low power electronics*, 2006.

[73] Chu R.C. Advanced cooling technology for leading-edge computer products. In *5th International Conference onSolid-State and Integrated Circuit Technology*, 1998.

[74] S. Reda, R. J. Cochran, and A. N. Nowroz. Improved thermal tracking for processors using hard and soft sensor allocation techniques. In *Transactions on Computers*, 2011.

[75] M. Rosenblum, S. A. Herrod, E. Witchel, and A. Gupta. Complete computer system simulation: The simos approach. In *IEEE Parallel and Distributed Technology: Systems and Applications*, 1995.

[76] T. S. Rosing and S. P. Boyd. "Managing power consumption in networks on chips". *T-VLSI*, 2004.

[77] M. M. Sabry, A. K. Coskun, and D. Atienza. Fuzzy control for enforcing energy efficiency in high-performance 3d systems. In *ICCAD*, 2010.

[78] J. Schtze, H. Ilgen, and W. R. Fahrner. An integrated micro cooling system for electronic circuits. In *IEEE transactions on industrial electronics*, 2001.

[79] O. Semenov, A. Vassighi, and M. Sachdev. Impact of self-heating effect on long-term reliability and performance degradation in cmos circuits. In *IEEE T-D&M*, 2006.

[80] S. Sharifi and T. S. Rosing. An analytical model for the upper bound on temperature differences on a chip. In *GLSVLSI*, 2008.

[81] K. Skadron, M. R. Stan, K. Sankaranarayanan, Wei Huang, S. Velusamy, and D. Tarjan. Temperature-aware microarchitecture: Modeling and implementation. In *TACO*, 2004.

[82] slamd. SLAMD Distributed Load Engine. www.slamd.com.

[83] A. Sridhar, A. Vincenzi, M. Ruggiero, T. Brunschwiler, and D. Atienza. 3d-ice: Fast compact transient thermal modeling for 3d-ics with inter-tier liquid cooling. In *ICCAD*, 2010.

[84] A. Sridhar, A. Vincenzi, M. Ruggiero, T. Brunschwiler, and D. Atienza. Compact transient thermal model for 3d ics with liquid cooling via enhanced heat transfer cavity geometries. In *THERMINIC*, 2010.

[85] SROECO. Sroeco. http://sroeco.com/solar/solar-electric-cost-commercial.

[86] C. Sumana and C. Venkateswarlu. Optimal selection of sensors for state estimation in a reactive distillation process. In *Process Control*, 2009.

[87] P. Tndel, T. A. Johansen, and A. Bemporad. Evaluation of piecewise affine control via binary search tree. In *Automatica*, 2003.

[88] D. B. Tuckerman and R. F. W Pease. High-performance heat sinking for vlsi. In *IEEE Electron Device Letters*, 1981.

[89] K. Van, D. Verkest, I. Bolsens, and H. De Man. Coware-a design environment for heterogeneous hardware-software systems. In *DAC*, 1996.

[90] K. Waldschmidt. Robustness in soc design. In *DSD 2006*, 2006.

[91] Y. Wang, K. Ma, and X. Wang. Temperature-constrained power control for chip multiprocessors with online model estimation. In *ISCA*, 2009.

[92] J. Warren, S. Schaefer, A. N. Hirani, and M. Desbrun. Barycentric coordinates for convex sets. In *Journal of Advances in Computational Mathematics*, 2007.

[93] G. F. M. Wiegerinck, H. J. M. Ter Brake, J. F. Burger, H. J. Holland, and H. Rogalla. Thermodynamic optimization of sorption-based joule-thomson coolers. In *Cryogenics, 47*.

[94] Y. Xie and W. Hung. Thermal-aware allocation and scheduling for systems-on-chip. In *DATE*, 2005.

[95] F. Zanini, , D. Atienza, L. Benini, and G. De Micheli. Multicore thermal management with model predictive control. In *ECCTD*, 2009.

[96] F. Zanini, D. Atienza, A. K. Coskun, and G. De Micheli. Optimal multi-processor soc thermal simulation via adaptive differential equation solvers. In *VLSISoC*, 2009.

[97] F. Zanini, D. Atienza, C. N. Jones, and G. De Micheli. Temperature sensor placement in thermal management systems for mpsocs. In *ISCAS*, 2010.

[98] F. Zanini, D. Atienza, and G. De Micheli. A combined sensor placement and convex optimization approach for thermal management in 3d-mpsoc with liquid cooling. In *Integration, the VLSI Journal*, 2011.

[99] F. Zanini, David Atienza, G. De Micheli, and S. P. Boyd. Online convex optimization-based algorithm for thermal management of mpsocs. In *GLSVLSI*, 2010.

[100] F. Zanini, C. N. Jones, D. Atienza, and G. De Micheli. Multicore thermal management using approximate explicit model predictive control. In *ISCAS*, 2010.

[101] Y. Zhang and A. Srivastava. Adaptive and autonomous thermal tracking for high performance computing systems. In *DAC*, 2010.

Curriculum Vitae

Francesco Zanini

Education

2007 – 20011(exp.)	**PhD Candidate** Computer Science Swiss Federal Institute of Technology, Lausanne Lausanne, Switzerland.
2006 – 2007	**Master in Embedded System Design** Advanced Learning and Research Institute Lugano, Switzerland.
2005 – 2006	**Research Master in Microelectronic Engineering** Institute of Microelectronics and Wireless Systems, NUIM Maynooth, Ireland.
2003 – 2005	**Master in Microelectronic Engineering** University of Parma Parma, Italy.
2000 – 2003	**Bachelor in Electronic Engineering** University of Parma Parma, Italy.
1995 – 2000	**Liceo Scientifico** Liceo Scientifico A.Manzoni Suzzara(MN), Italy.

Publications

Journal Papers

- [J1] F. Zanini, D. Atienza, C.N. Jones, L. Benini, G. De Micheli, "Online Thermal Control Methods for Multi-Processor Systems". *submitted*, 2011.
- [J2] F. Zanini, M. Sabry, D. Atienza, G. De Micheli, "Hierarchical Thermal Management Policy for High-Performance 3D Systems with Liquid Cooling". *JETCAS*, 2011.
- [J3] F. Zanini, D. Atienza, G. De Micheli, "A Combined Sensor Placement and Convex Optimization Approach for Thermal Management in 3D-MPSoC with Liquid Cooling". *INTEGRATION*, 2011.

Conference Papers

- [C1] F. Zanini, D. Atienza, G. De Micheli, "Convex-Based Thermal Management for 3D MPSoCs using DVFS and Variable-Flow Liquid Cooling". *PATMOS, 2011.*

- [C2] F. Zanini, D. Atienza, L. Benini, G. De Micheli, "Thermal-Aware System-Level Modeling and Management for Multi-Processor Systems-on-Chip". *ISCAS, 2011.*
- [C3] F. Zanini, D. Atienza, C.N. Jones, G. De Micheli, "Temperature Sensor Placement in Thermal Management Systems for MPSoCs". *ISCAS, 2010.*
- [C4] F. Zanini, C.N. Jones, D. Atienza, G. De Micheli, "Multicore thermal management using approximate explicit Model Predictive Control". *ISCAS, 2010.*
- [C5] F. Zanini, D. Atienza, G. De Micheli, S.P. Boyd, "Online Convex Optimization-Based Algorithm for Thermal Management of MPSoCs". *GLSVLSI, 2010.*
- [C6] F. Zanini, D. Atienza, A.K. Coskun, G. De Micheli, "Optimal Multi-Processor SoC Thermal Simulation via Adaptive Differential Equation Solvers". *VLSI-SoC, 2009.*
- [C7] F. Zanini, D. Atienza, L. Benini, G. De Micheli, "Multicore Thermal Management with Model Predictive Control". *ECCTD, 2009.*
- [C8] F. Zanini, D. Atienza, G. De Micheli, "A Control Theory Approach for Thermal Balancing of MPSoC". *ASPDAC, 2009.*
- [C9] F. Zanini, M. Soudan, R. Farrell, "Methodology for Minimizing Mismatches in Time-Interleaved ADCs". *IMEKO, 2007.*

Bachelor and Master Thesis Projects Papers

- [T1] F. Zanini, L. Fiorin, "Reconfigurability analysis and implementation of NoC based architectures". *ALaRI Advanced Master*,2007.
- [T2] F. Zanini, R. Farrell, "Architectural Improvements Towards an Efficient 16-18 Bit 100-200 MSPS ADC". *National University of Ireland Master*,2007.
- [T3] F. Zanini, A. Facen, A. Ricci, A. Boni, "Progetto di un Sistema Integrato di Identificazione Operante in Banda UHF". *Parma University Master*,2005.
- [T4] F. Zanini, D. Vecchi, A. Boni, C. Morandi, "Progetto di un Circuito di Campionamento per Convertitore A/D a 14-b e 100MS/s". *Parma University Bachelor*,2003.