Alma Mater Studiorum
Università degli Studi di Bologna

D.E.I. – Electrical, Electronic, and Information Engineering "Guglielmo Marconi"

RESEARCH DOCTORATE IN
AUTOMATIC CONTROL SYSTEM AND OPERATIONAL RESEARCH

XXV CYCLE

Ph.D. dissertation on

MODEL PREDICTIVE CONTROL IN THERMAL MANAGEMENT OF MULTIPROCESSOR SYSTEMS-ON-CHIP

Author:

MATTEO CACCIARI

Advisor:

Dott. Ing. **ANDREA TILLI**

Ph.D. Coordinator:

Prof. **ANDREA LODI**

2013
SSD ING-INF / 04 - AUTOMATICA
SC 09 / G1 – AUTOMATICA

Model predictive control in thermal management of multiprocessor systems-on-chip

Matteo Cacciari

DEI – Electrical, Electronic, and Information Engineering department Guglielmo Marconi

University of Bologna

A thesis submitted for the degree of

PhilosophiæDoctor (PhD)

March 2013

Acknowledgements

This thesis represents the conclusion of a three years long journey which gave me the opportunity to meet nice and kind people and having life experiences that were unimaginable to me just few years ago.

I cannot find words to express my gratitude to my advisor, Dott. Ing Andrea Tilli, for the many opportunities he gave me, his guidance and support. In these years he has always been present with suggestions, advices and encouragements. It has been a great pleasure working with you!

A special thank goes to Prof. Luca Benini for giving me the possibility of working on the stimulating and very actual topic of this thesis, and for having believed in me and my abilities making me part of the team for the thermal management of systems-on-chip.

I heartily thank Prof. Emanuele Garone of the ULB University for the amazing period in Bruxelles. I will never forget his kindness and generous hospitality, the interesting conversation while preparing dinner, and the valuable suggestions related to this thesis. I hope we will have the opportunity to work together again in the near future!

I owe a special thank to Andrea Bartolini for being a great (and much more than a) colleague, sharing with me many of the successes and difficulties of this thesis (I cannot forget the nights spent working together for finishing papers before the deadline!).

Christian, Giovanni, Raffaele we lived together this Ph.D. adventure, helping each other and proving unquestionably the saying “unity is strength”. It would not be the same without you!

I would like also to thank all the guys and (the few) girls of the Center of Complex Automated Systems (CASY) for the time spent together and the fun. Thanks also

to the researchers of Micrel Lab (University of Bologna) for being always kind and nice. My gratitude goes also to the researchers of the SAAS department (ULB University), my home mates, and all the people met in Brussel for making my stay in Belgium so memorable.

Special thanks goes to my friends Alan, Alessandro, Alex, Alice, Cecilia, Federico, Lorenzo, Luca, Martina, Matteo, Riccardo, Simone, Stefania, Stefano (I beg your pardon if I forgot someone) for being so patient with me when I had to study, for sharing enjoyable moments and for being simply the best!

Last but not least, my deepest gratitude goes to my family, to my mother Serenella for having always a good word to encourage me in the difficult times, to my father Lorenzo for making my life easier everyday, and to my brother Alessandro for bearing with patience my tapping on the keyboard during the nights and helping me with this thesis. Without their loving support, this thesis would not have been possible.

Matteo Cacciari

Alma Mater Studiorum – University of Bologna
March 2013

Abstract

Multiprocessor Systems-on-Chip (MPSoC) are the core of nowadays and next generation computing platforms. Their relevance in the global market continuously increase, occupying an important role both in everyday life products (e.g. smartphones, tablets, laptops, cars) and in strategic market sectors as aviation, defense, robotics, medicine. Despite of the incredible performance improvements in the recent years processors manufacturers have had to deal with issues, commonly called “Walls”, that have hindered the processors development. After the famous “Power Wall”, that limited the maximum frequency of a single core and marked the birth of the modern multiprocessors system-on-chip, the “Thermal Wall” and the “Utilization Wall” are the actual key limiter for performance improvements. The former concerns the damaging effects of the high temperature on the chip caused by the large power densities dissipation, whereas the second refers to the impossibility of fully exploiting the computing power of the processor due to the limitations on power and temperature budgets. In this thesis we faced these challenges by developing efficient and reliable solutions able to maximize performance while limiting the maximum temperature below a fixed critical threshold and saving energy. This has been possible by exploiting the Model Predictive Controller (MPC) paradigm that solves an optimization problem subject to constraints in order to find the optimal control decisions for the future interval. A fully-distributed MPC-based thermal controller with a far lower complexity respect to a centralized one has been developed. The control feasibility and interesting properties for the simplification of the control design has been proved by studying a partial differential equation thermal model. Finally, the controller has been efficiently included in more complex control schemes able to minimize energy consumption and deal with mixed-criticalities tasks.

Contents

List of Figures

LIST OF FIGURES

LIST OF FIGURES

Chapter 1

Introduction

The main purpose of this chapter is to introduce the reader to the central themes this thesis deals with. First, we motivate the need of dynamic controllers to improve performance of chips multiprocessor. Then, we outline the major contributions and the organization of the remainder of this thesis.

1.1 MPSoCs and Multi-core basics

A *system-on-chip* (SoC) is an integrated circuit that implements most or all of the functions of a complete electronic system (1). It integrates on the same chip components as memory hierarchies, central processing units (CPUs), specialized logic, busses and other digital functions. Most of these system usually need more processing units to address the complexity of combining together very different components to create a unique harmonious and efficient system and achieving desired performance goals. A *Multiprocessor system-on-chip* (MPSoC) is a SoC that uses multiple programmable processors as system components.

According to (2), the MPSoC architecture is made of three types of components:

- The *hardware subsystems* uses hardware components to implement specific functionalities of an application or global memories (HW in Fig. 1.1). The intra-subsystem communication represents the communication inside the hardware subsystems between the different HW components (e.g small buses or point-to-point networks).

- The *software subsystems* represent programmable subsystems, also called processor nodes of the architecture. Inside this subsystem, we find an intra-subsystem communication

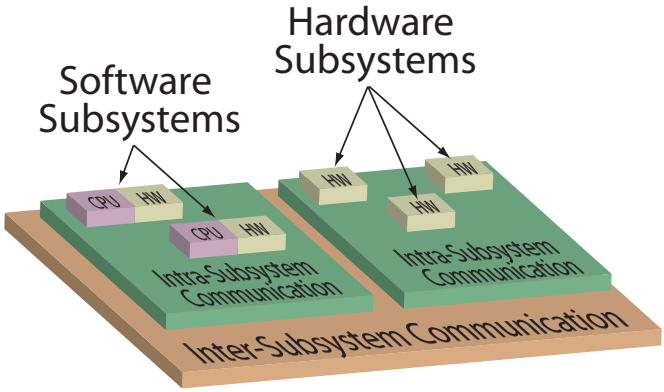


Figure 1.1: MPSoC architecture.

that connect the hardware components (HW in Fig. 1.1) as local memories, I/O components, or hardware accelerators, with computing resources. These latter represent the central processing units, CPUs, or equivalently the cores. Each core executes sequentially the instructions of a program stored in the memory. Depending on the number of cores a software subsystem can be defined as single-core or multi-core.

- Finally the *inter-subsystem communication* represents the communication architecture between the different software and hardware subsystems (e.g. Networks on Chip that allow simultaneous data transfers).

The architectures of the MPSoCs can be classified in two big families: *homogeneous* and *heterogeneous*. The former integrate on the same chip identical software subsystem instantiated several time. In literature this architecture is often referred to as parallel architecture model. The latter, instead, incorporate different software subsystems with different processing units like general purpose processors (GPP), digital signal processors (DSP) or application-specific instruction set processors (ASIP). The exchange of information between the subsystems can be manage according to two different communication models: shared memory and message passing. The shared memory approach allows all the CPUs to access simultaneously the memory to get information. This communication model fits well with homogeneous MPSoCs which has identical software subsystems. For heterogeneous MPSoCs it is preferable a message passing communication where each software subsystem explicitly asks for information.

Before proceeding it is useful to remark that in this thesis we will consider classical desktop multi-core processors as special cases of MPSoCs. This because the issues treated in this work

similarly affects all the architectures comprising multiple cores on the same chip. Moreover, it is important to highlight that in literature the two terms are often used as synonymous. Today multi-core processors can be compared to homogeneous MPSoCs, both contain multiple processing units on the same chip substrate and exploit parallelism to improve computational performance. Researchers have also shown the benefit of heterogeneous multi-core processors (3). However, according to the definition in (1), the main difference between the two architectures is related to the applications which they are designed for. Indeed, whereas multi-cores are targeted to general-purpose uses, the MPSoC are usually related to embedded applications.

Multi-core processors are commonly used in desktops, laptops, servers and data centers. Because of the high differentiability of the applications running on these devices, designers realize general-purpose architectures with the aim of improving average-case performance, intended as computation capacity or throughput. However, it is clear enough that in these architectures performance is strongly dependent on the application executed, making the variance in computing time larger.

On the other hand, MPSoCs are widely used in networking, communications, signal processing, and multimedia among other applications (e.g. smartphones, cameras, consoles, MP3 readers, DVD players, ...). Their architecture is designed in order to balance the complexity of the technology with embedded applications requirements. These requirements could be computing time deadlines in real-time applications, low-power consumption in mobile devices or short time-to-market. In these cases the use of a general-purpose architecture is counterproductive since it reduces performance at the expense of a useless generality, when application requirements are known. The aim of designers is improving worst-case performance making the computing time predictable. Thus, it is more convenient to design a new architecture rather than redesign the one of a multi-core. The new architecture should be the result of the tradeoff between the hardware specialization to meet application requirements with high performance and the programming complexity, increasing with the irregularity of the architecture and the variety of components integrated.

Fig. 1.2 shows the architecture of two MPSoCs by using block diagrams. The first, Fig. 1.2a, represents the Lucent Daytona structure (4). This processors is the first MPSoC processor of the history. It was presented in 2000 and it has been designed for wireless base stations. As we can see, Daytona has an homogeneous architecture with four CPUs attached to a high-speed bus. The second, Fig. 1.2b, represents the architecture of the ST Microelectronics Nomadik (5). This is a cell phones heterogeneous MPSoC which uses an ARM926EJ as its host processor.

Connected to the same bus there are two programmable accelerators (small DSPs), for audio and video.

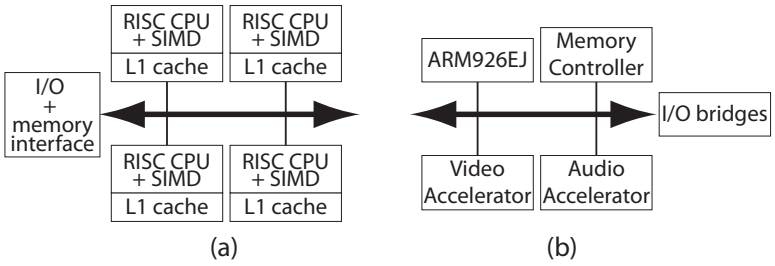


Figure 1.2: MPSoCs: (a) Lucent Daytona (Homogeneous); (b) ST Nomadik SA (Heterogeneous).

It is interesting to note that, although tailored to the requirements of the application, the Daytona configuration is similar to the one of a multi-core.

Aware of the differences between multi-cores and MPSoCs, we remark again that the focus of this thesis will be devoted on both these two categories without making any distinction between the architectural and application-oriented characteristics of the processors. Indeed the issues tackled in this thesis affect both the processors families alike.

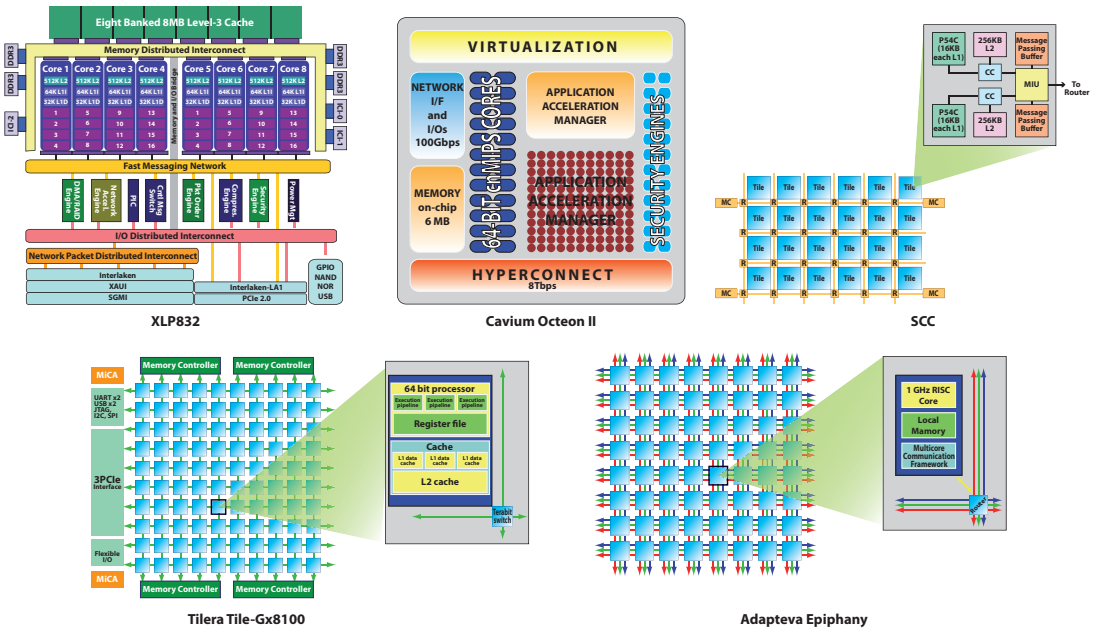


Figure 1.3: MPSoC examples.

In the next future it is expected the number of cores on the same chip will greatly increase

to encounter the demand for higher performance. Multiprocessors on chip will transform in many-core heterogeneous processors. In Fig. 1.3 we present some typical chip multiprocessors recently appeared on the market. On the network side, where packet processing is important, there are the Cavium Octeon II which features up to 32 MIPS cores, and the Broadcom/NetLogic XLP II. On general-purpose side we find the Tilera Tile-Gx8100 with 100 identical core and Adapteva Epiphany that is very reminiscent to Tilera's. It has 64 less powerful cores and manually managed cache memory. It is designed to maximize floating point horsepower with the lowest possible energy footprint. Finally Intel SCC that is a platform for many-core software research. It has 24 dual-core tiles arranged in a 6x4 mesh and each core is a P54C CPU (see Appendix B for more details).

We conclude this section showing in Fig. 1.4 the vastity of applications the multiprocessor chips are used for.

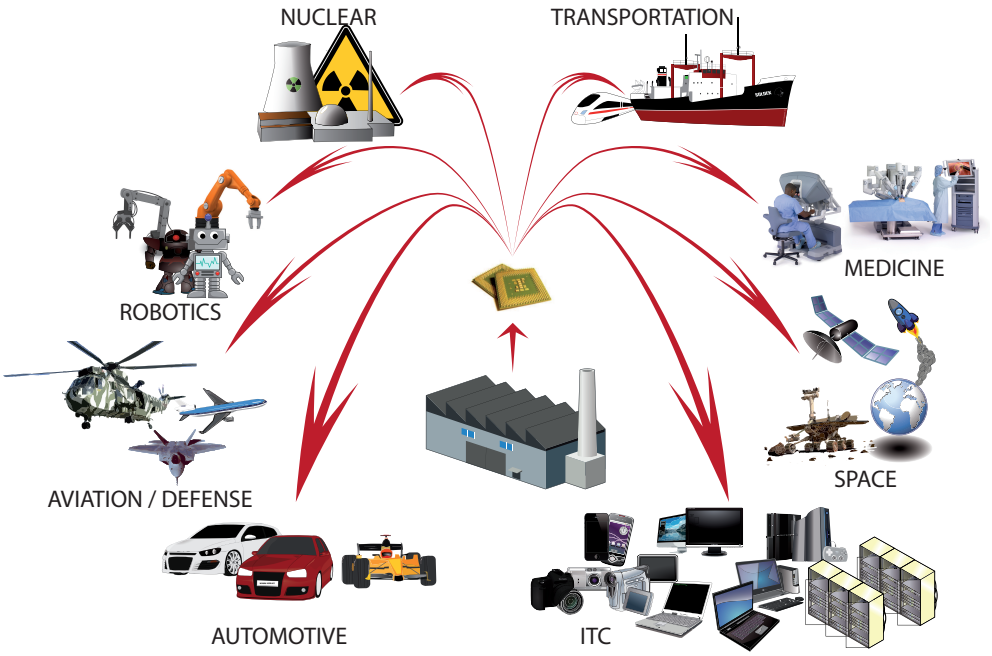


Figure 1.4: MPSoC utilizations.

1.2 Motivations

Multiprocessors on chip are playing an increasingly important role in the global economy. They are the core of nowadays and next generation computing platforms. Multiprocessors appear in

a widespread market area ranging from consumer electronics and communication products to high performance computing devices. As an example, Fig. 1.5 shows the information and communication technology chain. Here we can find different devices commonly used in everyday life and containing multiprocessors chip, but also huge web and data centers.

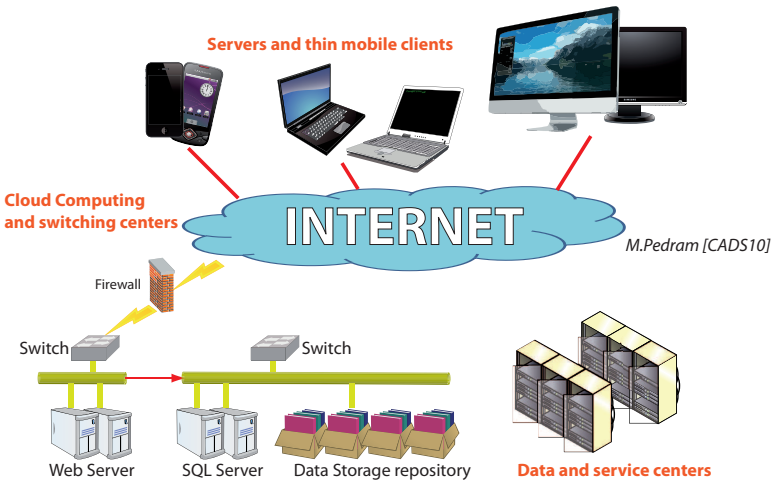


Figure 1.5: ITC Network.

Smartphones, as well as laptops and tablets, are used by an exponentially increasing number of people. The International Data Corporation[1] (IDC) (7) shows that the worldwide smartphone market grew 54.7% year over year in the fourth quarter of 2011 (4Q11). The total smartphone shipment volumes reached 491.4 million units in 2011, up a strong 61.3% from the 304.7 million units in 2010. Although there was a slowdown from 2010 (+75.7%), IDC expects double-digit growth for the foreseeable future. According to the latest research from Strategy Analytics (8) [2], the number of smartphones in use worldwide surpassed the 1 billion units in the third quarter of 2012, after only 16 years from the first Nokia smartphone appeared on the market. However, Strategy Analytics forecasts that the next billion will be achieved in less than three years. Fig. 1.6 shows the increasing trend of smartphone shipments.

According to a study conducted in December 2012 by IDC 122.3 million tablets will be sold in 2012, rising to 172.4 million units in 2013 and 282.7 million units in 2016. The NPD

[1] IDC is the premier global provider of market intelligence, advisory services, and events for the information technology, telecommunications and consumer technology markets.

[2] Strategy Analytics Strategy Analytics, Inc., a global research and consulting firm, focuses on market opportunities and challenges in the areas of Automotive Electronics, Digital Consumer, Virtual Worlds, Wireless Strategies, Tariffs and Enabling Technologies.

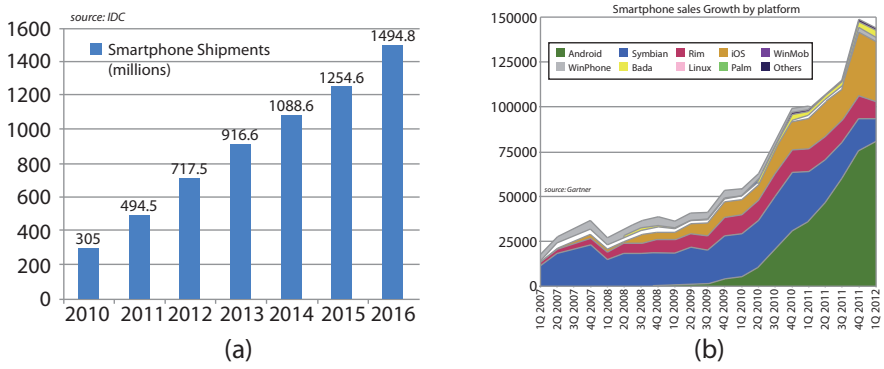


Figure 1.6: Smartphone shipments: (a) shipments forecast, (b) shipments per platform.

DisplaySearch Quarterly Mobile PC Shipment and Forecast Report (10) adds that in 2013, for the first time, tablet shipments are expected to reach more than 240 million units worldwide surpassing the notebook shipments (207 million units) that encountered a decrease of 8% in the last quarter of 2012. Nevertheless, notebook shipments will be stimulated by the emerging market. Fig. 1.7 shows the aforementioned trends.

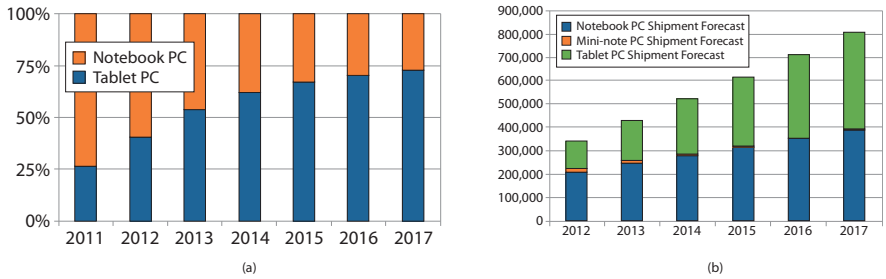


Figure 1.7: Mobile statistics: (a) tablets vs. notebooks shipments, (b) tablets and notebooks shipment trend.

The 2012 Cisco Connected World Technology Report [1] highlights that Global mobile data traffic will increase by 18 times more from 2011 to 2016. In particular smartphone and tablets traffic will be respectively 50 and 62 times greater in 2016 than they are now, the 71% of mobile data traffic will be dedicated to watching videos on portable devices by 2016 and smartphones, laptops, and other portable devices will drive about 90 percent of global mobile data traffic by 2016 (130 Exabytes of worldwide data traffic in 2016).

Internet and the cloud computing paradigm – the practice of using a network of remote

[1]Cisco Connected World Technology Report is based on a study commissioned by Cisco and conducted by InsightExpress, a market research firm based in the United States in 2012.

servers hosted on the Internet to store, manage, and process data, rather than a local server – constitutes a link between the mobiles products and the high performance computing platforms (e.g. data and service centers). Indeed, the increasing adoption of smart devices, combined with mass connectivity, high-speed broadband networks and cloud computing paradigms will drive increased adoption of streaming services resulting in the continuing rapid growth of data and service centers. This data is also confirmed by statistics. According to a Cisco study (11), global data center traffic will grow 4-fold, reaching a total of 6.6 zettabytes annually by 2016. Data center traffic will continue to dominate internet traffic for the foreseeable future, but the nature of data center traffic is undergoing a fundamental transformation, since the two-thirds of the total traffic will be dominated by global cloud traffic, the fastest-growing component of data center traffic. IDC forecasts that the total number of U.S. data centers will reduce from 2.94 million by 2012 to 2.89 million by 2016 because of the evolution of information technologies to the cloud. However, the size of data centers will increase significantly, growing from 611.4 million square to more than 700 million square feet in 2016. A trend that could be explained by the development of large-capacity data centers.

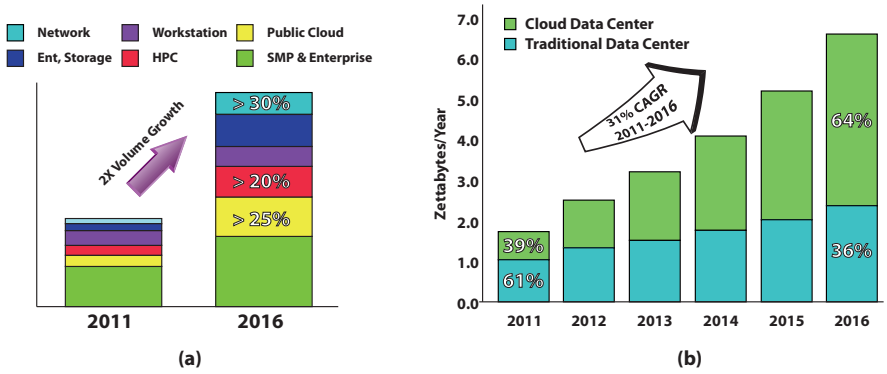


Figure 1.8: Forecast of Data Center: (a) Data Center Processor growth (CAGR), (b) Data Center traffic growth (11).

All the data previously presented show the exponential increase of computing platforms and devices containing multiprocessors chip (one for smartphones, ten of thousand for data centers). In particular we referred to few devices belonging to the information and communication technology area, but the products that exploit multiprocessors includes many others belonging to the aviation, automotive, medical, defence, space, industrial, rail, telecommunications, marine and civil nuclear area.

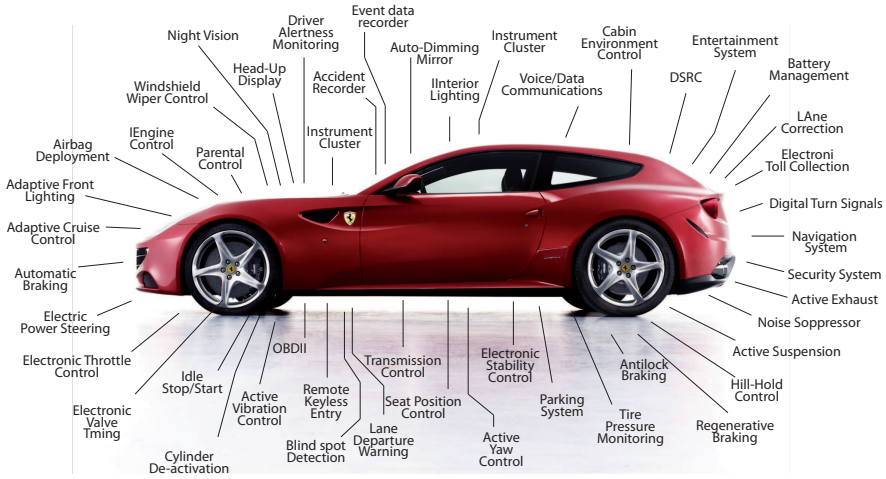


Figure 1.9: Control systems in high-end cars (12).

As an example, multiprocessors are widespread in automotive to face the challenge of increasing performance, and, at the same time, reducing costs and dimensions (e.g. power usage, electromagnetic compatibility, printed circuit board area and wiring issues). Indeed, multiprocessors may reduce considerably the number of Electronic Control Units, ECUs, present on a vehicle, which grew up above 70 for high-end cars (see Fig. 1.9). This solution, on one side improve performance increasing the average throughput and the computational power; on the other side it reduces costs, eliminating redundant hardware, reducing and uniforming software components, and simplifying the final software validation according to the safety standards.

This introduces also some challenges in the software development. Indeed, it is important that safety critical functions could run alongside non-safety critical functions without their safety characteristics being compromised. Moreover, the software must be able to manage resource sharing and the parallel running of different operative systems (because the different functions may be best served by different operating systems). The solution, already used in aviation, is virtualization technology, that is splitting the software in portions (managed by microkernels) each one acting as an independent virtual machine, granting exclusive access to the configured system resources (13).

Other examples are the machine used for robotic surgery and artificial limbs (14) (15) controlled by multi-core processors in medicine, the control unit on a airplane or the processing unit inside a PLC in automation industry.

Power consumption is a key issue for all these computing devices. Focusing on the Infor-

mation and Communication Technology (ITC) industry, global consultants Gartner (16) estimated that in 2007 it accounted about 2% of the total global CO_2 emissions, but this percentage is expected to grow in the future, despite some modest achievements in energy efficiency. The International Institute for Sustainable Development (IISD)(17) reports that the ITC sector already represents the 8 per cent of global electricity consumption and this is predicted to grow to 10-12 per cent of all electrical consumption in the next decade. However, according to SMARTer 2020 report (18), ITC can play a role in reducing annual emission, enabling an abatement of the 16.5% of the total emissions in many end-use sectors (e.g. transportation, agricolture, buildings, manufacturing, power, and service). Nevertheless, Fig. 1.10 shows that, despite of the potential benefits offered by ITC, its power consumption will grow reaching 2.3% of global emissions in 2020.

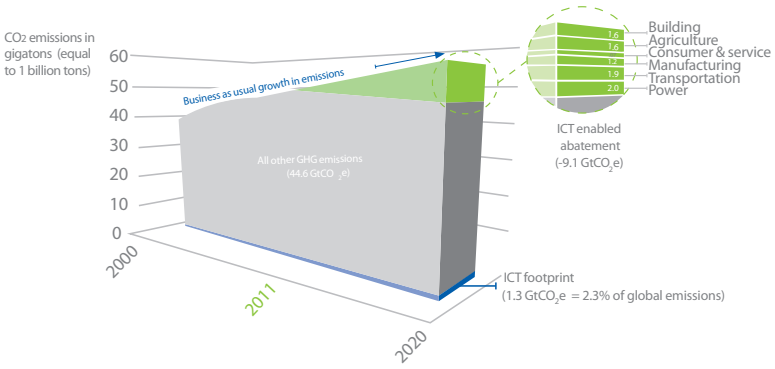


Figure 1.10: ITC emissions and potential benefits of ITC for other sectors emissions (18).

This power consumption explosion has its main causes in the increasing diffusion of these devices (mainly in developing countries) and in the rising performance demand. This latter, in particular, translates in a manufacturer competition to accomodate the market demand, making more powerful processors, but trying to maintaining the power consumption limited.

At first, manufacturers managed this situation by exploiting the miniaturization of the chip components and then, by using parallelism (the passage from single-core to multi-core allows the improvement of the throughput reducing the power consumption).

Nevertheless, the power consumption problem is still unresolved. The miniaturization improve the performance of our systems at the cost of higher power consumption and higher power densities – the power consumed per area – on the chip. As a consequence on-chip temperatures dramatically increase. In large multiprocessors, the power, unevenly consumed on

the chip, generates high temperature variations across the die or “hot spot”. Here the temperature reaches harmful values that strongly undermines the reliability of the MPSoC. Nowadays temperature is a key issue for manufacturers and it represents a crucial limit for processors development. In literature, this issue is commonly referred to as “Thermal Wall” and it affects all the devices containing multiprocessors.

Another recent issue related to high temperatures and power consumption is the so called “Utilization Wall”. This issue rises when the chip contains a high number of cores, but only a subset of cores can be activated at the same time due to power and temperature reasons. Roughly speaking chips contain too many transistors that cannot be supply with power at the same time. Thus, some of them must be left unpowered – or dark, in industry parlance – while the others are working. The phenomenon is known as *dark silicon*. Actually chips are not suffering for this issue, but provisions show that we will soon enter in the dark silicon era and mobile devices will experience the problem first due to the growing performance demand and the extreme power constraints. In the next future the combination of miniaturization and increasing number of cores on the same chip (19) (20) will result in an augment of dark silicon as shown in Fig. 1.11.

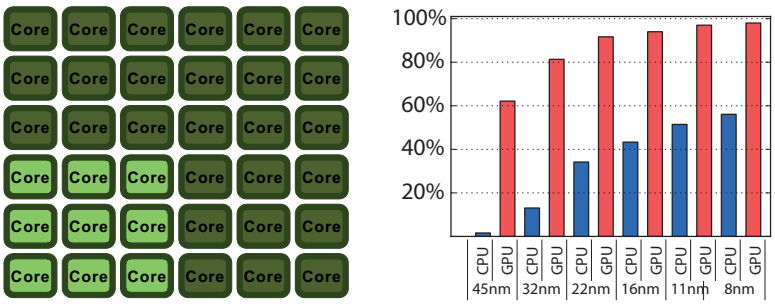


Figure 1.11: Forecasts of percent dark silicon across technology nodes (21).

We have already remarked that “Thermal Wall” and “Utilization Wall” are issues that affect all computing devices. However, it is worth to note that these problems have stronger impacts on some devices rather than others.

As an example data centers are very sensitive to the thermal issue. Here, the huge number of cores running at the same time to perform calculations produces a great amount of heat which must be dissipated in order to avoid undesired computing arrests. In such systems the 50% of power is consumed for feeding the complex cooling infrastructure, dramatically impacting on costs and environment (22).

Also mobile devices are sensitive to temperature because of the constraints on cooling systems. The limited power budget as well as the reduced size of the devices make the active cooling impracticable and shrink the surfaces available for heat dissipation complicating the architecture design. Moreover, in the next future, the limit on power budgets will prevent the simultaneous use of all processors at the same time making the mobile devices more susceptible to the "Utilization Wall".

1.3 Thesis contributions

Multiprocessor (or MultiCore) Systems-on-Chip (MPSoC) are the core of nowadays and next generation computing platforms. In this thesis, two main issues, related to such processors, will be considered, since they are crucial in limiting their development. These issues, which has been introduced in the previous section (and will be deeply treated in the next chapter), are referred in literature as "Thermal Wall" and "Utilization Wall". The first concerns the damaging effects of high temperatures on chips, whereas the second refers to the impossibility of fully exploiting the computing power of the processor due to the limitations on power and temperature budgets. The central aim of this thesis has been searching and developing efficient and reliable control solutions for maximize performance, limiting, at the same time, temperatures and power consumptions. Model Predictive Control (MPC) schemes are the main tools we used for implementing our control algorithms. Exploiting the predictions computed by a dynamic model of the system to be regulated, MPC controllers solve an optimization problem subject to constraints in order to find the optimal control decisions for the future intervals. The capability of handling constraints in a systematic way, maximizing at the same time a previously defined cost function, makes MPC very attractive for this application.

The main contribution of this thesis is the development of a distributed MPC controller for managing on-chip temperatures. The basic idea behind this solution consists in forecasting the chip area in which temperature will violate the critical temperature limit and manage to avoid overheating. The main tools are frequency and voltage "knobs" that allows the controller to reduce the speed of each core. As consequence, the power consumption reduces and the temperature decreases as well. In literature, solutions founded on this idea already exist, but all of them exploit centralized MPC schemes, i.e. schemes where a unique problem is solved to determine the optimal speed of all cores. The solution implemented in this thesis differs from the previously mentioned one because it is distributed: each core decides its frequency

exploiting also the information coming from the neighbor cores. This solution has comparable performance, but ensures two big advantages: first the system is more reliable because the control algorithm is split on all cores, second the computational complexity is considerably reduced. Indeed, it is known that the computational complexity of a MPC problem exponentially increases with the number of cores, whereas the distributed one linearly scale.

The dynamic model used for predictions plays a fundamental role in MPC schemes. Accuracy and simplicity are the main characteristics a model should posses. Accuracy for having exact behavior forecasting, and simplicity to improve controller efficiency by reducing computational complexity. In this thesis some techniques to obtain a model with such properties have been studied. In particular, the model has been obtained by:

1. solving a distributed ARX identification problem;
2. solving a H_∞ optimization problem;
3. using a proper orthogonal decomposition (POD) approach;

More in detail, the first two approaches can be used to update the model at run-time when the prediction error is unacceptable.

An important methodological contribution of this work derives from the study of the controller feasibility for thermal system context. This problem is usually disregarded in the specific literature on the thermal control of MPSoC, even though it is extremely important for guaranteeing the respect of temperature constraints at each time instant. In case of infeasibility the controller may lose its authority on the system, resulting in dangerous situations. In this thesis we proved that centralized and distributed MPC schemes are always feasible for a generic class of thermal systems. In order to cover all possible cases we used a thermal model described by Partial Differential Equations (PDE). However, a model based on PDE cannot be incorporated in a MPC algorithm due to its complexity. Time and spatial discretizations can be used to find a simple and accurate model, but the new control scheme can result infeasible. Whereas in the centralized case the feasibility loss could happen, in the distributed solution its unavoidable. In order to guarantee the property in the distributed case, we developed a complex two-layer control scheme where a Safety controller supervises the distributed MPC solution. The methodological analysis also provided an interesting property that allows the simplification of the controller design. It permits to reduce the number of temperature constraints from an infinite to a finite number. Indeed, the control problem should maintain the temperature of

every infinitesimal volume element of the chip under the threshold. Thanks to this property the same result can be obtained by constraining the temperature of a finite number of points corresponding to the chip sources, i.e. the cores.

Beside the two-layer control solution previously mentioned, we developed other complex solutions which use as basic ingredient the distributed MPC scheme.

We developed a fully distributed controller able to manage the temperature and the energy consumption of a MPSoC. Each core has a local energy mapper and a local MPC thermal controller. The energy mapper allows each core to set its frequency in order to maximize energy saving, preserving performance loss within a tolerable bound. The thermal controller trims this frequency if the temperature reached is too high. More in detail, the energy mapper algorithm reduces the core frequency if the executing task is memory-bound, i.e. involves extensive memory use. In this case reducing the core speed does not lead to execution time overheads because memory access speed is the limiting factor.

Another control solution addresses the "Utilization Wall" issue for mobile devices such as tablets and smartphones. Even though the "Utilization Wall" issue affects all computing devices, our solution it has been designed specifically for mobile devices for two reasons: first, the effects of the "Utilization Wall" will hit mobile devices due to the tighter constraints on power and temperature, and second, the quality of service perceived by the user depends on the responsiveness rather than the average throughput. The basic idea of this control solution, called computing sprinting, consists in running all cores only for short time intervals (in order to remain below the critical temperature). Indeed, the chip is designed to dissipate the heat of a subset of cores switched on at the maximum speed. If all the cores run together the chip will melt. The distributed MPC thermal controller intrinsically guarantees the sprinting functioning, maximizing performance (i.e. the cores speed) at the same time. However, the proposed solution provides another MPC control layer which manages the thermal capacity of the chip. It guarantees a sprinting window every fixed period allowing the critical tasks – the deterministic task with hard real time deadlines necessary for the correct functioning of the system – to be executed at the maximum speed. In this way the controller can manage mixed-criticalities systems. We refer to this solution (innovative in MPSoCs literature) as guaranteed re-sprinting solution.

The last solution proposed realizes a communication-aware MPC thermal controller. Starting from the centralized MPC solution it has been possible to modify the control algorithm in order to establish a communication between two cores. In other words we constrained two or

more cores to have the same frequency implementing a message passing requirement. This solution also allows the controller to dynamically choose which cores must have the same frequency.

We developed these solutions by using the Matlab/Simulink environment. First, the accurate model to simulate the real system has been generated using a finite element technique. Then, we developed the control and identification algorithms necessary for implementing the control schemes. We used different toolboxes to simplify the operation (MPC Toolbox, Hybrid Toolbox, Yalmip, CVX). The distributed MPC controller algorithm has been implemented in C/C++ language in view of a future implementation on a real Intel Single-chip Cloud Computer (SCC) containing 48 P54C Pentium cores. The C/C++ code version also allows us to estimate the execution time necessary to solve a single control problem obtaining information on the computational overhead and complexity.

The results shown in this thesis have been carried out within the team dealing with Thermal Control of Systems-on-Chip (Prof. Luca Benini, Dr. Andrea Tilli, Dr. Roberto Diversi) at Department of Electrical, Electronic, and Information Engineering ”Guglielmo Marconi” (DEI) of the University of Bologna and in collaboration with Professor Emanuele Garone of the Service d’Automatique et d’Analyse des Systémes (SAAS) at the Universitè Libre de Bruxelles.

1.4 Thesis Overview

The thesis is organized as follows.

In the chapter 2 the “Thermal Wall” and “Utilization Wall” issues are contextualized. Then, the main solutions proposed in literature to manage these issues are shown.

In chapter 3 some theoretical basics useful in further chapters will be given. Some knowledge on optimization problem with constraints will be introduced before focusing on MPC theory. Here the main components of a MPC scheme will be presentd going into deep with feasibility and stability issues. In the second part the computational complexity for large scale systems will be treated by showing the benefits of distribution.

In chapter 4 the distributed MPC solution is presented. This correspond to the basic solution that will be used in most of the complex control solutions mentioned in the previous section. In the first part the focus will be devoted to highlight the importance of the model for MPC accuracy. In this context some methods are shown to obtain accurate and reduced order models of the system (distributed ARX identification, proper orthogonal decomposition (POD) and

conservative identification (H_∞)). Then, the centralized and distributed MPC control schemes will be accurately described showing the strengths of the latter solution. Finally the feasibility property will be proved for centralized and distributed controllers.

In chapter 5 some complex control schemes, which use the distributed MPC solution as basic element, are presented. First, a distributed scheme able to manage the temperature and save energy through a higher layer energy mapper. Second, a two-layer controller (centralized and distributed) able to guarantee the controller feasibility trough the use of a safety layer based on switch controllers. Finally a modified solution of the basic controller able to guarantee message passing capabilities.

In chapter 6 the solution implemented for the "Utilization Wall" problem will be presented. Although this solution could be part of the previous chapter, we preferred to hold a separated chapter for it because of the wideness of the topic.

Finally in Chapter 7 the conclusion are drown. Moreover, the future development will be considered.

The aim of the appendices is to add useful information on the work. Appendix A give some hints on optimization theory and multi-parametric programming. Appendix B gives some details on the technique used to implement the accurate model used as real system in simulations, and some notions on power consumption of multiprocessors. Appendix C contains the Matlab/Simulink and C/C++ code to implement MPC control algorithms.

Bibliography

[1] A. Jerraya, W. Wolf, *Multiprocessor Systems-on-Chips (Systems on Silicon)*, Morgan Kaufmann, Aug. 2004. 1, 3

[2] K. Popovici, F. Russeau, A. Jerraya, M. Wolf, *Embedded Software Design and Programming of Multiprocessor System-on-Chip: Simulink and System C Case Studies* , Springer, Mar. 2010. 1

[3] R. Kumar, D.M. Tullsen, N.P. Jouppi, P. Ranganathan, *Heterogeneous chip multiprocessors*, Computer, Vol.38(10),32-38, Nov. 2005. 3

[4] B. Ackland, A. Anesko, D. Brinthaupt, S. J. Daubert, A. Kalavade, J. Knobloch, E. Micca, M. Moturi, C. J. Nicol, J. H. ONeill, J. Othmer, E. Sackinger, K. J. Singh, J. Sweet, C. J. Terman, and J. Williams, *A single-chip, 1.6 billion, 16-b MAC/s multiprocessor DSP*, IEEE J. Solid-State Circuits, vol. 35(3):412423, Mar. 2000. 3

[5] A. Artieri, V. DAlto, R. Chesson, M. Hopkins, and M. C. Rossi, *NomadikOpen Multimedia Platform for Next Generation Mobile Devices*, 2003. technical article TA305. http://www.st.com 3

[6] http://www.adapteva.com/

[7] http://www.idc.com/ 6

[8] http://www.strategyanalytics.com/ 6

[9] *Cisco Visual Networking Index Forecast Projects 18-Fold Growth in Global Mobile Internet Data Traffic From 2011 to 2016*, Feb. 2012,
http://newsroom.cisco.com/press-release-content?type=webcontent&articleId=668380

[10] H. Himuro, R. Shim, *Quarterly Mobile PC Shipment and Forecast Report*, Jan. 2013, http://www.displaysearch.com/ 7

[11] http://www.cisco.com/ *Cisco Global Cloud Index: Forecast and Methodology, 20112016* 2012, White paper,
http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns1175/Cloud_Index_White_Paper.pdf vii, 8

[12] http://www.cvel.clemson.edu/
http://www.cvel.clemson.edu/auto/systems/auto-systems.html vii, 9

[13] S.S. Thiebaut, M. Gerlach, *Multicore and virtualization in automotive environments*, Oct. 2012,
http://www.edn.com/design/automotive/4399434/Multicore-and-virtualization-in-automotive-environments 9

[14] K. Eaton, *Your Smartphone Is An Artificial Limb* Aug. 2011,
http://www.fastcompany.com/1775278/your-smartphone-artificial-limb 9

[15] W. Xiao, H. Huang, Y. Sun, Q. Yang, *Promise of Embedded System with GPU in Artificial Leg Control: Enabling Time-frequency Feature Extraction from Electromyography*, in Proc. IEEE Conf. Eng. Med. Biol. Soc., Vol. 1, pp. 69266929, 2009. 9

BIBLIOGRAPHY

[16] C. Pettey, *Gartner Estimates ICT Industry Accounts for 2 Percent of Global CO2 Emissions*, Apr. 2007, http://www.gartner.com/it/page.jsp?id=503867 10

[17] B.St. Arnaud *Using ICT for Adaptation Rather Than Mitigation to Climate Change*, Oct. 2012, www.iisd.org/pdf/2012/com_icts_starnaud.pdf 10

[18] http://gesi.org *GeSI SMARTer2020: The Role of ICT in Driving a Sustainable Future* Dec. 2012, http://gesi.org/SMARTer2020 vii, 10

[19] J. Held , J. Bautista , S. Koehl, *From a Few Cores to Many: A Tera-scale Computing Research Overview*, 2006, White paper, Intel, http://www.intel.com/content/dam/www/public/us/en/documents/technology-briefs/intel-labs-tera-scale-research-paper.pdf 11

[20] K. Asanovic , *The Landscape of Parallel Computing Research: A View from Berkley*, tech report UCB/EECS 2006-183, Dept. Electrical Eng. and Computer Science, Univ. of Calif., Berkeley, 2006. 11

[21] H. Esmaeilzadehy, E. Blemz, R.St. Amantx, K. Sankaralingamz, D. Burger, *Dark Silicon and the End of Multicore Scaling*, in Proc. ISCA, pp. 365-376, June 2011. vii, 11

[22] Greenpeace, *Make IT Green*, Mar. 2010, http://www.greenpeace.org/international/Global/international/planet-2/report/2010/3/make-it-green-cloud-computing.pdf 11

Chapter 2

MPSoCs Issues and Solutions

In this chapter the main MPSoCs issues tackled in this thesis are presented. First, a brief introduction highlights the technology walls encountered by processors vendors from the birth of the first processors. Then, the issues of today MPSoCs and some solutions present in literature are shown.

2.1 Processors issues from the beginning

A computer is an electronic device designed to accept data, perform prescribed mathematical and logical operations at high speed according to a set of instructions, and display the results of these operations. The first fully electronic general-purpose computer ENIAC (Electronic Numerical Integrator and Computer), introduced in the 1946, was a huge machine that contained 17.468 vacuum tube and required teams of people to operate. However, it was the invention of transistors that revolutionized the computing device world, strongly accelerating their development.

The transistor effect was discovered in 1947 at Bell's Lab and for this discovery John Bardeen, Walter Brattain, and William Shockley were awarded with the Nobel Prize in Physics in the 1956. The first silicon transistor was produced in Texas instruments in the 1954 and the first MOS at Bell Labs in 1960. Transistors replaced the bigger, heavier, fragile, and more power consuming vacuum tubes (used to amplify and switch signals), becoming the building block for all modern electronics and the foundation for microchip. Their importance was also remarked in 2009 when the invention was named an IEEE Milestone. De facto, transistors have introduced the third revolution for civilization: the information revolution (after the agri-

culture and industrial revolution). As machines have incremented man's mechanical power by simplifying and making more comfortable everyday jobs, so information technology extended man's intellectual power simplifying the storage, processing, transmission and exploitation of information. In the last half of the 20th century, science has become a productive force (of information): service jobs are more common than jobs in manufacturing or agriculture.

Nowadays, transistors are present in almost all devices storing, transmitting, displaying, or manipulating information. On a silicon chip we can find many thousands or even billions of transistors. Commonly, the number of transistors, i.e. the *transistor count* on a device is used as a metric for integrated circuits complexity. Microprocessors represent the most complex application of transistors, containing billions of them. Fig. 2.1 represents the number of transistors contained in a microprocessors respect to time. As it is possible to see the number of on-chip transistors doubles every eighteen months. This trend is better known as Moore's Law, named after Gordon Moore (Intel co-founder), in a paper of the 1965, stated that the number of transistors on a chip will roughly double each year (1) (in 1975 he refined this to every two years).

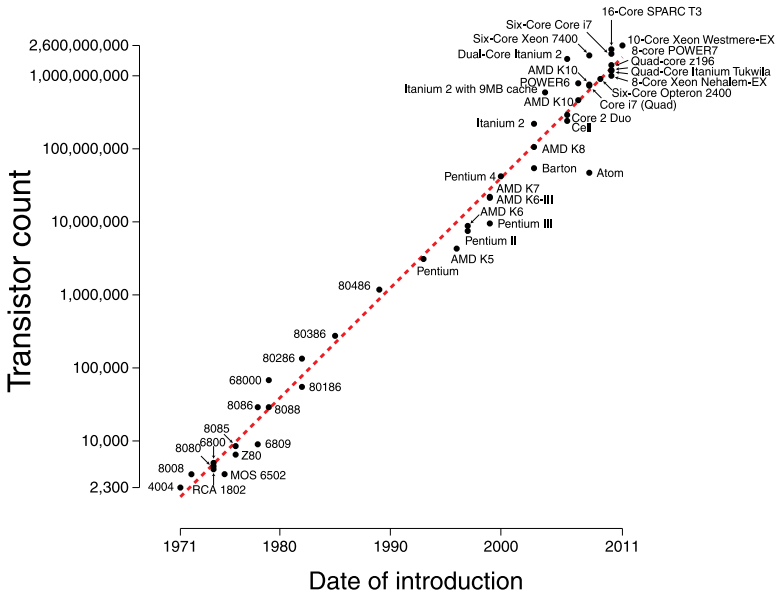


Figure 2.1: Transistor count 1971-2011.

The Moore's Law born as a simple observation of the future trend of transistors density on chips, but it became accepted as a reference for all processors vendors. Though it refers to the transistors size, in literature it is common to find different versions of the law describing

correlated technological trends that arise as consequence of the original one (e.g. chip costs, power density, clock scaling, silicon area, storage costs, ...).

The history of the microprocessor begins with the birth of the Intel 4004 in 1971, the first commercially available microprocessor. It consisted of 2300 transistors with pMOS technology (10 μm) and a clock rate of 740 kHz (2). From that moment, led by Moore's Law, microchip manufacturers started a "rush" for improving processors performance. The aim was to accomodate the market demands and seize the leadership position in a profitable and fast moving sector. To better understand the progress made in this market it is enough to think that if the transportation industry had kept the same pace of microprocessors industries, today we could have traveled from New York to London in about a second for roughly a few cents (3). After more than forty years, Moore's Law still holds, despite of many "brick walls" encountered and successfully circumvented. At the time of the first processors the main issue was the dimension of the programs limited by the size of the computer's memory; nowadays, power and temperature are the main issues. In the follows we show the main reasons that push chips vendors to move from a world dominated by single processors to one dominated by multiprocessors on a chip.

2.1.1 The "Power Wall"

The previously mentioned rush for processors performance improvement started in the early 70's with the birth of the first microprocessor and its trend is accurately described by Moore's Law. The improvements to make single processors computation faster were primarily technology driven. The first remarkable step was the transition from the nMOS bipolar logic to the, still in use, CMOS in the 80's (4). The main reasons CMOS technology became the most used technology implemented on chips regarded the noise immunity and the low static power consumption (5). More in detail, the CMOS structure (a nMOS and a pMOS in series) allows the components to draws significant power only during the ON/OFF (close/open) switching transition. At the contrary nMOS logic normally have some standing current even when not changing state, resulting in a much higher waste of heat and power. Moreover, the energy required for a logic switching (the energy necessary to charge the transistor capacitance) depends on the square of the supply voltage, therefore if voltage scales (possible only in CMOS transistors) also power and heat scale. This advantages were well understood by Robert Dennard that in a paper in the 1974 (6) postulated the scaling theory: the MOSFET transistors (nMOS or

pMOS) density, operation speed and energy efficiency will grow proportionally to the degree of miniaturization. In other words smaller transistors switch faster at lower power.

The Dennard scaling theory is at the base of Moore's Law and it drove miniaturization in the industry until now, enabling computing devices to be portable (7). Table 2.1 show the implication of Dennard's theory assuming a scaling factor α for each technology generation. Therefore, since every technology generation has commonly a $\alpha = 1.4$ scaling factor (depen-

Parameters	Scaling Factor
Device dimensions tox, L, W	$1/\alpha$
Doping concentration Na	α
Voltage V_{dd}	$1/\alpha$
Current I	$1/\alpha$
Capacitance $\varepsilon A/t$	$1/\alpha$
Delay time per circuit $V_{dd}C/I$	$1/\alpha$
Power dissipation per circuit $V_{dd}I$	$1/\alpha^2$
Power density $V_{dd}I/A$	1
Integration density	α^2

Table 2.1: Dennard's scaling theory

dent on industry strategies) the transistors dimensions reduces of the 30% ($1/\alpha = 0.7\times$), the area shrinks of the 50% ($1/\alpha^2 = 0.5\times$), and the transistor density doubles. At the same time circuit performance increases by about 40% ($\alpha = 1.4\times$ frequency increase that corresponds to $1/\alpha = 0.7\times$ delay reduction) and the supply voltage is reduced by 30% ($1/\alpha = 0.7\times$) to meet the condition of having a constant electric fields according to Dennard's theory. As a result, active power ($P = C \cdot V_{dd}^2 \cdot f$, where C is the capacitance being switched per clock cycle, V_{dd} is the supply voltage, and f is the switching frequency) reduces by 50% ($1/\alpha^2 = 0.5\times$) (8). Therefore, considering the same chip area , in every technology generation transistor density doubles and circuit becomes 40% faster at the same power consumption. These data are summarized in Fig. 2.2.

However, miniaturization is only one of the factors that in these years concurs to improve performance. Other important factors are microarchitecture techniques and cache memory improvements that we briefly introduce without going into details.

Microarchitecture techniques refers to the way in which the resources are organized and the design techniques used in the processor to reach the target cost and performance goals

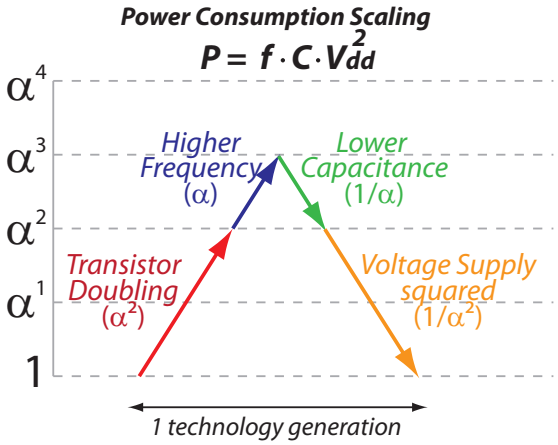


Figure 2.2: Dennard's implications (23).

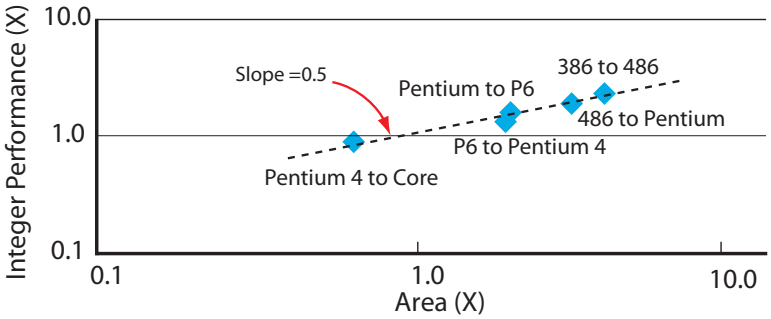


Figure 2.3: Pollack's Rule.

(e.g. pipelining, branch prediction, out-of-order execution, and speculation). Microarchitecture techniques exploit the growth in available transistors to improve performance. The performance increase by microarchitecture alone is empirically described by Pollacks Rule (9), which states that performance increases as the square root of the number of transistors or area of a processor. In other words, if the number of transistors doubles, a new microarchitecture delivers only a 40% performance increase (see Fig. 2.3). Anyway, it is important to notice that developers do not modify the microarchitecture every technology generation.

Also memory (DRAM) architecture influences performance. Following the Moore's Law, memory density doubles every two years, but performance improves more slowly (see Fig. 2.4a), resulting in a bottleneck for the overall system performance. However, according to (8) the slow improvements depends on economical choices rather than technological impediments. Market demanded high density and low cost memories at the expense of speed (see Fig. 2.4b). Although it was technically possible to have a memory as fast as processors, manufacturers chose

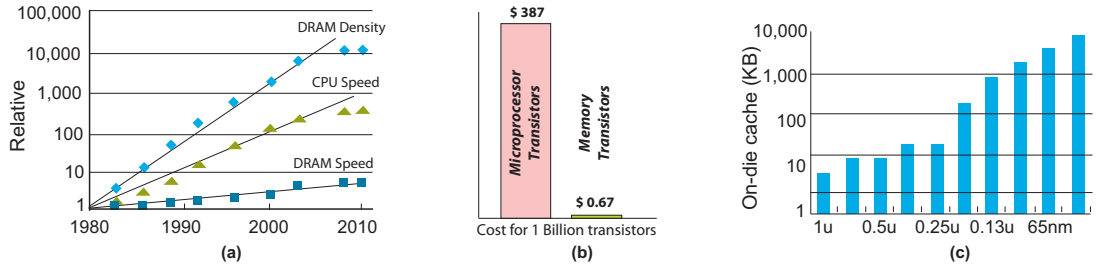


Figure 2.4: DRAM density (a), costs (b), and performance (c) (8) (10).

a more economically viable path to reduce the speed gap between processors and memories. The idea was to introduce some small, expensive but fast memories close to the processors, containing copies of the most frequently used data. Nowadays caches are organized in three-layer hierarchical structures and it is common to find caches on the die (built "subtracting" transistors to cores). This because, comparing the performance benefits of increasing the transistors on a core with the ones obtained by reducing the speed gap between memories and processors, often it results more convenient the second choice (see Fig. 2.4c).

Over the past two decades, both scaling, and microarchitecture and memory hierarchy techniques allows a 1000x microprocessor performance improvement. However, this trend slowed down due to physical scaling limits. Scaling is not a "free lunch" (12) anymore. Power consumption, reliability and variability constitute barriers to the development of microprocessors (13). In mid-2003 manufacturers hit the "Power Wall". The benefits of the scaling theory subsist if the electric field is kept constant, which means that when transistors scale, also the supply voltage (V_{dd}), the SiO_2 insulator layer and the threshold voltage of transistors (V_{th}) have to scale to deliver circuit performance. Unfortunately all these three elements encountered limitations. As the threshold voltage reduces, subthreshold leakage current – the current flowing between source and drain when the transistor is OFF (i.e. open or equivalently in subthreshold region) – exponentially increases (it depends on $e^{(V_{GS}-V_{th})}$ where V_{GS} is the voltage between gate and source). As shown in Fig. 2.5, the greater is the scaling, the greater is the portion of power due to leakage. Thus, to keep leakage under control, the threshold voltage must be limited, resulting in performance degradations. Solutions commonly adopted by circuit designers to alleviate subthreshold leakage issue are stacked gates, body bias, and sleep transistors.

Also the oxide insulating layer plays an important role in power consumption. Indeed, as the gate dielectric gets thinner (as consequence of the transistor scaling) the performance improves, but, at atomic dimensions, the gate leakage current – the current flowing between

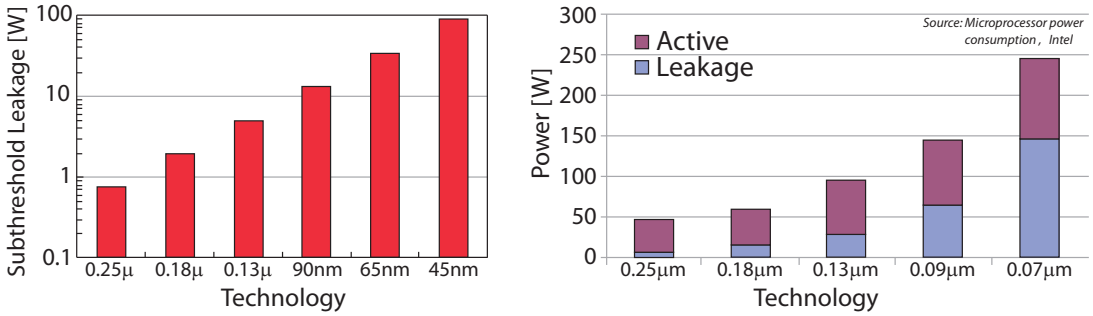


Figure 2.5: Leakage power trend: (a) subthreshold leakage power (14), (b) Active vs leakage power.

gate and drain – increases, resulting in an augment of the power dissipated. This phenomena is called *tunneling*. Circuit designers mitigated this problem by using high K dielectrics.

Due to transistors atomic dimensions, threshold voltage limits and constraints to meet the processor performance targets (e.g. noise immunity, cell state stability, ...), the supply voltage is approaching a lower bound. As a result (see Fig. 2.6) the power consumption increase over generations, reaching a maximum limit in mid-2003. The maximum clock frequency reached was around 4GHz (15).

Table 2.2 summarizes how these limitations modify the Dennard's implications showed in Table 2.1.

Parameters	Scaling Factor
Device dimensions tox, L, W	$1/\alpha$
Doping concentration Na	α
Voltage V_{dd}	1
Current I	1
Capacitance $\varepsilon A/t$	$1/\alpha$
Delay time per circuit $V_{dd}C/I$	$1/\alpha$
Power dissipation per circuit $V_{dd}I$	1
Power density $V_{dd}I/A$	$1/\alpha$
Integration density	α^2

Table 2.2: Post-Dennard's scaling theory

Considering α the scaling factor, area reduces by 50% ($1/\alpha^2 = 0.5\times$), frequency increases by 40% (α), capacitances scales by 30% ($1/\alpha$), but voltage does not scale (1), leading to a

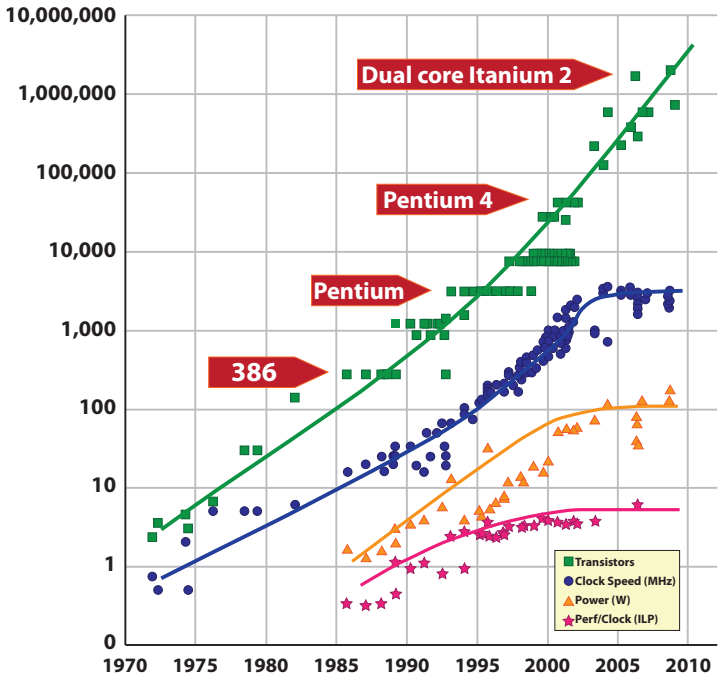


Figure 2.6: Intel CPU trend (12).

power consumption that doubles every technology generation (α^2). Fig. 2.7 summarizes these data.

All these limitations and the increasing design complexity (due to the features added to improve performance as multi-threading, hyper-threading, speculative execution, ...), pushed companies to search for new solutions. Again Moore's Law helped designers, indeed, despite power and clock frequency limitations, the number of transistors continues to climb, providing cheaper transistors and the possibility of including multiple cores on the same chip. In 2004 AMD put on the market the Opteron processor which signed the switch from single-core to multi-core paradigms. However, it is worth to highlight that the embedded multiprocessor systems-on-chip history began earlier than general purpose multicore, in 1990 with Daytona MPSoC.

The basic idea of multiprocessors was that if average throughput cannot be improved by increasing speed, due to power budget limitations, then it could be increased by parallelizing the operations, that is executing more tasks on slower multiple cores at the same time. This solution allowed the designers to increase the data throughput, reducing the voltage and the frequency. We can summarize the multiprocessors in three words: simpler, slower, efficient

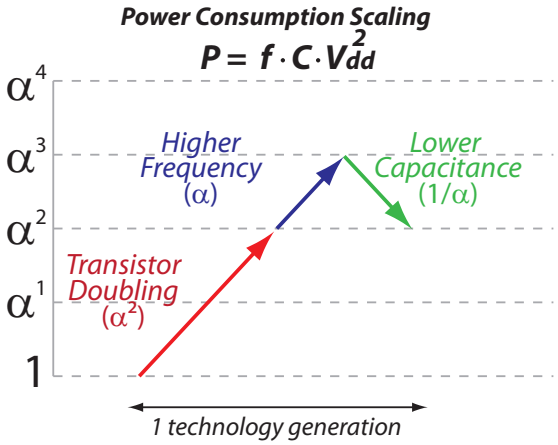


Figure 2.7: Dennard's failure implications.

(i.e. they consume less power).

However, differently from what one can expect, doubling the number of cores does not mean doubling the performance, this because most existing software is single-threaded and parallelizing compilers have limitations in static analysis and/or lack of information at compile-time. Therefore only a small fraction of codes can be automatically parallelized.

Nevertheless, the number of cores on the same chip increases fast and researchers already forecast thousands of cores in the next future. Researchers predicted that the number of cores on a silicon chip will double with every technology generation (9), bringing MPSoCs to the many-core paradigm – processors with a high number of cores, where the multi-processor techniques are no longer efficient due to congestion problems. Fig. 2.8 shows the increasing demands for MPSoCs in a wide range of market sectors, particularly for smartphones in ICT sector.

2.1.2 The "Thermal Wall"

The diminishing return on performance and the increasing power consumption in traditional scaling approach, led designers to integrate on the same chip multiple cores. The primary aim was improving performance per Watt exploiting parallelism. However, multiprocessors benefits comprise also the possibility of setting the performance (frequency, voltage, on/off) of each core individually, and distributing the load in order to reduce heat across the die, improving reliability and leakage.

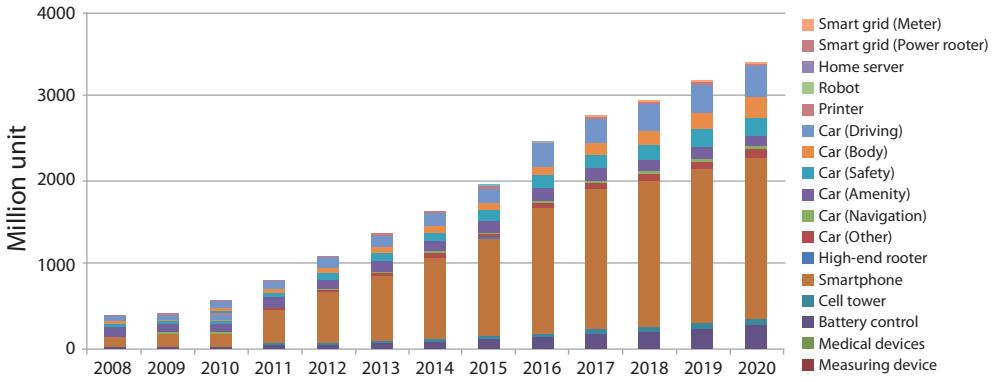


Figure 2.8: MPSoCs Demand.

This last sentence introduces a new problem which is directly connected with power consumption and constitutes the actual key limiter for multiprocessors development. This problem is the temperature and its detrimental effects on reliability, leakage, and performance.

It is worth to note that the temperature issue is not new and the First Law of Thermodynamics is the proof that the problem is always existed. It states that energy is conserved, which means that it can be converted from one form to another, but neither created, nor destroyed. Therefore, the power consumed by transistors is converted into heat.

However, the actual power consumption of chip coupled with the area scaling (due to transistors shrunk) result in an extraordinary power density increase which involves dramatically high temperatures. This trend is perfectly illustrated in the famous Intel forecast of the 1999 shown in Fig. 2.9. Following the actual trend, in the next future, researchers expects unimaginable power densities similar to the one produced by a nuclear reactor or a rocket nuzzle.

Moreover, to complicate the issue, the switch to multiprocessors technology introduced new thermal challenges. The large chip used as support for cores as well as the difference of workloads executed and power consumed on cores generate temperature variations across the die that contribute to worsen the reliability, the performance, and the cooling efficiency of the chip. These temperature variation may manifest as hot spots or temporal and spatial temperature gradients. The former are small areas of the cores heavily utilized where the consumed power density/temperature is higher than in other part of the chip. The latter represent thermal cycles, that are temperature fluctuating along time, and temperature maps characterized by non-uniformity on the chip area.

High temperatures adversely affect performance by reducing the computing speed of the chip. This because temperature degrades carrier mobility – the mobility of electrons and holes

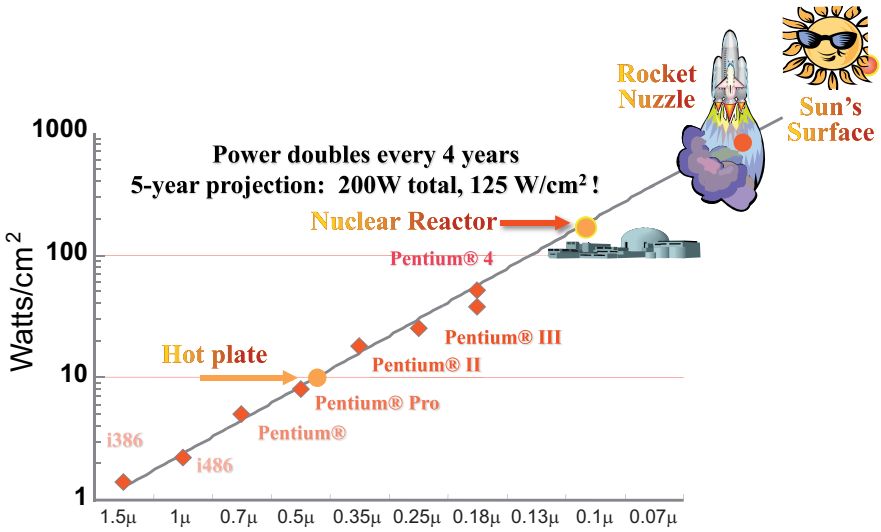


Figure 2.9: Power density trend (16).

in semiconductors under an electric field.

The temperature influences leakage power, that exponentially increases with it, making the problem of power consumption even more evident.

Moreover, high temperatures strongly affect reliability and lifetime of chip components (17), (18), (19), (20). Components lifetime exponentially reduce with temperature, accelerating failure mechanisms as electro-migration, stress migration and dielectric breakdown. The time to failure – the time interval from when a component is put into service to when the component fails – decreases as function of $e^{E_a/(k \cdot T)}$ according to the Arrhenius relationship, where E_a is the activation energy, i.e. the energy necessary for the failure mechanism to occur, k is the Boltzmann's constant ($1.38 \times 10^{23}\ J/^{\circ}K$), and T is the absolute temperature.

These challenges occur with high steady-state temperatures, but they are rarely dangerous under $150^{\circ}C$. At lower magnitude, temperature gradients are the major causes for reliability loss. Repeated changes of temperature in time (thermal cycles) reduce considerably the mean time to failure of metallic structures and cause package fatigue and plastic deformations that are proportional to the magnitude and the frequency of the cycles. On the other side, changes in temperature along space accelerate negative bias temperature instability (NBTI) and hot carrier injection (HCI) effects. Both cases refer to the breakage of $Si-H$ bonds happening at the Si-channel/gate-oxide interface (Si/SiO_2) of MOSFETs during transistors operation (when the gate bias is negative and temperature are elevated). When this happens, hydrogen diffuses

away from the interface, leaving behind the so called dangling bonds or interface traps. As a result the absolute threshold voltage increases, the carrier mobility diminishes, and the drain current reduces (21) (22). Spatial temperature variations also have other undesired effects: clock skew problems induced by circuit delays that increase with temperature (indeed local resistances are proportional to temperature), and cooling efficiency decrease since the power spent to feed the cooling system is proportional to the highest temperature measured.

All these issues constitute the "Thermal Wall", the new major limiter for high performance processors. In the next future power density is expected to grow due to area scaling and transistor power consumption. In order to ensure the correct functioning and the lifetime of devices, an accurate thermal management is necessary.

2.1.3 The "Utilization Wall"

Looking forward to the future, Moore's Law scaling will continue to improve transistors density, but with small performance improvements. Power wall and energy efficiency constraints will force designers to deeply exploit parallelism and customizations (8). Integrating on the same chip multiple cores (eventually heterogeneous) and ad-hoc hardware to support computation can improve considerably the average-throughput of the microprocessor. Researchers expect that the number of cores integrated on a chip will double every technology generation reaching a number of hundreds or thousands of processing units. However, this solution is expected to fail in following the historical exponential performance rate due to the energy constraints (25).

With the failure of Dennard's scaling theory, on every technology generation, the frequency and the number of transistors increase, but, unfortunately, the same happens to the power consumption (see Fig. 2.7). As a result, the heat generated on the chip cannot be entirely dissipated. Cooling infrastructures are limited by cost (e.g. data centers consume the 50% of the energy to power cooling systems), or by physical constraints (e.g. in mobile phones active cooling is impracticable), obliging the system to use only a fraction of the chip transistors at full speed, at one time. In literature this issue takes the name of "Utilization Wall" and it represents a big concern for chip designers. Although it is expected that the problem will arise most clearly in the next future, some effect of this wall are indirectly present in modern processors. As an example Intels Nehalem "turbo mode" power off some cores in order to run others at higher speeds (24).

The rate of utilization of a chip will drop exponentially (2× per generation). According to the experiments conducted in (25), with a 22nm technology only the 79% of the die can work at full frequency, and this percentage drops to less than the 50% at 8 nm. The fraction of the chip that remains underclocked, is called "dark silicon". The term was coined in 2009 by Mike Muller who wrote in (26): *"Without fresh innovations, designers could find themselves by 2020 in an era of "dark silicon," able to build dense devices they cannot afford to power."*

Fig. 2.10 shows the power density and the dark silicon trends according to the data from (8) and ITRS.

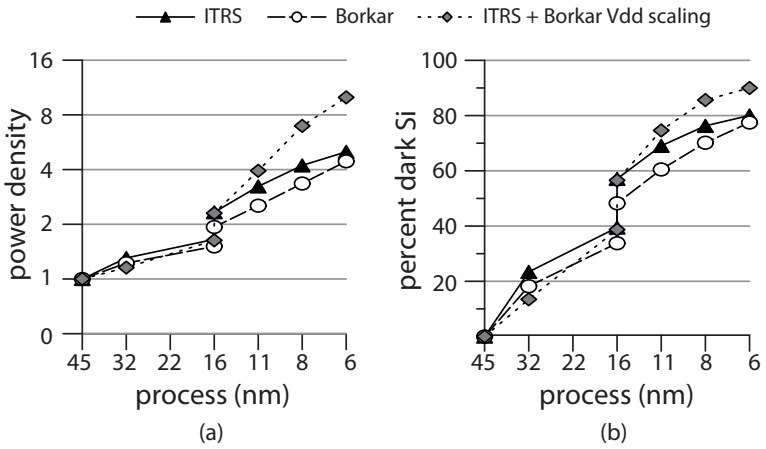


Figure 2.10: (a) Power density trend; (b) Dark silicon trend (8).

In order to improve device performance the "Utilization Wall" issue need a careful management. In particular, mobile platforms seem to be the most susceptible to "Utilization Wall" effects. Battery capacity and heat dissipation limits strongly reduce the energy available to the microprocessors to run the core at full speed, de facto limiting performance. In this case a thermal management must guarantee thermal safety and performance maximization (not only as average throughput, but also as Quality of Service perceived by the user).

Before concluding this section it is worth to note that we define dark silicon not only the silicon completely unused, but also the silicon rarely used or used at a lower frequency.

2.2 Related Works

The "Thermal Wall" and the "Utilization Wall" are crucial limits for the development of multiprocessor (or multi-core) systems-on-chip. We have already seen in the pervious Section that

the first concerns the damaging effects of high/variable on-chip temperatures, whereas the second refers to the impossibility of fully exploiting the computing power of the processor due to the limitations on power budgets and cooling systems. This Section contains some of the most effective solutions found in literature to solve these issues.

2.2.1 Solutions for thermal issue

As processors scale, the power density collected on the chip exponentially increases, resulting in high temperatures and even high variations across the die and hot spots that undermine the processor reliability and efficiency. Moreover, the exponential dependence of leakage on temperature aggravates the problem even further. The research community and leading electronics companies have invested significant efforts in developing thermal control solutions for computing platforms.

In general we can group the approaches used to tackle the thermal issue in two big families: *Static Thermal Management techniques (STM)* and *Dynamic Thermal Management techniques (DTM)*. The former increase the power dissipation of the chip (the so called "thermal design power" or TDP) by acting on architectural design (heat sink, fan, floorplanning, ...). The latter, instead, reduce the operating temperature at "run time" through the use of dynamic voltage and frequency scaling (DVFS), thread migration/scheduling and clock gating.

The increasing power density coupled with cooling infrastructures physical and economical constraints pose severe and sometimes insuperable challenges to STM approaches. Whereas in the past STM techniques were enough to guarantee worst-case power dissipation, nowadays cooling systems are unable (or inaccessibly expensive) to completely remove the heat under these conditions.

Studies showed that air cooling systems are approaching the dissipation limit, set to $1.5W/mm^2$ in (27) even though establishing an exact value is difficult due to the great amount of variables and parameters affecting heat dissipation. This pushed researchers to find more efficient, but often more complex and more expensive solutions as liquid cooling. The water has an higher heat capacity that, compared to conventional air cooling techniques, allows the on-chip temperature to reduce up to $45^{\circ}K$ and, since leakage power is exponentially related with temperature, a 12.8% average leakage power reduction (28).

Mobile phones poses great constraints on the cooling infrastructure: the small dimensions of the device limit the heat convectively dissipated and prevent the use of active solutions.

However, cooling techniques are only a subset of the STM techniques. In principle the chip area could be increased in order to reduce the power density, but costs and architectural constraints make this solution inadmissible. Thermal-aware floorplanning manages chip components position in order to maximize performance and energy reduction and at the same time minimizing temperature. This can be done by decreasing wire length and maximizing the distance between hot units. According to (29) the performance loss with thermal-aware floorplanning is less than 2% respect to the 6% - 21% obtained with DTM techniques.

Nevertheless, while mechanical cooling solutions and STM techniques remain the primary mechanisms for dealing with thermal wall, they are costly, unwieldy and not completely solve the problem. As the power density increase the thermal problem must be addressed at all levels of the design cycle. Today and tomorrow thermal management techniques will be a mix of STMs and DTMs and these latter will play an increasingly primary role. As power density increases also the maximum power consumption/temperature increases. However, the average power consumed is considerably lower than the maximum one, and this gap is expected to become larger in the future. As an example the Alpha 21264 processor has a maximum power dissipation of 95W, but the average power dissipation was found to be only 72W for typical applications (31). DTM techniques allows chip designers to focus on average rather than worst-case thermal conditions, i.e. cooling systems can be designed to handle the average-case, letting the MPSoC managing the emergencies through dynamic techniques. This is extremely important since studies revealed that cooling costs increase exponentially with temperature (30). Fig. 2.11a shows the increasing disparity between the maximum and average power consumption, while Fig. 2.11b highlights the exponentially increasing dependence of cooling costs on thermal dissipation.

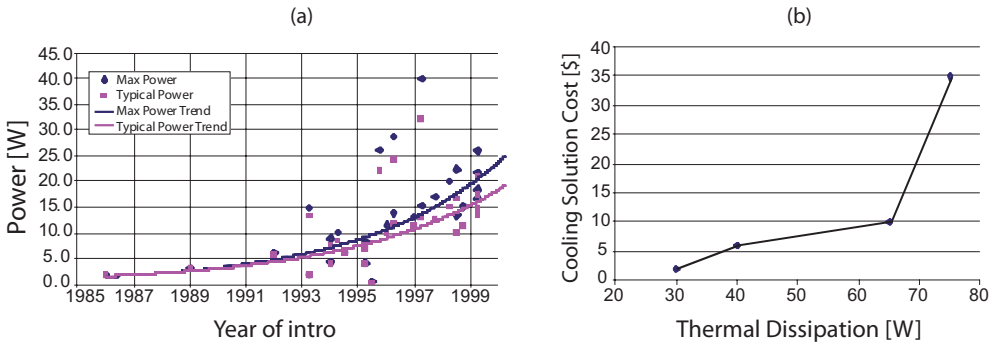


Figure 2.11: (a) maximum vs. average power consumption; (b) cooling costs vs thermal dissipation (30).

DTM techniques have received a lot of attention in recent years and this thesis will focus on these latter. In the follows we briefly summarize some interesting solutions to the problem of dynamic thermal management.

According to (31), the key goals of DTM techniques are: (i) to provide inexpensive hardware or software responses, (ii) that reliably reduce power, (iii) while impacting performance as little as possible. In simple words this means maximizing performance while maintaining the chip below a safe temperature. DTM techniques can be classified in two categories: the *temporal DTM* solutions and the *spatial DTM* solutions (32) (33).

The former controls the temperature by reducing the amount of energy dissipated. In other words they slow down or arrest the cores in order to make the chip cool down. Dynamic Voltage and Frequency Scaling (DVFS), Instruction Cache Throttling and Fetch-Toggling belong to this category. It is worth to note that usually these techniques imply performance degradation, since cores run at a reduced speed.

The second category comprises all the techniques that control the temperature by distributing the activities over the chip area. In other words the idea is to cool down the hot areas of the chip (i.e. areas with high power consumptions) by moving the workload in colder areas (i.e. areas with low power consumptions). As an example, assume a dual core chip where one core is running a cpu-bound task (high power consumption), whereas the other is running a memory-bound task (low power consumption). As a result the first core will reach a higher temperature than the second. Then, the thermal manager should schedule the future tasks in order to balance the temperature on the chip assigning to the second core cpu-bound tasks and memory-bound tasks to the first one. Migration at granularity of functional unit, pipeline, cache bank, execution clusters, thread migration (or equivalently core hopping) belong to this category.

It is worth to note that there exist also hybrid and hierarchical DTM solutions which comprise multiple of the previously mentioned solutions. The former use the DTM techniques in a gradual way from the less to the most aggressive to minimize performance loss, whereas the latter select the most appropriate technique from a set of possible candidates.

Among all these DTM techniques we focus on DVFS ones. DVFS techniques reduce power consumption by adjusting the clock frequency and/or the supply voltage of cores dynamically. Because the power consumption is proportional to the frequency and the square of the supply voltage, reducing this latter yields more significant power saving. However, in order to have stable operation, supply voltage has to be reduced only if frequency is reduced. According

to the relationship shown above, DVFS is able to reduce temperatures by managing power consumption with a performance loss linearly proportional to frequency.

A general DVFS technique is triggered by an event as it could be a thermal sensor on the die. Then, the DTM algorithm computes, according to some rules, the frequency and voltage to apply to the system. In general frequency and voltage "knob" are made available by chip manufacturers. In the research community, DVFS schemes can be implemented using different algorithm and rules.

Early approaches focused only on temperature management, ignoring performance optimization. The most common techniques used in today microprocessors are threshold based: in case of temperature bound violation the frequencies of all cores were set to the minimum value. As an example, Thermal Monitor 1 (TM1) of Pentium 4 and Dual Cores inserts idle clock cycles (Thermal Throttling) when the temperature reaches a critical value. It reduces the duty cycle of the microprocessor by 50% favoring the chip cooling down. For design complexity reasons, the first DVFS approaches on multiprocessors were global, all cores were adjusted according to the same policy. These approaches can be triggered by the operating system or directly by hardware mechanisms. However, both have drawbacks particularly for multi-scale systems (many-core and 3D-integrated stacks). The former cannot safely bound the run-time temperature and it has been shown to worsen the thermal cycles and system reliability, whereas the latter bring major performance degradation or even application failures (32) (34). Moreover, as the cooling costs and the on-chip temperatures increased, DTM (and hence DVFS) techniques became more aggressive at the expense of performance. The new DTM techniques must take into account also the issue of maximizing performance.

For these reasons (i.e. minimizing performance loss and improving reliability), DTM algorithms started exploiting most advanced solutions belonging to feedback control theory. The first approaches studied were based on classic PID algorithms (34) (35) (36). These algorithms permit to apply a frequency/voltage proportional to the thermal emergency, taking into account the prior history of the system.

More recently, studies focused on more sophisticated algorithms based on optimal control theory (37). In particular Model Predictive Control (MPC) methods look very promising due to their capability of dealing explicitly with performance and state-space constraints (i.e. temperature bounds) (38) (39). MPC schemes use a system model to predict the future temperature and find the optimal control decision by solving a constrained optimization problem for one or more control steps in the future (more details will be given in Chapter 3). If an accurate

thermal model is available, MPCs can guarantee a reliable temperature capping in any working condition. In the follows two recently proposed MPC solutions are presented.

In (38), Zanini et al. implemented a MPC scheme in order to make smoother the DVFS approach and maximize performance. The thermal behavior of the chip has been modeled using a finite element techniques. They split the chip in two layers (one representing the junction silicon and one representing the copper of the heat spreader) and again each layer has been split in cubic cells. To each cell they assigned an equivalent RC electric circuit where R and C are equivalent respectively to a thermal resistance and a thermal capacitance. The final thermal model is:

$$t_{k+1,1:2n} = A \cdot t_{k,1:2n} + B \cdot f^{\alpha}_{k,1:p} + W \tag{2.1}$$

where $t_{k,1:2n}$ are the temperatures of the $2n$ cubic cells at time k (i.e. the model state), A is the state matrix, $f_{k,i}$ represents the frequency of the $i-th$ core at the time k, α expresses the dependence between the power consumption and the frequency (i.e. $p_{k,j} = f^{\alpha}_{k,j}$ where $1 \leq \alpha \leq 2$), B is the input matrix, and W is an offset vector considering the room temperature effect in the heat spreading process. This model is used from the MPC algorithm at each sampling interval to forecast the future temperatures of the chip in the next h intervals. The MPC scheme can be stated as,

$$min \sum_{k=0}^{h-1} \left(f^{\alpha}_{k+j,1:p} - r^{\alpha}_{k,1:p}\right) \cdot S \cdot \left(f^{\alpha}_{k+j,1:p} - r^{\alpha}_{k,1:p}\right) \tag{2.2a}$$

$$s.t.$$

$$0 \leq t_{k+1,1:2n} \leq t_{max} \quad \forall k = 0, \ldots, h-1 \tag{2.2b}$$

$$0 \leq f^{\alpha}_{k+1,1:p} \leq f^{\alpha}_{max} \quad \forall k = 0, \ldots, h-1 \tag{2.2c}$$

where (2.2a) is the objective (or cost) function, S is a weight matrix, and $r_{k,i}$ expresses the required operating frequency of the $i-th$ core requested by a higher level software as the operating system. $(f^{\alpha}_{k,i} - r^{\alpha}_{k,i})$ represents the error between the offered power and the required one. Notice that minimizing this value means maximizing the performance. The equations (2.2b)-(2.2c) represent a set of constraints respectively on future temperatures and offered core power. $t_{k+1,1:2n}$ are the temperatures of the cells at time $k+1$, t_{max} is the maximum temperature allowed, $f^{\alpha}_{k+1,1:p}$ is the future power consumption of the p cores, and f^{α}_{max} is the maximum power allowed.

Thus, summarizing, f is the manipulable input, r comes from the OS, and t is measured with thermal sensors. At each sampling interval, the solution of this problem is the optimal

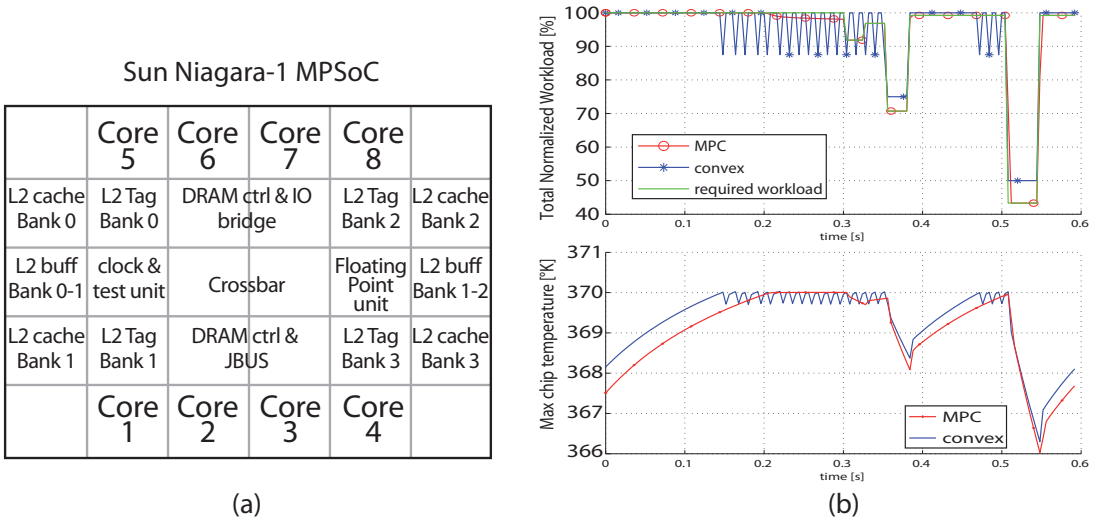


Figure 2.12: (a) Sun Niagara-1 MPSoC model; (b) MPC vs. Convex-based solution. (38).

$f^{\alpha}_{k+1,1:p}$ that maximize performance, meeting the constraints. It is worth to note that the solution of this problem returns the frequency of all cores.

From the implementative perspective, the authors built two solutions: the implicit and the explicit solutions. The first solves the optimization problem (2.2) on-line at each sampling time, requiring a great computing effort. The second solves the optimization problem off-line for all possible scenarios and store the optimal control decisions in a look-up table. This solution requires less on-line computations (it only has to detect the current scenario at each sampling instant), but a great amount of memory space.

Finally the authors test their solution on the model of a Niagara 8 cores processor. A comparison between this solution and the state-of-the-art convex-based solution shows an improvement up to $5\times$ (see Fig. 2.12).

In (39) Wang et al. present a MPC scheme that constraint both the power, and the temperature of the cores while maximizing the performance. The Fig. 2.13 shows in detail the proposed control scheme. The power monitor, the temperature sensor on each core and the system-level performance monitor of each core collect information that will be used by the controller to compute the new DVFS levels of the cores (the manipulated variables of the control loop). The new levels are applied to the cores by the DVFS modulator and the online model estimator updates the power system.

The thermal model of the system is computed with a least square identification technique.

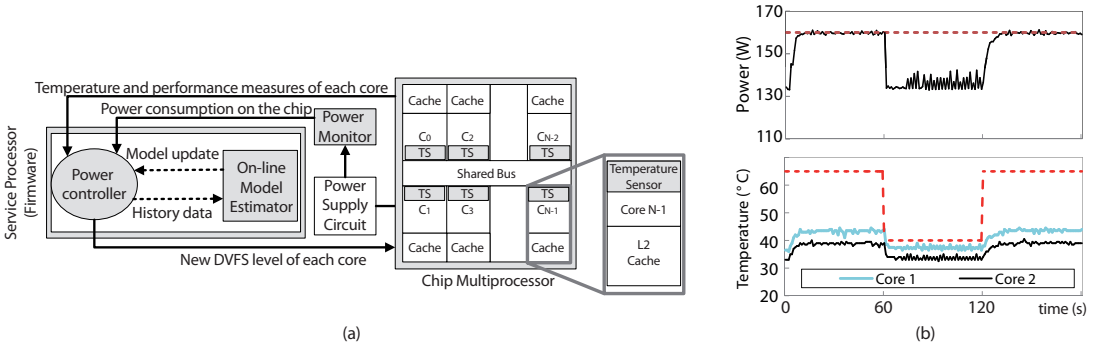


Figure 2.13: (a) Temperature-constrained power control loop for a CMP with N core; (b) power and temperature plot (39).

The final model has the form,

$$\Delta t(k) = A_t \cdot t(k-1) + B \cdot \Delta f(k-1) \tag{2.3}$$

where $t(k)$ and $f(k)$ are respectively the temperature and the frequency array containing all cores temperatures and frequencies at time k, $\Delta t(k) = t(k+1) - t(k)$, $\Delta f(k) = f(k+1) - f(k)$, A_t is the state matrix and B is the input matrix between frequency and temperature. The model assumes a linear relation between frequency and power dissipation ($p(k)$).
The MPC scheme can be stated as,

$$min \sum_{i=1}^{p} \|cp(k+1|k) - ref(k+1|k)\|^2_{Q(i)} + \tag{2.4a}$$

$$\sum_{i=0}^{M-1} \|\Delta f(k+1|k) + f(k+1|k) - F_{max}\|^2_{R(i)} \tag{2.4b}$$

$$s.t.$$

$$F_{min,j} \leq f_j(k+1) \leq F_{max,j} \quad \forall j = 1,\ldots,N \tag{2.4c}$$

$$cp(k) \leq P_s \tag{2.4d}$$

$$B_i \cdot f(k+1) < s_i \quad \forall i = 1,\ldots,N \tag{2.4e}$$

$$f_i(k+1) = f_j(k+1) \quad \forall i,j = 1,\ldots,N \tag{2.4f}$$

where N is the number of core, (2.4a) is the cost function that penalizes the power error between the total power consumption of the chip (cp) and the ideal reference trajectory of the power (ref), computed as an exponential trajectory between $cp(k)$ and P_s (this latter is the

maximum power value). The function (2.4b), instead, is the control penalty that minimizes the distance between the manipulated DVFS frequency level, $f(k+1)$ and the maximum DVFS frequency level, F_{max}. The constraint (2.4c) implies the future manipulated frequency is in the range $[F_{min}, F_{max}]$, (2.4d) imposes the total power is lower than the maximum power allowed (P_s), (2.4e) is a temperature constraint $t_i(k+1) < T_i - \delta$ reformulated according to the model equation (δ is a safe margin and s_i is a constant value dependent on T_i), (2.4e) represents optional constraints due to application or hardware requirements.

The authors also provide a study of the sensitivity to the parametric disturbance and a on-line model estimator able to update the temperature model.

For what concerns results, they implemented their solution on a real Xeon X5365 Quad Core processor and compared the results with other state-of-the-art solutions obtaining greater performance. They used also a cycle-accurate simulator (SESC) to test different chip architecture.

2.2.2 Solutions for utilization issue

Due to the novelty of the problem, the literature on this topic is limited, even though we expect it will considerably expand.

According to (23) there exist four potential approaches to deal with the challenges posed by dark silicon and the "Utilization Wall". Each of them has some benefits, but none is ideal to solve the problem. For this reason future solutions will probably incorporate all four of them.

The first solution consists in shrinking the chip size, eliminating unused dark silicon. On the one side this solution allows the designer to save time reducing chip complexity, money on silicon (the cost reduce linearly with area), and leakage (since the number of transistors is lower). However, the silicon cost after few technology generations would be only a negligible fraction of the total cost which comprises the costs for tests, marketing, sales, support, maintenance, packaging, ... Then, there will be no more incentives in investing money to pursue Moore's Law. With area shrinking hot spots will be smaller and, according to recent studies the smaller are the hot spots and the greater is the efficiency in dissipating heat with colder neighbors.

On the other hand the dark silicon could be exploited to build more competitive products on the market. Moreover, it is known that the maximum on-chip temperature is inversely proportional to the chip area, therefore, area shrinking would result in a lowering of the TDP which would

compel designers to reduce performance.

The second solution consists in using the dark silicon to build specialized cores. Because with every technology generation transistors become cheaper than power consumption, it is convenient to spend these transistors to introduce custom hardware that turns on only when necessary and that consumes less energy than a general purpose core. An example of such a chip is the UCSD GreenDroid mobile processor (40) that contains hundreds of specialized cores, called conservation cores or c-cores, instead of the dark silicon. These c-cores are automatically generated from application source code in order to save energy (see Fig. 2.14).

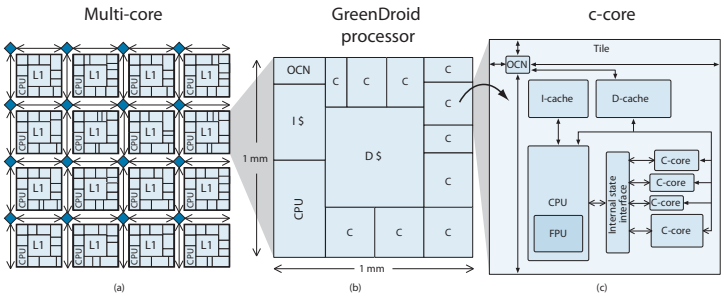


Figure 2.14: (a) Many-core processor with c-cores; (b) GreenDroid; (c) c-core (40).

The third solution consists in populating the dark silicon with homogeneous core that would exceed the power budget (i.e. the TDP) and using them underclocked or at the maximum speed for short burst. In simpler words, every technology generation, we can use the dark area to increase the number of cores. Clearly all these cores cannot run at the maximum speed all together because of the limits on the power and temperature. Then, we can tackle the problem in two different ways: distributing the power budget on the whole chip or in the time. In the first case, it is possible to ensure a safe temperature and power capping by limiting the speed of all the cores. As alternatives it is possible to use cores with lower performance but greater energy-efficiency (e.g. Near-Threshold Voltage Processor), or to use the dark space to increase the cache size, reducing the energy consumption for off-chip readings and the memory-reading bottleneck. In the second case, we can turn on all cores at the maximum speed for a short time interval in order to keep a safe temperature.

In (41) the authors exploit this approach to improve the performance of mobile devices multiprocessors. They assume to have a chip with 16 core each one consuming 1W but a TDP of only 1W. This means that the cooling infrastructure (package + heat sink) is able to dissipate

the power consumed by one core continuously active. Otherwise, if all cores are active, the chip will exceed the sustainable power budget (i.e. the TDP) resulting in a chip overheat. However, this is true only in steady-state conditions. Indeed, even though the chip generates heat faster than the system can dissipate it, the temperature will takes some time to reach the critical value. This time interval depends on the thermal capacitance of the chip that is usually very short. This fact depends on the materials used to build the package that are usually optimized for minimizing the thermal resistance from the junction to the ambient, neglecting the heat storage capabilities. The authors suggest to activate all 16 cores at full speed for the time allowed by the thermal capacity and, after that, returning in a state of nominal operation, with 15 cores in a rest condition. The approach is called computational sprinting and it proposes also the use of solid-liquid Phase Change Material (PCM) inside the package in order to increase the thermal capacity of the package (i.e. the duration of the sprint). Phase Change Materials are solid at ambient temperature and can store extra heat during the melting process, releasing it to the ambient later on, during solidification. During the process of melting the temperature remains quite constant because the thermal energy is used to break the bonds between molecules. As consequence of that, PCM allows packing in a small volume and within a small temperature gap a large thermal capacitance placed close to the silicon die.

Fig. 2.15a-b show the sprint/rest mechanisms. T_{melt} is the melting point of the PCM, while T_{max} is the maximum safe temperature.

The authors chose mobile device processors because the utilization issue is more significant in these systems due to the power requested by new multiprocessors, unsustainable with battery technology, and the constraints on heat dissipation, the small dimensions of the device limits the air flow and prevent the use of active cooling infrastructures. Another reason for this choice are the different applications and requirements respect to desktop or high computing devices. Indeed, many interactive applications are characterized by short bursts of intense computation punctuated by long idle periods waiting for user inputs, while the performance can be measured in terms of responsiveness rather than average-throughput.

From the architecture perspective the authors first show that the dangerous power-grid drops and the ripples caused by switching from sprint to rest mode can be effectively mitigated by introducing a gradual activation scheduling of the cores with $128\mu s$ total duration. Second, they show that high peak power phases, required by the sprint mode can be effectively delivered using Li Polymer battery and ultra capacitors. Moreover, the 16A peak current is not

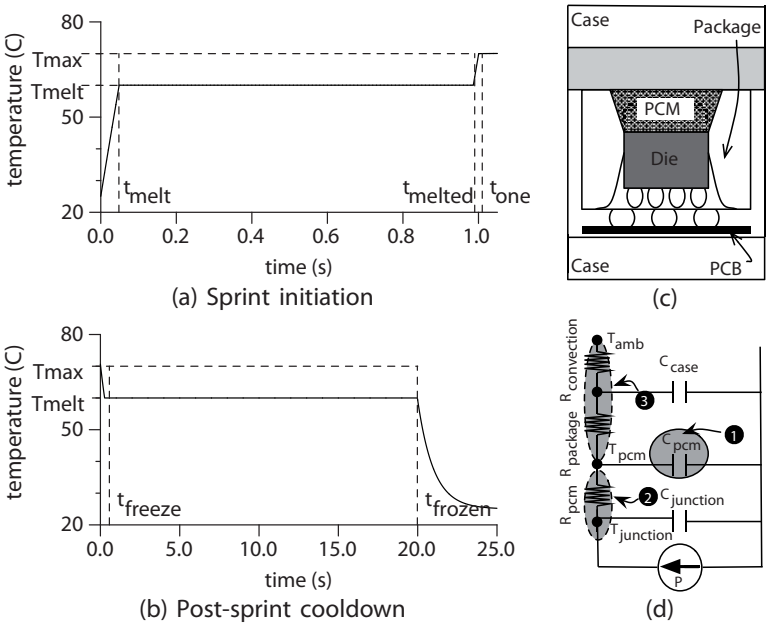


Figure 2.15: (a) Sprinting transient; (b) Resting transient; (c) Chip augmented with PCM; (d) Thermal model of the chip (41).

typical in mobile packages and would require a large number of pins to be delivered, this cost can be mitigated by using higher pin voltages combined with an on-chip voltage regulator.

In the paper, authors use 150g of PCM material to substitute the heat-spreader, augmenting its thermal capacitance to sustain 1s of continuous sprint. It is shown that the heat stored during the sprint requires a cooling period of 25s to restore the initial sprinting capability. The PCM material is modeled with a variable capacitance and both the PCM and the silicon layers are modeled as isothermal blocks.

In the same paper the authors quantify the performance gain during sprinting phases, in average benchmarks achieved a 10.2x speed-up compared to single-core execution time. The same thermal capacitance can be exploited by voltage/turbo-boosting, common in today high-end multi-processors. Indeed voltage/turbo boosting allows to speed-up the performance of single-threaded workload by increasing for a short time period the supply voltage of the core and consequently its clock frequency. The voltage supply-frequency relation is almost linear, whereas dynamic power depends on the square of supply voltage and linearly with the frequency. As consequence of that in (41) the authors demonstrate that with the same 16x sprint power budget a single-core that uses voltage-boosting can only achieve an average 2.5x speed-up on the same benchmarks (41). This is 8x slower than the computational sprint one.

For what concerns thermal management, if the sprinting applications execute enough to completely melt the PCM, the sprint is stopped by a threshold-based feedback thermal controller when the critical junction temperature is reached. Above this value the HW could be severely damaged. Safe thermal stop are enabled architecturally with fast cache flush and tasks migration, ensuring the correct software execution. Nonetheless, no possible performance optimization or re-sprinting requests are taken into account in such approach. On the one hand, whenever the critical junction temperature is reached, sprint is stopped without seeking for profitable intermediate solutions, exploiting DVFS. On the other hand, in the above stop policy, only thermal issues are taken into account, without considering which room has been left for a subsequent re-sprint.

Finally a fourth possibility is to find a semiconductor substitute for MOSFETs as for example Tunnel Field Effect Transistors (TFETS) and Nano-Electro-Mechanical switches. Indeed, even though technology will improve MOSFETs in the future, they will always be limited by leakage.

Bibliography

[1] G.E. MOORE, *Cramming more components onto integrated circuits*, Electronics, Vol. 38(8), Apr. 1965. 20

[2] http://www.wikipedia.org/ http://en.wikipedia.org/wiki/Intel_4004 21

[3] D.A. Patterson, J.L. Hennessy *Computer Organization and Design: the hardware/Software interface*, 4th edition, Morgan Kaufmann Publishers, 2009. 21

[4] Sam Naffziger *High-Performance Processors in a Power-Limited World*, Symposium on VLSI Circuits Digest of Technical Papers pp.93-97, 2006. 21

[5] http://www.wikipedia.org/ http://en.wikipedia.org/wiki/CMOS 21

[6] R.H. Dennard, F.H. Gaensslen, H. Yu, V.L. Rideout, E. Bassous, A.R. Leblanc, *Design of Ion-Implanted MOSFETs with Very Small Physical Dimensions*, in proc. of the IEEE, VOL. 87(4):668-678, Apr. 1999. 21

[7] M. Bohr, *A 30 Year Retrospective on Dennards MOSFET Scaling Paper*, Solid-State Circuits Newsletter, Vol. 12(1):11-13, winter 2007. 22

[8] S. Borkar, A.A. Chien *The future of microprocessors*, Communications of the ACM, Vol. 54(5):67-77, May 2011. vii, viii, 22, 23, 24, 30, 31

[9] S. Borkar, *Thousand Core ChipsA Technology Perspective*, DAC 07, Jun. 2007, San Diego, California, USA. 23, 27

[10] R. Fish III, *The future of computers - Part 1: Multicore and the Memory Wall*, Nov. 2011, http://www.edn.com/. vii, 24

[11] R. Fish III, *The future of computers - Part 2: The Power Wall*, Jan. 2012, http://www.edn.com/.

[12] H. Sutter, *The Free Lunch Is Over A Fundamental Turn Toward Concurrency in Software*, Mar. 2005, http://www.drdobbs.com/ vii, 24, 26

[13] S. Borkar, N.P. Jouppi, P. Stenstrom, *Microprocessors in the Era of Terascale Integration*, DATE '07, pp.1 -6, Nice, Apr. 2007. 24

[14] S. Borkar, *Gigascale Integration-Challenges and Opportunities*, Jul. 2009, http://software.intel.com/en-us vii, 25

[15] J. Parkhurst, J. Darringer, B. Grundmann, *From Single Core to Multi-Core: Preparing for a new exponential*, in Proc. ICCAD'06, Nov. 5-9, 2006, San Jose, CA. 25

[16] F. Pollack, *New microarchitecture challenges in the coming generation of CMOS Process Technologies*, Intel Corp. Micro32 conference key note, 1999. vii, 29

[17] D. Cuesta, J.L. Risco-Martin, J.L. Ayala, J. Ignacio Hidalgo, *3D thermal-aware floorplanner using a MOEA approximation*, Integration, the VLSI Journal, Vol. 46(1):10-21, Jan. 2013. 29

BIBLIOGRAPHY

[18] R. Viswanath, V. Wakharkar, A. Watwe, V. Lebonheur, *Thermal Performance Challenges from Silicon to Systems*, Intel Technology Journal, Q3, 2000. 29

[19] A. Coskun *Efficient Thermal Management for Multiprocessor Systems*, Ph.D. dissertation, University Of California, San Diego, 2009. 29

[20] *Failure Mechanisms and Models for Semiconductor Devices*, JEDEC Publication JEP122C, http://www.jedec.org. 29

[21] D.K. Schroder *Negative bias temperature instability: What do we understand?*, Microelectronics Reliability, Vol.47(6):841852, 2007. 30

[22] H. Kufluoglu, M.A. Alam *A Unified Modeling of NBTI and Hot Carrier Injection for MOSFET Reliability*, 10th International Workshop on Computational Electronics (IWCE-10), pp.28-29, 2004. 30

[23] M.B. Taylor *Is Dark Silicon Useful? Harnessing the Four Horsemen of the Coming Dark Silicon Apocalypse*, in Proc. 49ht DAC '12, pp.1131-1136, 2012. vii, 23, 39

[24] G. Venkatesh, J. Sampson, N. Goulding, S. Garcia, *Conservation Cores: Reducing the Energy of Mature Computations*, in Proc. 15th ASPLOS 2010, March 2010. 30

[25] H. Esmaeilzadeh, E. Blem, R.St. Amant, K. Sankaralingam, D. Burger, *Dark silicon and the end of multicore scaling*, IEEE Micro, Vol 32(3):122-134, Jun. 2012. 30, 31

[26] M. Muller *ARM CTO: power surge could create 'dark silicon'*, 2009, http://www.eetimes.com/electronics-news/4085396/ARM-CTO-power-surge-could-create-dark-silicon- 31

[27] P. Zhou, J. Hom, G. Upadhya, K. Goodson, and M. Munch. *Electro-kinetic microchannel cooling system for desktop computers*, In Proc. of SEMI-THERM, 2004. 32

[28] J. Kong, S.W. Chung, K. Skadron, *Recent Thermal Management Techniques for Microprocessors*, ACM Computing Surveys (CSUR), Vol. 44(3)No. 13, Jun. 2012. 32

[29] K. Sankaranarayanan, S. Velusamy, M. Stan, K. Skadron, *A Case for Thermal-Aware Floorplanning at the Microarchitectural Level*, Journal of Instruction-Level Parallelism, Vol.8, pp. 1-16, 2005. 33

[30] S.H. Gunther, F. Binns, D.M. Carmean, J.C. Hall, *Managing the impact of increasing microprocessor power consumption*, Intel Technology Journal, 2001. viii, 33

[31] D. Brooks, M. Martonosi, *Dynamic Thermal Management for High-Performance Microprocessors*, 7th International Symposium on High-Performance Computer Architecture HPCA, pp.171-182, Mexico, Jan. 2001. 33, 34

[32] P. Chaparro, J. Gonzalez, G. Magklis, Q. Cai, A. Gonzalez, *Understanding the Thermal Implications of Multicore Architectures*, IEEE Transaction on Parallel and Distributed Systems, Vol. 18(8):1055-1065, Aug. 2007. 34, 35

[33] A. Naderlinger *A Survey of Dynamic Thermal Management and Power Consumption Estimation*, Software Systems Seminar, Salzburg, 2007. 34

[34] K. Skadron, T. Abdelzaher, M. R. Stan, *Control-Theoretic Techniques and Thermal-RC Modeling for Accurate and Localized Dynamic Thermal Management*, in Proc. of the 8th International Symposium on High-Performance Computer Architecture, Anaheim, California, 2002, pp. 1728. 35

[35] M. Kadin, S. Reda, A. Uht, *Central vs. distributed dynamic thermal management for multi-core processors: which one is better?*, in ACM Great Lakes Symposium on VLSI, New York, NY, 2009, pp. 137-140. 35

[36] Z. Wang, X. Zhu, C. McCarthy, P. Ranganathan, V. Talwar, *Feedback Control Algorithms for Power Management of Servers*, in 3rd International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks, Annapolis, MD, June 2008. 35

[37] F. Zanini, D. Atienza, G. De Micheli, S.P.F. Boyd, *Online convex optimization-based algorithm for thermal management of MPSoCs*, in Proc. of the 20th symposium on Great lakes symposium on VLSI, 2010, pp. 203-208. 35

[38] F. Zanini, D. Atienza, L. Benini, G. De Micheli, *Multicore Thermal Management with model predictive control*, in Proc. of the 19th European Conference on Circuit Theory and Design, pp. 90-95, 2009. viii, 35, 36, 37

[39] Y. Wang, K. Ma and X. Wang, *Temperature-Constrained Power Control for Chip Multiprocessors with Online Model Estimation*, in Proc. of the 36th annual international symposium on Computer architecture, Austin, TX, USA, June 2009. viii, 35, 37, 38

[40] N. Goulding-Hotta, J. Sampson, G. Venkatesh, S. Garcia, J. Auricchio, P.C. Huang, M. Arora, S. Nath, V. Bhatt, J. Babb, S. Swanson, M. Taylor, *The GreenDroid mobile application processor: An architecture for silicon's dark future*, IEEE Micro, Vol. 31(2):86-95 , March 2011. viii, 40

[41] A. Raghavan, Y. Luo, A. Chandawalla, M. Papaefthymiou, K.P. Pipe, T.F. Wenisch, M.M.K. Martin, *Computational Sprinting*, In HPCA, Feb. 2012. viii, 40, 42

BIBLIOGRAPHY

Chapter 3

Model Predictive Control

In this chapter a background knowledge of MPC controllers is presented. First, the MPC strategy is explained, and the pros and cons and a brief summary of the evolution history of MPC are shown. Subsequently, the main elements composing a MPC controller are listed and a classification of the main MPC structures are reported. Finally, the common challenges affecting a MPC controller, such as feasibility, stability and computational complexity, are described with a collection of the most common techniques to solve them.

3.1 Background

Model Predictive controllers are not representative of a specific control strategy, but designate a wide range of control methods. The basic idea of the controller belonging to the MPC family is to solve, at each sampling time and starting from the current state, an open-loop optimal control problem over a finite horizon, yielding as a result the optimal control decision for the next time interval. More in detail, the controller uses a dynamic model to forecast the system behavior over a determined horizon h_p, namely the *prediction horizon*. The predicted outputs can be indicated as $y(t+k|t)$ for $k = 1,\dots,h_p$ which highlights that the values at time $(t+k)$ are estimated by knowing the measurements up to instant t. The goal of the controller is to find the future decisions sequence that optimizes a specified *objective function* (or *cost function*), eventually respecting some constraints. We called $u(t+k|t)$ for $k = 0,\dots,h_p$ and $V(x,t,u)$ the input sequence and the objective function respectively. Of the decision sequence computed by the controller, only the first control action, $u(t|t)$, is applied to the plant during the $[t,t+1]$ sampling interval. The procedure is then repeated the next sampling time with the new data

available over the new horizon $[t+1, t+1+h_p]$ (1) (2) (3) (4). This control strategy is referred to as receding horizon strategy where the term "receding horizon" is introduced to indicate that the horizon recedes as time proceeds. The receding horizon strategy previously described

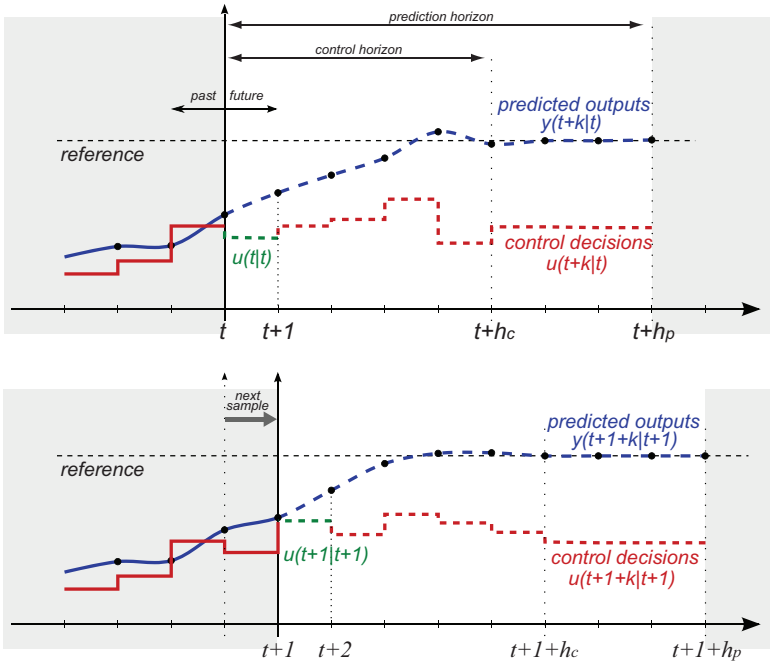


Figure 3.1: receding horizon strategy. Adapted from (4).

is shown in Fig. 3.1, where h_c represents the *control horizon*, that is the number of sampling interval over which the control decisions are computed. After h_c samples the control input remains constant to the last computed value. This strategy is commonly employed in real application as a technique for reducing the computational complexity of the MPC algorithm at the expense of the optimality of the solution (over the prediction horizon). The aforementioned mechanism belongs to the set of strategies named move-blocking strategies that refers to all those approaches where the input sequence or its derivative are imposed to be constant over several time steps in order to improve the controller efficiency. A MPC controller, on which a move-blocking strategy is applied, is usually referred to as *Move-blocking MPC scheme*. If $h_c = h_p$ the control sequence is allowed to change over all the prediction horizons. However, it is worth to note that often it is more convenient to use a $h_c < h_p$ than reducing h_p.

A very typical example for understanding how the MPC works is comparing MPC strategy with the strategy used in driving a car. According to the car characteristics (we know the behavior of our car if we take a curve at 70Km/h rather than 40Km/h) and knowing the reference

trajectory, the driver is able to manage the control actions (brakes, accelerator, and steering) in order to track the desired trajectory. On the contrary a classic PID takes decision only when an error between the current and the desired trajectory is detected, that would be analogous, referring to the driving example, to taking decision by using the rear-view mirror. Also the game of chess resembles the MPC strategy. Indeed, a player chooses the best move by forecasting in advance the next opponent moves.

In literature the terms Model Predictive Control (MPC) and Receding Horizon Control (RHC) are often used as synonyms. According to (4) a MPC is a particular case of RHC where the finite time optimal control law is computed by solving an on-line optimization problem. According to (5), instead, MPC and RHC are equivalent and the difference of their names has only historical reasons. Indeed, in principle receding horizon controllers dealt with state-space models, whereas model predictive ones with I/O models. In this thesis we assumed they are synonyms.

3.1.1 History

The history of MPC has developed in a contrary sense comparing to the other control paradigms. Indeed, MPC strategy started to receive attentions from the research community only after being profitably used in process industry applications, in the seventies. The reasons of this success were mainly due to its ability of handling, simply and effectively, hard constraints on control and states (26).

It is worth noting that the development of the modern control theory (infinite time and finite time optimal control), in the sixties, has posed the bases for the development of the MPC. Indeed also MPC deals with the resolution of optimization problems. However, while the elegant and powerful optimal control theory had little impact on control development in industries, the MPC properties fitted perfectly with the industry requirements. According to (6) the main reasons for the failure of the LQ controller were:

- the incapacity of dealing with constraints;
- the complexity of modifying and updating the solution to account for new products and aging (low flexibility);
- the unsustainable costs for developing accurate models;
- the difficulty of finding a solution (if it exists) to problem with nonlinearities;

- the relatively low exposure of technicians and control engineering to the optimal control theory.

The MPC, instead, had all the characteristics to tackle the typically complex, nonlinear, multi-variable, and constrained problems arising in the process industry (usually the most economical system functioning regions are placed extremely close to critical constraints, therefore the controller must be able to consider these constraints during the computation of the optimal solution, see Fig. 3.2b). We can reasonably say that MPC has born as an approximated alternative to the optimal LQ controls, with the aim of meeting the industry targets. The word approximated is because it has been proved that solving repeatedly a finite time optimal control problems in a receding horizon fashion yields an infinite horizon "suboptimal" controller (4) (the solution is optimal in the deterministic case, i.e. without uncertainties).

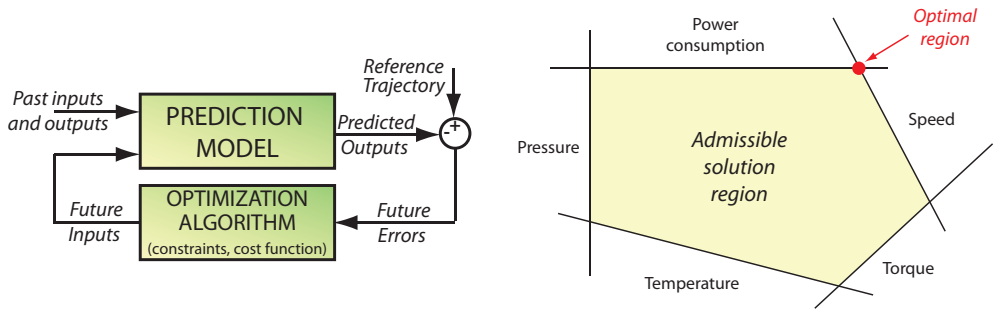


Figure 3.2: (a) general MPC scheme; (b) typical industrial working region

The first ideas regarding MPC originated in the early sixties. In the 1963 Propoi proposed a MPC solution for linear systems with hard constraints on control relying on linear programming, whereas in the 1968 Lee and Markus substantially anticipated the MPC strategy stating that:

One technique for obtaining a feedback controller synthesis from knowledge of open-loop controllers is to measure the current control process state and then compute very rapidly for the open-loop control function.The first portion of this function is then used during a short time interval, after which a new measurement of the process state is made and a new open-loop control function is computed for this new measurement. The procedure is then repeated.

However, these ideas remained unobserved in the academic world until the industry put these ideas into practice opening up new methodological issues that captured the interest of theoreticians.

From the mid seventies, in literature, the first papers related to MPC in process industry applications appeared. The first were presented by Richalet et al. in the 1976 and 1978 (9, 10) and Cutler et al. in the 1980 (11). Both proposed unconstrained MPC solutions based on quadratic performance indices and employ a model, impulse or step response based, to predict the effect of future control inputs on the system. The constraints management was realized by ad hoc solutions. The name of these two industrial MPC algorithms were IDCOM (IDentification and COMmand) the software version of the Model Predictive Heuristic Control (MPHC) approach, and DMC (Dynamic Matrix Control).

The second generation of controllers allowed engineers to manage constrained MPC. The QDMC represented the evolution of the DMC where the problem was posed as a QP problem able to explicitly deal with constraints (12). The subsequently MPC solutions, belonging to the third generation of MPC technology and developed in the nineties, increased in complexity in order to solve practical problems as the management and recovery from infeasible solutions, the distinction between hard and soft constraints, or the management of multi objectives inside the cost function. Example of such controllers are SMOC, IDCOM-M, HIECON, PCT. However, some important theoretical issues remained unsolved in these solutions. As an example the feasibility and the closed-loop stability are delicate problems that the academic research addressed ever since.

3.1.2 Advantages and disadvantages

In this section we have summarized the desired characteristics of a MPC solution.

- MPC can handle control problems where an off-line solution cannot be computed (due to nonlinearities, constraints, and uncertainties): it solves the optimization problem on-line for the current state;
- MPC can control multivariable plants also in presence of delays, unstabilities and non-minimum phase issues;
- MPC allows the systems to work in proximity of the constraints where usually the performance are maximized and the costs minimized;

- MPC intrinsically compensates dead times;
- MPC is easy and intuitive to tune (also for complex systems);
- MPC results in applying at each sample interval a simple linear control law;
- MPC is flexible and require less time, compared to optimal controllers, to be modified or adapted to new requirements (e.g. new safety regulations, new products, and new machines), environment modifications, and aging effects;
- MPC internal model can be obtained from data by identification approaches reducing the costs of model development that could be unjustifiable for small batch productions.

Among the drawbacks of the MPC one of the most critical is the need to solve the optimization problem on-line during a sampling interval. This the reason why the MPC technology has been addressed as a technology fitting well only with system with slow dynamics in which the sampling time could be maintained large. However, as the technology and the performance of the processing elements improve, the concept of "slow" is rapidly changing allowing one to control system with quite fast dynamics as the thermal behavior of a microprocessor. Another drawback is represented by the accuracy of the model used for predictions. It is far from obvious to find a model accurately describing the behavior of the system, while keeping an acceptable order dimension that guarantees the solvability of the problem in the sampling interval. Model accuracy and computational speed are not the only limitations of MPC; it is worth to note that for safety critical applications on-line approaches are difficult to certify since it is difficult to show the correctness of a mathematical programming solver. This represents one of the reasons for which sometimes it is convenient to use an explicit MPC solution, that is a MPC solution in which the control decisions are pre-computed off-line for each possible state value. Finally, it is important to note that during the design phase it is necessary to consider the problem of feasibility and stability of the controller. Indeed, even if the system is stable, the predictive controller, using the receding horizon strategy, realizes a feedback control policy that may destabilize the closed-loop system. Moreover, the controller, optimizing the system performance over a finite prediction horizon, could take the system to a state in which the constraints cannot be met.

3.2 MPC structure

The typical goal for MPC controllers is to optimize the performance according to a cost function meeting some constraints (see Fig. 3.2a). Thus, the main elements constituting a MPC can be identified in:

- a model used for forecasting the future outputs of the system;
- a constrained optimization problem to be solved for obtaining a control decision sequence over a finite prediction horizon.

Different choices of these elements determine a different MPC algorithm.

3.2.1 Prediction models

The basic concept of MPC relies in using a dynamic model of the system to forecast its future behaviors. For this reason the models, commonly called *prediction models* in order to highlight their function inside the control algorithm, are instrumental for ensuring the effectiveness and the efficiency of a controller. The main properties a model must meet are the accuracy and low dimensions. The former guarantees accurate predictions, necessary for taking a correct control decision, whereas the latter ensures efficiency reducing the computational complexity.

An interesting characteristic of MPCs consists in their flexibility in accepting all possible model forms.

State-space model are commonly used in literature due to the simplicity of dealing with multivariable systems. A general state-space model can be described by the following equations,

$$\begin{aligned} \dot{x}(t) &= f(x(t),u(t)) \qquad x(0) = x_0 \\ y(t) &= h(x(t),u(t)) \end{aligned} \tag{3.1}$$

where $x \in \mathbb{R}^n$ is the state vector, $u \in \mathbb{R}^m$ is the input vector, $y \in \mathbb{R}^p$ is the output vector, and $t \in \mathbb{R}$ is the time.

However, it is worth to note that usually these models are specified in discrete-time. The reason is that in common applications, controllers are implemented on digital computers. Moreover, discrete-time models enable the use of powerful mathematical programming softwares for solving the optimal control problems (4). Thus, the generic nonlinear model can be stated as,

$$\begin{aligned} x(t+1) &= f(x(t),u(t)) \qquad x(0) = x_0 \\ y(t) &= h(x(t),u(t)) \end{aligned} \tag{3.2}$$

The linearity is often used as an approximation of the real system which usually behaves in a nonlinear fashion. The motivations for such an approximation is the ease of solution and analysis of linear models. The same reasons are behind the use of a linear time-invariant model as the one shown below,

$$\begin{aligned} &x(t+1) = A \cdot x(t) + B \cdot u(t)) \qquad x(0) = x_0 \\ &y(t) = C \cdot x(t) \end{aligned} \tag{3.3}$$

Notice that due to the delay always present between the measurements of $y(t)$ and the application of the $u(t)$, the feed-through matrix from u to y is usually not considered. The prediction output can be obtained as,

$$\hat{y}(t+k|t) = C \cdot x(t+k|t) = C \cdot \left[A^k \cdot x(t) + \sum_{i=0}^{k} A^{i-1} \cdot B \cdot u(t+k-i|t) \right] \tag{3.4}$$

where the notation $x(t+k|t)$ means the state x at time $t+k$ estimated at time t.

The input/output model representation is more convenient if we have few information of the internal model structure. It bases on the transfer function concept and the prediction model can be stated as,

$$\hat{y}(t+k|t) = \frac{b_1 \cdot z^{-1} + b_2 \cdot z^{-2} + \cdots + b_{nb} \cdot z^{-nb}}{1 + a_1 \cdot z^{-1} + a_2 \cdot z^{-2} + \cdots + a_{na} \cdot z^{-na}} \cdot u(t+k|t) \tag{3.5}$$

Other types of models, typically used in commercial softwares, are the *input response* and *step response* models. The idea behind these formulations is recording the open-loop response of each output variable (until they reach a steady-state value after $trun$ samples) when a impulse or step input is applied to each inputs variable. The response of any other input signal can be obtained by the knowledge of the previously found responses (because of the linearity and the superposition principle). The predicted output for the impulse model is given by the convolution sum,

$$\hat{y}(t+k|t) = \sum_{i=1}^{trun} h(i) \cdot u(t+k-i|t) \tag{3.6}$$

where we assumed a SISO model for the sake of simplicity and h represents the impulse response sequence truncated at the $trun$-th value. A similar model can be obtained for the step response model. Notice that these methods can be used only with asymptotically stable plants and the number of parameters required to have good approximation of the system could be large.

The flexibility of MPC allows one to use also more complex models as for example nonlinear models, partial derivative models, neural networks, ... However, it is worth to note that

the more complex are the models and the more time is required from the solver to compute the control decisions. Thus, it is convenient to avoid, if possible, needlessly complex models according to the principle of parsimony.

3.2.2 Constrained optimization problem

The MPC algorithms differs also for the optimization problems they have to solve. An optimization problem can be seen as composed by two main elements: the *cost function* (or *objective function*) and the *constraints*.

A general form of the **cost function** can be expressed as,

$$J_t(X_{t\to t+h_p|t}, U_{t\to t+h_p|t}) = p(x(t+h_p|t)) + \sum_{k=0}^{h_p-1} q(x(t+k|t), u(t+k|t)) \tag{3.7}$$

where $p(\cdot)$ represents the *terminal cost*, $q(\cdot)$ is the *stage cost*, h_p is the *prediction horizon* (the cost is defined over a finite horizon), and $U_{t\to t+h_p|t} = [u(t|t), \ldots, u(t+h_p-1|t)]'$ and $X_{t\to t+h_p|t} = [x(t|t), \ldots, x(t+h_p|t)]'$ are the control decisions and the state sequences for the time interval $[t, t+h_p]$. We assume the stage cost as continuous and such that $q(0,0) = 0$. The goal expressed by this cost function could be the regulation to zero of the state x or the tracking of a specified reference output trajectory. In particular, if the future reference evolution is known a priori, then the controlled system can act before of experiencing tracking errors compensating actuation delays.

The **constraints** are the main feature that distinguish MPC from classical finite horizon optimization problems. First of all, the state sequence $X_{t\to t+h_p|t}$ can be obtained by applying the $U_{t\to t+h_p|t}$ to the prediction model. Thus, the dynamic behavior of the state is constrained to be related to the control inputs by the model equations. Moreover, it is worth to note that, in real systems, the manipulated variables ($u(t)$) are usually bounded by physical constraints. As examples a motor have a limited maximum torque, a pneumatic valve has a limited displacement, and a processor has a maximum computational speed (or frequency). Additionally, it is often necessary to impose constraints on states and outputs for reasons of safety, product quality, or efficiency. As an example it could be preferable to maintain the temperature of a process below a specific critical value. Dealing with state constraints, it is often imposed a terminal constraints, i.e. bound on the last element of the state sequence. These requirements translate

in constraints on both the state and control sequences $U_{t\to t+h_p|t}$ and $X_{t\to t+h_p|t}$, therefore,

$$\begin{array}{ll} x(t+k+1|t) = f(x(t+k|t),u(t+k|t)) & k=0,\dots,h_p-1 \\ x(t+k|t)\in\mathbb{X} & k=0,\dots,h_p-1 \\ u(t+k|t)\in\mathbb{U} & k=0,\dots,h_p-1 \\ x(t+h_p)\in\mathbb{X}_f & \end{array} \tag{3.8}$$

where $\mathbb{X}$ and $\mathbb{U}$ are subset of $\mathbb{R}^n$ and $\mathbb{R}^m$ respectively. Usually $\mathbb{U}$ is a convex and compact set, while $\mathbb{X}$ is convex. $\mathbb{X}_f \subseteq \mathbb{X}$ is the terminal constraint set. Notice that the constraints on outputs can be expressed as states constraints since outputs are usually function of the state.

It is also worth to remark that constraints can be classified as *hard* or *soft*. The former regards those bounds which can never be crossed, whereas the latter are bounds that can be occasionally crossed. Usually, input constraints are typically hard constraints because they are dictated by the physical limits of the actuators (they cannot be softened). The states constraints instead are typically soft constraints because they represent desirable behaviors of the system (nevertheless they could be hard as well). Notice that a hard state constraint can be softened by adding to the optimization problem the so called *slack variables*, i.e. variables that assume non-zero values if the constraints are violated. The cost function contains a term dependent on the slack variables: the greater is the penalty weight associated to a function and the smaller is the constraint violation.

The **optimal control problem** at a particular $x(t)$ consists in minimizing the previously mentioned cost function, and meeting the constraints. The final problem can be stated as,

$$\min_{U_{t\to t+h_p|t}} J_t(x(t),U_{t\to t+h_p|t}) \triangleq p(x(t+h_p|t)) + \sum_{k=0}^{h_p-1} q(x(t+k|t),u(t+k|t)) \tag{3.9a}$$

$s.t.$

$$x(t+k+1|t) = f(x(t+k|t),u(t+k|t)) \quad k=0,\dots,h_p-1 \tag{3.9b}$$

$$x(t+k|t)\in\mathbb{X},\ u(t+k|t)\in\mathbb{U} \quad k=0,\dots,h_p-1 \tag{3.9c}$$

$$x(t+h_p|t)\in\mathbb{X}_f \tag{3.9d}$$

$$x(t|t) = x(t) \tag{3.9e}$$

where it is worth to note that the initial time at which we optimize the problem is relevant only if the cost function and the constraints are time-varying, otherwise we can write the previous problem as,

$$\min_{U_0} J_0(x(0),U_0) \triangleq p(x(h_p)) + \sum_{k=0}^{h_p-1} q(x(k),u(k)) \tag{3.10a}$$

$s.t.$

$$x(k+1) = f(x(k),u(k)) \quad k=0,\dots,h_p-1 \tag{3.10b}$$

$$x(k) \in \mathbb{X},\ u(k) \in \mathbb{U} \quad k=0,\dots,h_p-1 \tag{3.10c}$$

$$x(h_p) \in \mathbb{X}_f \tag{3.10d}$$

$$x(0) = x(t) \tag{3.10e}$$

where $U_0 = [u(0),\dots,u(h_p-1)]'$ Notice also that the cost value can be expressed as a function of the initial state $x(t)$ only, not of the whole state sequence, because, as already mentioned, we constrained the state evolution to be solution of the equation (3.9b).

In both cases, the problem is a *parametric optimization problem* in which the decision variable is $U_{t\to t+h_p|t}$ (or U_0) and both the cost function and the constraints depend on the parameter $x(t)$. The optimal solution of the problem is denoted by $U^\circ_{t\to t+h_p|t} = [u^\circ(t|t),\dots,u^\circ(t+h_p-1|t)]'$ while the correspondent optimal value of the cost function is $J^0_t(x(t))$. According to the receding horizon strategy only the control input $u(t) = u^\circ(t|t) = u^\circ(x(t|t))$ is applied to the system. For the sake of notational simplicity we consider hereafter the time-invariant case. Thus, the optimal solution can be written as $U^\circ_0 = [u^\circ(0),\dots,u^\circ(h_p-1)]$. We define $\mathcal{U}_0(x)$ as the set of U_0 such that the constraints are met and $\mathcal{X}_{h_p}$ as the set of $x \in \mathbb{X}$ such that the set $\mathcal{U}_{h_p}(x)$ is not empty.

The properties of both the cost function and the constraints (and hence also the prediction model) determine the classification of the optimization problem. We refer to the general problem (3.9) as a *nonlinear program*. When the cost and the constraints of the continuous optimization problem (3.9) are affines, then the problem is called a *linear program* (LP). Differently, if the cost function is a convex quadratic function and the constraint functions are affine, then the problem is called *quadratic program* (QP). The previously mentioned families of problem can assume the suffix *mixed-integer* if the optimization variables belong to a set obtained by the Cartesian product of a binary set and a real Euclidian space.

3.2.3 Different MPC solutions

In the past years the MPC was relegated to slow dynamics applications because of its need to solve the optimization problem on-line. As already mentioned the advance in computer

technology has progressively shifted the MPC applications to systems with increasingly faster dynamics opening up to new and more complex MPC schemes. The research on MPCs simply follows step-by-step the MPC schemes realized for new applications.

The first, and the simplest, MPC algorithms were used in the process industry. They belonged to the family of the so called *linear MPC*. The term *linear MPC* refers to those MPC schemes in which linear models are used to predict the future behavior of the system, which usually are inherently nonlinear. With the aim of maximizing performance and reducing costs, new algorithms based on more accurate nonlinear prediction model were implemented, introducing the family of *nonlinear MPC*. However, beside the accuracy and performance advantages, we have to consider that the use of a nonlinear model implies a lower efficiency in solving the optimization problem compared to a linear one, and difficulties in the stability analysis (13).

With the aim of reducing the on-line computational complexity of MPC algorithms, *explicit MPC* schemes solve the optimization problem off-line for all possible values of a parameter (e.g. the state vector x) rather than on-line for the current parameter value (14). Another approach that has had a good diffusion in the last decade was the *hybrid MPC*, a MPC algorithm in which the model comprises both continuous and discrete signals in the same framework. This scheme is able to handle switching linear dynamics, on/off inputs, logic states, and logic constraints on input and state variables. Both the explicit and hybrid algorithms, as an example, has been successfully applied in automotive applications (see (15) and references therein).

In aerospace systems and UAVs, besides hybrid schemes, *linear time-varying MPC* algorithm has been profitably used. In this case the controller uses a linear time-varying model. The problem can be reformulated as a QP problem and solved each sampling time.

The computational complexity difficulties of applying MPC control schemes to large scale systems led to *decentralized and distributed MPC* algorithms. Indeed, it is widely known that complexity exponentially scales with the model dimension. Nevertheless, a centralized schemes, if applicable, ensures better performance and prevent communication difficulties.

Finally in these recent years *stochastic MPC* schemes have been used to handle uncertainties. In past literature uncertainties were addressed using *Robust MPC* schemes that were designed on the pessimistic worst-case scenario and assuming bounded uncertainties. Stochastic MPC solutions take into account uncertainties with stochastic prediction models, and cost and constraint functions based on expected values.

The research community is aiming to bridge the gap between theory and practice. Analyzing literature, we can say that linear, explicit and hybrid MPC theory is mature, whereas

distributed/decentralized and stochastic MPC theory still lack contributions.

In this thesis we dealt with a time-invariant **linear MPC** scheme, although the prediction model underlying our controller algorithm is nonlinear. The greater efficiency of linear problem solvers, compared to nonlinear one, explains the rationale behind this choice. Indeed, for applications with fast dynamics the computational burden introduced by nonlinear MPC is still a serious barrier for its implementation. In the case of the processors thermal management, although thermal processes usually present slow dynamics, the tiny dimensions of the package and the huge power consumption make a controller with high sampling time necessary (the choice of a discrete-time controller is forced by the system in which the controller is implemented that is digital). However, the sampling time must be carefully chosen. Indeed, the time spent for solving the constrained optimization problem must be a small percent (e.g. 0.5%) of the sampling interval in order to make the computing effort for regulating the temperature invisible to the device users. Thus, the choice of the correct sampling time involves the solution of a trade-off problem.

The great majority of linear MPC algorithms in literature, as the ones presented in this thesis, rely on the solution of a convex QP. The general formulation of a convex QP-based MPC assumes a linear plant model, a quadratic cost function and linear inequalities as constraints,

$$\min_{U_0} J_0(x(0),U_0) \triangleq x(h_p)' P x(h_p) + \sum_{k=0}^{h_p-1} x(k)' Q x(k) + u(k)' R u(k) \tag{3.11a}$$

$s.t.$

$$x(k+1) = Ax(k) + Bu(k) \quad k = 0,\dots,h_p-1 \tag{3.11b}$$

$$E x(k) + M u(k) \leq \psi_k \quad k = 0,\dots,h_p-1 \tag{3.11c}$$

$$x(0) = x(t) \tag{3.11d}$$

where P and Q are symmetric and positive semi-definite (i.e. $P = P' \succeq 0$, $Q = Q' \succeq 0$) and R is symmetric and positive definite (i.e. $R = R' \succ 0$). In this case the Karush-Kuhn-Tucker (KKT) conditions are sufficient conditions for optimality, and the solution U_0 can be shown to be unique. Notice that all the constraints can be expressed with the equation (3.11c).

In the rest of this thesis we also treated other MPC schemes with the main purpose of reducing the computational complexity of the basic algorithm. Our main contribution is the

development of a **distributed MPC** solution which will be presented in the next chapter. Finally we tested the performance and characterized the complexity of our algorithm using an **explicit distributed MPC solution**. In the next Sections we briefly introduced the main issues of MPC schemes and the theory behind the distributed and explicit solutions.

3.3 Explicit MPC

In the previous Sections we presented the computational effort for solving on-line the constrained optimization problem as the biggest drawback of MPC schemes. Indeed, if we consider the linear quadratic MPC problem (3.11), the usual way of implementing the solution consists of translating the problem into a QP general form, as the one shown below, and then solving it on-line at every sampling time (see Section A.1 in Appendix A).

$$\frac{1}{2}x(0)'Y_{QP}x(0)+\min_{U_0}\frac{1}{2}U_0'Q_{QP}U_0+x(0)'F_{QP}U_0 \tag{3.12a}$$

$$s.t.$$

$$M_{QP}U_0 \leq W_{QP}+E_{QP}x(0) \tag{3.12b}$$

$$\tag{3.12c}$$

However, the computing time necessary for solving the QP problem could prevent the on-line use of the MPC solution in system with fast dynamics. Technology advances considerably reduced this issue, but other practical problems affect this solution: the hardware cost, and the complexity and the determinism of the software could move the users toward other solutions.

In this case it could be convenient to find a pre-computed control feedback function that relates the optimal solution to the current state of the system, preventing the solution of the QP problem on-line. The explicit solution shifts the computational burden off-line reducing considerably the complexity. Exploiting the multi-parametric programming approach (see Section A.2 in Appendix A), it is possible to find the optimal control input U_0° as an explicit function of the measured state parameter $x(0)$, that is $U_0^\circ(x(0)) = f(x(0))$ for all $x(0)$ in the set of feasible states. In (14) it has been proved that this function is piecewise affine respect to the state variables. The domain of the function, the feasible state set, is partitioned in convex polytopic regions, called *critical regions*, and a linear state feedback control law is associated to each region in order to yield the optimal control value. The union of all these control laws is the piece-wise control function.

Thus, the on-line computation reduces to detect the region and evaluate the control input using the related affine function. Assume $\{H^i \cdot x(0) \leq k^i\},\ i = 1,\ldots,N_r$ is the polyhedral set which defines the N_r partitions of the state space, and $F^i \cdot x(0) + G^i,\ i = 1,\ldots,N_r$ is the set of feedback control laws. Then, the algorithm executed on-line by the explicit solution can be summarized as:

1. measure the current state x(0);
2. detect the *i*-th polytope containing the state checking which condition $H^i \cdot x(0) \leq k^i$ is satisfied;
3. apply the correspondent control low $u(0) = F^i \cdot x(0) + G^i$

The on-line computational effort is strongly reduced. Additionally, compared to the implicit solution – the on-line MPC – we notice two other advantages. The first regards the hardware costs; the control algorithm need simple and cheap hardware components to compute the control law, therefore the approach is preferable for mass productions. Secondly, the low complexity of the software reduces the difficulties in estimating the worst-case CPU time necessary for solving the problem favoring safety certifications and the use in hard real-time scenarios.

However, explicit MPC also entails some disadvantages. First, the complexity introduced to solve the multi-parametric program off-line, that requires an increasing computational effort as the size of the problem increase, and second, the storage capacity of the memory. Indeed, as the problem size increases, the number of regions and the data that must be saved into the memory increase. These data regard the gain and offset arrays that define the regions (H^i and k^i) and the gain and offset arrays of the control law (F^i and G^i) associated to each region. We can say that the complexity is not disappeared but it has moved to memory usage. Thus, the decision between implicit or explicit MPC must be related to a trade-off between CPU and memory usage. Usually the explicit MPC solutions are limited to application with fast dynamics, but small dimensions (6-8 free moves and 8-12 states+references).

3.4 Distributed/Decentralized MPC for large scale systems

The size and complexity of a system are central problems for the design and development of MPC schemes (and in general for all typical controllers). According to (16), a system is considered as *large-scale* if it possess at least one of these properties:

- (decomposition) it can be partitioned into many small-scale and interacting subsystems;
- (complexity) its complexity prevents the use of conventional techniques of modeling, analysis, control, design, and computation do not give reasonable solutions with reasonable effort;
- (centrality) components and information cannot be grouped in one geographical location.

Typical examples of large scale systems are power networks, urban traffic networks, digital communications networks, flexible manufacturing networks, ecological systems, economic systems. We can also include to these examples multiprocessor systems, because of the elevated number of interacting processing units (the cores), although packaged in a small space.

As an example, a distributed linear time-invariant system with each subsystem controllable and coupled with the others can be modeled as,

$$\begin{bmatrix} x_1(t+1) \\ \vdots \\ x_{n_s}(t+1) \end{bmatrix} = \underbrace{\begin{bmatrix} A_{11} & \cdots & A_{1n_s} \\ \vdots & \ddots & \vdots \\ A_{n_s1} & \cdots & A_{n_sn_s} \end{bmatrix}}_{A} \cdot \begin{bmatrix} x_1(t) \\ \vdots \\ x_{n_s}(t) \end{bmatrix} + \underbrace{\begin{bmatrix} B_{11} & \cdots & B_{1n_s} \\ \vdots & \ddots & \vdots \\ B_{n_s1} & \cdots & B_{n_sn_s} \end{bmatrix}}_{B} \cdot \begin{bmatrix} u_1(t) \\ \vdots \\ u_{n_s}(t) \end{bmatrix} \tag{3.13}$$

where x and u represent respectively the state and the input vectors of the system, n_s is the number of subsystems, and $x_i \in \mathbb{R}^{n_i}$ and $u_i \in \mathbb{R}^{m_i}$ are the state and input vectors of the i-th subsystem. Moreover, $A \in \mathbb{R}^{n \times n}$ and $B \in \mathbb{R}^{n \times m}$ are the state and input matrices respectively, and $A_{ij} \in \mathbb{R}^{n_i \times n_j}$ and $B_{ij} \in \mathbb{R}^{n_i \times m_j}$ represent the contributions of the j-th subsystem state/input vector to the i-th subsystem state vector.

In large-scale systems a *centralized* control scheme – a scheme where all the information processed and all the control commands are dispatched by one central agency – is generally impossible or uneconomical. This is due to the lack of scalability of computational complexity, the impossibility of obtaining a centralized model, the difficulty of maintenance, and the impracticability of conveying all the communication signals to a single location. The same issues affect all controller families, hence also MPC schemes. In both the implicit and explicit MPC solutions the complexity increases as the dimension of the systems to be controlled increases. In the former case computational complexity increases, whereas in the latter is the memory usage that makes the controller impracticable.

A natural solution to the above mentioned issues is the development of *decentralized* or *distributed* control schemes in which each subsystem is controlled with a specified degree of autonomy respect to the other subsystems. Each subsystem is computationally tractable and

the local control inputs are computed using local measurements and reduced-order models of the local dynamics (17) (18). This control configuration enables:

- a computing effort reduction (the subsystems to be controlled are simpler);
- a communication load reduction (less data to be transmitted and for shorter distances);
- maintenance and reliability improvement (in case of damage the other subsystems continue working properly, data are not transmitted for long distances);
- a flexibility improvement (it is easier to update or modify parts of the system);
- a simplification of subsystems synchronization working at different time scale;
- a cost reduction (less communication links and less powerful hardware).

The same considerations hold for MPC controllers. Previously mentioned advantages open up to decentralized and distributed MPC schemes whose popularity is continuously increasing. In this scenario the original large-scale optimization problem is replaced by a set of small and tractable local optimization problems that work independently or cooperating one with each other.

The decentralized/distributed MPC schemes have not a fixed structure or algorithm. During their design, developers can choose some properties (or ingredients) to implement the control policy according to requirements.

A first choice is the degree of interaction between the local regulators. According to (19), a regulator can exchange information with another regulator or it can pursue for its optimal solution independently. This represents the difference between a *decentralized* and a *distributed* MPC scheme (see Fig. 3.3).

Decentralized MPC is composed by local agents that take control decisions independently one from another. Each local controller supervises a partition of the system whose inputs and outputs does not overlap inputs and outputs of other subsystems. Therefore they manage a limited amount of information (i.e. inputs, states, outputs). If some overlapping information exists these are neglected. Moreover, the time necessary to compute the control solution is not affected by communication overhead as delays or packets losses. Despite the diffusion of these schemes, there exist very few algorithms with guaranteed properties.

Contrary to the decentralized MPC schemes, distributed ones allow the transmission of information between the local agents, expanding the knowledge of local controllers respect to

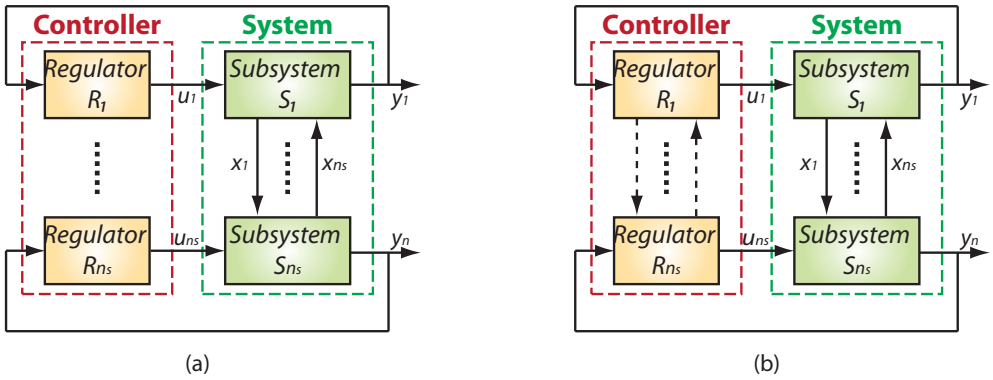


Figure 3.3: Decentralized (a) and Distributed (b) control approaches (19)

what is happening around them. This means an improvement of performance, at the expenses of a greater complexity due to communication and synchronization issues. Additionally, the complexity of the prediction model increases if a local regulator receives the predicted future control actions as input information (u_i in Fig. 3.2b). Indeed, in this case the local regulators should know the model of all the subsystems.

Another design choice for the development of distributed MPC schemes regards the topology of the communication network. We define as *fully connected algorithm* a MPC scheme where all the regulators transmit information to all the others. A *partially connected algorithm* instead is a MPC scheme where all the regulators transmit information to a subset of the others. This latter may be particularly convenient for large-scale system where some interaction between subsystems produce negligible performance deterioration.

Designers can also manage the rate of information exchange between controllers. The information can be transmitted only at the beginning or repeatedly within the same sampling interval. In the first case we refer to the MPC algorithm as *non-iterative*, whereas in the second as *iterative*.

Finally, it is possible to decide if using the information received from other local controllers for pursuing a global or a local goal. We define as *noncooperative* an algorithm where each local regulator minimizes a local performance index, and as *cooperative* an algorithm where each local regulator minimizes a global cost function.

As an example, consider the optimization problem (3.11) where we assume the separability of the cost function, i.e. the weights are diagonal matrices,

$$Q = \begin{bmatrix} p_1 Q_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & p_{n_s} Q_{n_s} \end{bmatrix} \quad R = \begin{bmatrix} p_1 R_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & p_{n_s} R_{n_s} \end{bmatrix} \quad P = \begin{bmatrix} p_1 P_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & p_{n_s} P_{n_s} \end{bmatrix} \tag{3.14}$$

such that,

$$\begin{aligned} J_0(x(0), U_0) &= p_1 \cdot J_{0,1}(x_1(0), U_{0,1}, \ldots, U_{0,n_s}) + \cdots + p_{n_s} \cdot J_{0,n_s}(x_{n_s}(0), U_{0,1}, \ldots, U_{0,n_s}) = \\ &= \sum_{j=0}^{n_s} p_j \cdot J_{0,j}(x_j(0), U_0) \end{aligned} \tag{3.15}$$

where

$$\begin{aligned} &J_{0,1}(x_1(0), U_{0,1}, \ldots, U_{0,n_s}) = x_1(h_p)' P_1 x_1(h_p) + \textstyle\sum_{k=0}^{h_p-1} x_1(k)' Q_1 x_1(k) + u_1(k)' R_1 u_1(k) \\ &\vdots \\ &J_{0,n_s}(x_{n_s}(0), U_{0,1}, \ldots, U_{0,n_s}) = x_{n_s}(h_p)' P_{n_s} x_{n_s}(h_p) + \textstyle\sum_{k=0}^{h_p-1} x_{n_s}(k)' Q_{n_s} x_{n_s}(k) + u_{n_s}(k)' R_{n_s} u_{n_s}(k) \end{aligned} \tag{3.16}$$

and $U_{0,j} = [u_j(0), \ldots, u_j(h_p - 1)]$.

Moreover, it is assumed that the constraints are uncoupled, i.e. there is no interaction or coupling of the inputs in the constraint relation (3).

In this scenario we can define the **centralized problem** as,

$$\min_{U_0} \sum_{j=1}^{n_s} p_j J_{0,j}(x_j(0), U_0) \tag{3.17a}$$

$$s.t.$$

$$x(k+1) = Ax(k) + Bu(k) \quad k = 0, \ldots, h_p - 1 \tag{3.17b}$$

$$u(k) \in \mathbb{U} \quad x(k) \in \mathbb{X} \quad k = 0, \ldots, h_p - 1 \tag{3.17c}$$

where A and B are defined in (3.13)

The **decentralized problem** correspond to the other extreme in distributing the decision making in a large-scale system. Whereas centralized control knows everything about the system and optimizes respect to all the decision variables, the local control of decentralized schemes has no information about the other subsystems and it optimizes only the local cost function. The regulator R_j will have the form,

$$\min_{U_{0,j}} J_{0,j}(x_j(0), U_{0,j}) \tag{3.18a}$$

$$s.t.$$

$$x_j(k+1) = A_{jj}x_j(k) + B_{jj}u_j(k) \quad k = 0,\dots,h_p-1 \tag{3.18b}$$

$$u_j(k) \in \mathbb{U}_j \quad x_j(k) \in \mathbb{X}_j \quad k = 0,\dots,h_p-1 \tag{3.18c}$$

However, a completely decentralized scheme is not able to achieve the overall objective of the system. Distributed solutions offer a middle ground between decentralized and centralized control, allowing one to obtain better performance than the former and lower complexity than the latter. The **noncooperative problem** exploits the information of other subsystems, but it pursues its own objective. It can be defined as,

$$\min_{U_{0,j}} J_{0,j}(x_j(0), U_0) \tag{3.19a}$$

$$s.t.$$

$$x_j(k+1) = A_{jj}x_j(k) + B_{jj}u_j(k) + A_{ij}\hat{x}_i(k) + B_{ij}\hat{u}_i(k) \qquad \begin{matrix} k = 0,\dots,h_p-1 \\ i = 1,\dots,n_s,\ i \neq j \end{matrix} \tag{3.19b}$$

$$u_j(k) \in \mathbb{U}_j \quad x_j(k) \in \mathbb{X}_j \quad k = 0,\dots,h_p-1 \tag{3.19c}$$

where the ˆ symbol means that the predictions of that variable are available.

Finally, in the **cooperative problem** the control agents share a common objective, obtaining performance improvements respect to the noncooperative one. The problem can be defined as,

$$\min_{U_{0,j}} \sum_{j=1}^{n_s} p_j J_{0,j}(x_j(0), \hat{U}_{0,1},\dots,U_{0,j},\dots,\hat{U}_{0,n_s}) \tag{3.20a}$$

$$s.t.$$

$$x(k+1) = Ax(k) + B[\hat{u}_1(k),\dots,u_j(k),\dots,\hat{u}_{n_s}(k)]' \quad k = 0,\dots,h_p-1 \tag{3.20b}$$

$$u_j(k) \in \mathbb{U}_j \quad x(k) \in \mathbb{X} \quad k = 0,\dots,h_p-1 \tag{3.20c}$$

The thermal management solution we developed in this thesis belongs to the family of distributed schemes. We assigned to each core (or to subsets of cores) a local MPC. Each controller transmits information to a subset of controllers, therefore the topology of the communication network is partially connected. We assumed no delays in information transmissions

since the distance of the core are very close and we use a non-iterative algorithm. The information transmitted are not control decisions, but output information that prevent the controller to know the whole system model. We finally use a noncooperative policy so that each controller optimizes only the local cost function. All the properties we chose for our control solution aim to reduce computational complexity of the algorithm.

3.5 Feasibility, Stability, and Computational Complexity

The design of a MPC algorithm hides some dangerous treats which may compromise the correct functioning of the controller. These are mainly due to the finiteness of the prediction horizon and the presence of constraints. In a MPC the control decisions are optimized over the prediction interval, without considering that the reached state could be impossible to stabilize (optimality does not imply stability) or even avoid the feasible solution of the problem. The stability and feasibility are not ensured by the MPC algorithm. Thus, the MPC developers should, in principle, analyze the impacts of the different tuning choices to prove the validity of these properties. Since this operation is usually prohibitive, feasibility and stability are usually obtained by reformulating the problem.

Another issue regards the complexity of solving the optimization problem that is usually correlated to the problem dimension and the number of variables.

These issues are briefly accounted in the following Subsections.

3.5.1 MPC Feasibility

First of all we need to define what is the meaning of feasibility. Consider a generic optimization problem,

$$\min \ f_0(z) \tag{3.21a}$$
$$s.t.$$
$$f_i(z) \leq 0, \quad i = 1, \ldots, n_{ineq} \tag{3.21b}$$
$$h_i(z) = 0, \quad i = 1, \ldots, n_{eq} \tag{3.21c}$$
$$z \in Z \tag{3.21d}$$

the optimization variable, z, is *feasible* if:

- it belongs to the set of value for which the objective and all constraint functions are defined (3.21d);
- it satisfies the constraints (3.21b) and (3.21c).

An optimization problem, instead, is a *feasible problem* if there exists at least one feasible z. On the other hand, a problem is said to be *infeasible* if such a value does not exist. The set of all the feasible z is called the *feasible set* (20).

The MPC strategy solves an optimization problem at each time step starting from the current state $x(t)$ (or $x(0)$ if we assume a time-invariant systems) over a finite horizon. It optimizes the control sequence $U_{t\to t+h_p|t}$ (or U_0 in the time-invariant case) that represents the optimization variable. For the sake of notation simplicity we will consider the time-invariant case hereafter. The problem is feasible if a control sequence U_0 exists meeting the constraints. However, it is worth to note that in the MPC problem formulation it is present the initial state $x(0)$, therefore the feasibility of the problem also depends on this parameter. For this reason $x(0)$ is said to be feasible if the problem is feasible starting from $x(0)$. $\mathcal{X}_0$ represents the set of initial states $x(0)$ for which the optimal control problem is feasible, i.e.

$$\begin{aligned}\mathcal{X}_0 = \big\{ & x(0) \in \mathcal{X} : \exists U_0 \; such that \; x(t) \in \mathbb{X},\, u(t) \in \mathbb{U},\, t = 0,\dots,h_p-1,\, x_{h_p} \in \mathcal{X}_f \\ & where \; x(t+1) = Ax(t) + Bu(t),\, t = 0,\dots,h_p-1 \big\}\end{aligned} \tag{3.22}$$

where $\mathcal{X}$ is the set of all possible x, $\mathbb{X}$ and $\mathbb{U}$ are the state and input constraint sets respectively, and $\mathcal{X}_f$ is the terminal set that we want the states to reach at the end of the horizon, i.e. after h_p prediction samples.

The feasibility ensures the existence of a solution to the problem, but this is not enough for guaranteeing the feasibility of the MPC algorithm. Indeed, the feasibility of the optimization problem can be lost during the functioning. The problem is due to the "short sight" of the MPC which may steer the state to a value for which no feasible control decisions exist in the next sampling interval. Instead, it is desirable a MPC possessing the property called *recursive* (or *persistent*) feasibility, which can be stated as,

Definition 3.5.1. *If the controller is feasible at any time, for all input control sequences and for all initially feasible state $x(0)$, then the MPC controller is recursively (or persistently) feasible.*

Proving this property is computationally difficult, since it requires to search at any time the set of states that remain feasible at the next sampling interval. Typically the feasibility at time $t = 0$ is assumed and the structure of the problem (cost function, constraints, and terminal

constraints) is modified so that feasibility is preserved at the following time steps. As an example, it is possible to soften state constraints using slack variables, or it is possible to insert constraints on the terminal set $\mathcal{X}_f$ or varying the length of the prediction horizon.

Recursive feasibility of finite-horizon MPC problems can be explicitly enforced by constraining the state at the final prediction step to a controlled invariant set. First, using the invariant set theory, it is possible to prove that a necessary and sufficient condition for guaranteeing recursive feasibility can be given by imposing that the initial state set $\mathcal{X}_0$ is equal to the set of all initial states generating feasible closed-loop trajectories, $\mathcal{O}_\infty$.

Lemma 3.5.1. *Consider the problem (3.11) and denote with f_t the receding horizon control law that associates the optimal input u_0° to the current state $x(0)$, $f_t(x(0)) = u_0^\circ(x(0))$. Let $\mathcal{O}_\infty$ be the maximal positive invariant set for the closed-loop system $x(1) = Ax(0) + B\,f_t(x(0))$. The RHC problem is persistently feasible if and only if $\mathcal{X}_0 = \mathcal{O}_\infty$.*

However, $\mathcal{O}_\infty$ depends on the matrices, Q, P, and R, therefore for some of these tuning variable the recursive feasibility may not be proved. According to (4), it is possible to make $\mathcal{O}_\infty$ independent from Q, P, and R by taking the terminal set $\mathcal{X}_f$ as a control invariant set of the system model with constraints.

Theorem 3.5.2. *Consider the problem (3.11), if $\mathcal{X}_f$ is a control invariant set for the constrained system (3.11b) then the MPC is persistently feasible.*

where we define as control invariant set for a system subject to constraints as the set of states x such that there exists a feasible control u for with the future state $Ax + Bu$ belong to the same set (refer to (4) for the proof).

Another approach for detecting if a MPC is recursively feasible, has been proposed in (21). The author considers a linear time-invariant problem as the one in (3.11), without the need of the terminal constraint. The idea is to prove the existence of problematic states – states for which the optimization problem has no solution – by exploiting a bilevel optimization problem.

Consider the QP problem (3.12) obtained from (3.11). The problem is infeasible if a feasible state is steered to an infeasible state by applying the optimal control decision, that is the following inequality is not satisfied,

$$M_{QP}\,U_{t+1} \leq W_{QP} + E_{QP}\,(Ax(t) + Bu(t)^\circ) \tag{3.23}$$

Then this condition can be rewritten as a set of three condition,

$$y \geq 0 \tag{3.24a}$$

$$y' M_{QP} = 0 \tag{3.24b}$$

$$y' \left[W_{QP} + E_{QP} \left(Ax(t) + Bu(t)^{\circ} \right) \right] < 0 \tag{3.24c}$$

by using the Farkas' Lemma, according to with,

Lemma 3.5.3. *Let $M \in \mathbb{R}^{p \times q}$ and $w \in \mathbb{R}^{p}$. Then either there is an $x \in \mathbb{R}^{q}$ such that $Mx \leq b$ or there is a $y \in \mathbb{R}^{p}$ such that $y \geq 0$, $y'M = 0$, and $y'b < 0$.*

Using conditions (3.24) we can implement the bilevel problem

$$\min_{y, x(t), U_t} \; y' \left(W_{QP} + E_{QP} \left(Ax(t) + Bu(t)^{\circ} \right) \right) \tag{3.25a}$$

$$s.t.$$

$$y \geq 0 \,, \;\; y' F = 0 \tag{3.25b}$$

$$U_t^{\circ} = arg \; problem \; (3.12) \tag{3.25c}$$

If the optimal value of the cost function is negative the problem is infeasible for an admissible state, according to Farkas' Lemma. The problem can be simplified by substituting to the constraint (3.25c) representing the inner optimization problem the Karush-Kuhn-Tucker conditions (since the problem is convex).

3.5.2 MPC Stability

The second issue for MPC is stability. In the first industrial MPC applications the possibility of automatically ensure stability was unavailable, requiring a manual tuning of the algorithm. The research community devoted considerable attention to this topic producing numerous solutions.

The problem of stability consists in designing a MPC algorithm guaranteeing that the origin of the closed-loop system is an asymptotically stable equilibrium point. As for the case of feasibility, the main approach for ensuring stability is modifying the structure of the MPC. The main modifications regard the terminal cost (the terms $p(x(t+h_p|t))$ in (3.9) and $x(h_p)'Px(h_p)$ in (3.11)), the terminal constraint set (defined as $\mathcal{X}_f$) and the terminal controller ($k_f(\cdot)$ for stabilizing the state inside $\mathcal{X}_f$). These are the three main "ingredients" for building a MPC satisfying stability (and often also feasibility).

There exist also different techniques for proving closed-loop stability. A strand of the literature shows that closed-loop stability may often be achieved using a sufficiently long prediction horizon (22) (23). Some methods require that the state $x(t)$ is shrinking in some norm, as the *Contraction Constraint* approach (24) (25) which requires that $x(t)$ is decreasing in some norm ($\|x(t+1|t)\| \leq \alpha \|x(t)\|$ and $\alpha < 1$). However, the most used approach consists in choosing the previously mentioned "ingredients" such that the cost function is a Lyapunov function. The reason is that for nonlinear controllers the natural tool for establishing stability is Lyapunov theory, and MPCs are for their nature nonlinear because of the presence of constraints (note that the explicit solution of the QP-MPC is piecewise linear even if the model is linear). In the excellent survey paper (26) the authors analyzed the MPC solutions present in literature for ensuring stability and recognized the three previously mentioned ingredients as the tuning knobs usable by the designers. Furthermore the authors distilled four conditions on the ingredients, sufficient for guaranteeing closed-loop stability. Before listing these conditions, we recall the general MPC optimization problem,

$$\min_{U_{t\to t+h_p|t}} \; J_t(x(t), U_{t\to t+h_p|t}) \triangleq p(x(t+h_p|t)) + \sum_{k=0}^{h_p-1} q(x(t+k|t), u(t+k|t)) \tag{3.26a}$$

$$s.t.$$

$$x(t+k+1|t) = f(x(t+k|t), u(t+k|t)) \quad k = 0,\dots,h_p-1 \tag{3.26b}$$

$$x(t+k|t) \in \mathbb{X}, \;\; u(t+k|t) \in \mathbb{U} \quad k = 0,\dots,h_p-1 \tag{3.26c}$$

$$x(t+h_p|t) \in \mathbb{X}_f \tag{3.26d}$$

$$x(t|t) = x(t) \tag{3.26e}$$

where $p(x(t+h_p|t))$ is the terminal cost, $\mathcal{X}_f$ is the terminal set and the local controller $k_f(\cdot)$ is merely implicit, but is required to prove stability.

The conditions are stated below:

A1 : $\mathcal{X}_f \subset \mathbb{X}$, $\mathcal{X}_f$ closed, $0 \in \mathcal{X}_f$ (state constraint satisfied in $\mathcal{X}_f$);

A2 : $k_f(x) \in \mathbb{U}$, $\forall x \in \mathcal{X}_f$ (control constraint satisfied in $\mathcal{X}_f$);

A3 : $f(x, k_f(x)) \in \mathcal{X}_f$, $\forall x \in \mathcal{X}_f$ ($\mathcal{X}_f$ is positively invariant under $k_f(\cdot)$);

A4 : $[p(f(x, k_f(x))) - p(x) + q(x, k_f(x))]$, $\forall x \in \mathcal{X}_f$ (p is a local Lyapunov function)

If these conditions hold, then the value function is a Lyapunov function, a sufficient condition for stability. More in detail, the last condition ensure the descending property of the Lyapunov function. This is easy to prove.

Consider as a Lyapunov function the optimal solution of the problem for the initial state $x(t)$,

$$J_t^\circ = J_t(x(t), U_{t\to t+h_p|t}^\circ)$$

where $U_{t\to t+h_p|t}^\circ = [u^\circ(t|t), \ldots, u^\circ(t+h_p-1|t)]'$ and $X_{t\to t+h_p|t}^\circ = [x^\circ(t|t), \ldots, x^\circ(t+h_p|t)]'$ is the resultant state sequence.

The successor state $x(t+1)$ is computed as $f(x(t), k_f(x(t)))$ and analogously to the previous definitions,

$$J_{t+1}^\circ = J_{t+1}(x(t+1), U_{t+1\to t+h_p+1|t+1}^\circ)$$

where $U_{t+1\to t+h_p+1|t+1}^\circ = [u^\circ(t+1|t+1), \ldots, u^\circ(t+h_p+1|t+1)]'$ and $X_{t+1\to t+h_p+1|t+1}^\circ = [x^\circ(t+1|t+1), \ldots, x^\circ(t+h_p+1|t+1)]'$ is the resultant state sequence.

Since it is difficult to directly compare J_t° and J_{t+1}° we can find an upper bound of J_{t+1}° using a feasible – but not optimal – input sequence for the time $t+1$ by shifting the optimal input sequence at time t. The new sequence is given by,

$$\widetilde{U}_{t+1\to t+h_p+1|t} = [u^\circ(t+1|t), \ldots, u^\circ(t+h_p|t),\ k_f(x(t+h_p))]'$$

The value function for this input sequence can be defined as,

$$\begin{aligned} J_{t+1}(x(t+1), \widetilde{U}_{t+1\to t+h_p+1|t}) = & J_t^\circ - q(x(t), k_f(x(t))) - p(x^\circ(t+h_p|t)) + \\ & + q(x^\circ(t+h_p|t), k_f(x^\circ(t+h_p|t))) + \\ & + p(f(x^\circ(t+h_p|t)), k_f(x^\circ(t+h_p|t)))) \end{aligned} \tag{3.27}$$

where $x(t)$ is the initial state. Since $J_{t+1}^\circ \le J_{t+1}(x(t+1), \widetilde{U}_{t+1\to t+h_p+1|t})$, it follows that,

$$J_{t+1}^\circ - J_t^\circ \le -q(x(t), k_f(x(t))) \tag{3.28}$$

that is true only if for all x there exists a input $u \in \mathbb{U}$ such that,

$$p(f(x, k_f(x))) - p(x) + q(x, k_f(x)) \le 0 \tag{3.29}$$

The four conditions previously stated can be satisfied by different choices of the "ingredients" that defines different MPC schemes. These are few example (30) (26):

- *Terminal state*: $x(t+h_p|t) = 0$.

- *Infinite prediction horizon*: $h_p = \infty$.
- *Terminal Weighting Matrix*: in the linear case, P is the solution of a Riccati inequality.
- *Invariant terminal set*: $x(t+h_p|t) \in \Omega$ and $u(t+k|t) = F_{LQ}x(t+k|t)$, $\forall k \geq h_m$, where F_{LQ} is the LQ feedback gain.

For what concerns the distributed MPC controllers the theory is still not mature. The actual trend is similar to the one of the MPC in the past: there exist many efficient algorithms, but strong theoretical results and a unifying picture are still partially lacking.

As an example, in (31) the authors showed that the cooperative MPC approach leads to closed-loop stability in the linear case.

In general, whereas cooperative schemes reach a Pareto equilibrium, noncooperative ones have been shown to reach a Nash equilibrium (the controllers optimize their objectives independently). We define the two types of equilibrium for completeness,

Nash equilibrium : point reached when the objective of each controller cannot be improved by varying any of its control actions;

Pareto equilibrium : point reached when a modification of the local objectives, necessarily worsens the global objective.

It is worth to note that the Nash equilibrium is not sufficient for stability, therefore, as for the non distributed case, some constraints are imposed on the local controllers.

3.5.3 MPC Complexity

The third issue we treated in this Section is the computational complexity which becomes prohibitive for large systems or systems with fast dynamics. The influencing factor for the computational complexity are the type of MPC scheme used and the dimension of the problem.

The first consideration derive from the fact that the solvers used for linear MPC are more efficient respect to the solvers used for finding a solution of a nonlinear MPC.

The complexity of a problem increases also with the dimension of the state variable (x) and of the controls (m), with the length of the prediction horizon (h_p), and with the number of constraints (n_c). Focusing on linear quadratic MPCs, as the one shown in (3.11), we have already noticed that it can be rewritten as a convex QP problem. Two popular methods for solving QP problems are the *active set* (27) and the *interior point* (28) methods. A naive

application of the active set method on a general QP problem can require $O(h_p^3(m+n+n_c)^3)$ operations. However, according to (29) with an appropriate reordering of the variables, it is possible to improve the number of operation obtaining $O(h_p(m+n+n_c)^3)$ or $O(h_p(m+n+n_c)^2)$ respectively with an interior point method and an active set method. In both these cases we can see that the computational complexity exponentially increase with the number of the states, of control and variables (for major details refer also to (20) and the references therein).

From the previously data it is clear that the number of inputs m multiplied with the prediction horizon h_p, usually called *degrees of freedom* of the problem, are the dominating factor for complexity in MPC. For this reason, as we have previously mentioned in Section 3.1, the move-blocking strategy allows to reduce the computational complexity (e.g. imposing the control value as constants after a specific interval).

Another useful tool for having some insights on the computational efforts of solving an MPC problem is the explicit MPC approach. In this thesis we exploited the explicit approach in two different ways. First, for reducing the on-line computational complexity, relegating the difficulties of solving the optimization problem off-line. The complexity of the algorithm depends on the number of regions N_r, that in turn depends on the dimension of the state (n), on the number of constraints (n_c), and on the number of controls (m) – in particular on the degrees of freedom ($s = m h_c$) (14). The number of regions is strongly related to the number of constraints, and the computational time necessary to find the control law increases with an exponential trend. For this reason this approach can be used only for small value of m, n_c, and h_p. To give a dimension of the term "small" if $n \geq 5$,$m \geq 3$, and $l \geq 12$ the number of regions cannot be efficiently managed. As a result solving a QP problem on-line is faster than detecting the region and applying the control law (32).

The second way we used the explicit approach has been as a metric for determine the complexity of an on-line algorithm. Indeed, as the complexity of the problem increases, so also the number of regions increase. We can say that the complexity of the MPC algorithm is manifested by the number of regions.

Fig. 3.4 show a comparison between the number of regions and the CPU time of a constrained double integrator system (33) when the degrees of freedom varies.

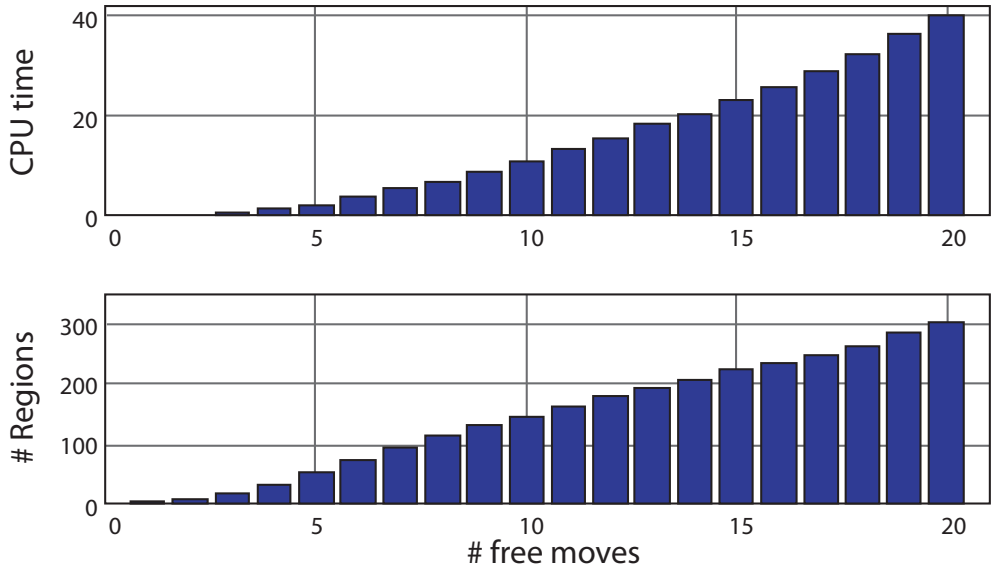


Figure 3.4: Number of regions and CPU time comparison varying the number of degrees of freedom

3.6 Notes

We conclude this chapter with some notes. The MPC solutions presented in this thesis for the thermal and energy management of processors have been developed with the intent of improving performance and reducing computational complexity.

These requirements motivates our choices of using distributed and explicit MPC solutions. The use of explicit solutions also allowed us to have a double check on the complexity data we obtained from experiments.

Another important observation is that, although we presented the common approaches used in literature for ensuring the feasibility and the stability, we obtained our proofs directly studying the physical properties of our system. More in detail, the study of the properties of the system allows us to prove the feasibility without increasing the complexity of the MPC scheme by adding useless constraints or terminal costs. Additionally, the validity of the proof regards all the thermal systems and all the controllers whose target is the temperature capping, therefore the proof is widely general. Moreover, the study of the physical properties of the systems allows us to find other useful properties which simplify the design of the controller.

It is finally worth to remark that in this thesis we omitted the stability proof simply because our application does not require it. Indeed, as it will be clear later on, we are not interested

in maintaining the temperature of the system arbitrarily close to a set point or an equilibrium point, but below the maximum temperature. Our priority is meeting the constraints. We will see that our problem only required the boundedness of the state that is guaranteed by the feasibility property. This argument will be deeply detailed in the last part of the next Chapter.

Bibliography

[1] E.F. Camacho, C. Bordons, *Model predictive control*, Springer, 1999. 50

[2] J.M. Maciejowski, *Predictive control: with constraints*, Pearson Education, 2002. 50

[3] J.B. Rawlings, D.Q. Mayne, *Model Predictive Control Theory and Design*, Nob Hill Publishing, 2009. 50, 67

[4] F. Borrelli, A. Bemporad, M. Morari, *Model Predictive Control for linear and hybrid systems*, in preparation, last update Nov, 2012. viii, 50, 51, 52, 55, 71

[5] W.H. Kwon, S.H. Han, *Receding Horizon Control: Model Predictive Control for State Models*, Springer, 2005. 51

[6] S.J. Qin, T.A. Badgwell, *An Overview Of Industrial Model Predictive Control Technology*, In Chemical Process Control, Vol. 93(316):232–256, 1997. 51

[7] A.I. Propoi, *Use of linear programming methods for synthesizing sampled-data automatic systems*, Automation and Remote Control, Vol. 24(7):837-844, 1963.

[8] E.B. Lee, L. Markus, *Foundations of optimal control theory*, New York: Wiley, 1967.

[9] J. Richalet, A. Rault, J.L. Testud, J. Papon, *Algorithmic control of industrial processes*, in Proc. of the Fourth IFAC symposium on identification and system parameter estimation, pp. 1119–1167, Tbilisi, 1976 53

[10] J. Richalet, A. Rault, J.L. Testud, J. Papon, *Model predictive heuristic control: Applications to industrial processes*, Automatica, Vol. 14:413–428, 1978. 53

[11] C.R. Cutler, B.C. Ramaker, *Dynamic matrix control - a computer control algorithm*, In Proc. American Contr. Conf., Vol. WP5-B, San Francisco, USA, 1980. 53

[12] C.R. Cutler, A. Morshedi, J. Haydel, *An industrial perspective on advanced control*, In AIChE annual meeting, Washington, DC, Oct. 1983. 53

[13] R. Findeisen, F. Allgöwer, *An Introduction to Nonlinear Model Predictive*, 21st Benelux Meeting on Systems and Control, VeidhovenJ, 2004, pp. 1–23. 60

[14] A. Bemporad, M. Morari, V. Dua, and E. Pistikopoulos, *The explicit linear quadratic regulator for constrained systems*, Automatica, Vol. 38(1):3-20, 2002. 60, 62, 76

[15] A. Bemporad, *Model Predictive Control Design: New Trends and Tools*, in Proc. of the 45th IEEE Conference on Decision & Control, Manchester Grand Hyatt Hotel San Diego, CA, USA, December 13-15, 2006. 60

[16] McGraw-Hill, *Dictionary of Scientific and Technical Terms*, 6th edition, The McGraw-Hill Companies, Inc. 63

[17] N.R. Sandell, P. Varaiya, M. Athans, M.G. Safonov, *Survey of decentralized control methods for large scale systems*, IEEE Trans. Automat. Contr., Vol. AC-23, pp. 108-128, Feb. 1978. 65

BIBLIOGRAPHY

[18] D.D. Siljak, *Decentralized control and computations: Status and prospects*, Annu. Rev. Contr., Vol. 20, pp. 131-141, 1996. 65

[19] R. Scattolini, *Architectures for distributed and hierarchical model predictive control : a review* Journal of Process Control, Vol. 19, pp. 723–731, 2009. viii, 65, 66

[20] S. Boyd, L. Vandenberghe, *Convex Optimization*, Cambridge University Press, 2004. 70, 76

[21] J. Löfberg, *Oops! I cannot do it again: Testing for recursive feasibility in MPC*, Automatica, Vol. 48(3):550–555, 2011. 71

[22] T.Parisini. R. Zoppoli, A receding hor1zon regulator for nonlinear systems and a neural approximation, Automatica, 31(10):1443–1451, 1995. 73

[23] P.O.M. Scokaert, J.B. Raw1ings, Constrained linear quadratic regulation, IEEE Trans. Auto. Cont., Vol. 43(8):1163–1169, Aug. 1998. 73

[24] E. Polak, T.H. Yang, Moving horizon control of linear systems with input saturation and plant uncertainty–part 1. robustness, Int. J. Control, Vol. 58(3):613–638, 1993. 73

[25] Z.Q. Zheng, Robust Control of Systems Subject to Constraints, Ph.D. dissertation. California Institute of Technology. Pasadena, CA, U.S.A, 1995. 73

[26] D.Q. Mayne, J.B. Rawlings, C.V. Rao, P.O. Scokaert, Constrained model predictive control: stability and optimality, Automatica, Vol. 36, pp. 789–814, 2000. 51, 73, 74

[27] B. Fletcher, *Practical Methods of optimization*, second edn, John Wiley and Sons, New York, 1987. 75

[28] S.J. Wright, *Primal-Dual Interior-Point Methods* , Society for Industrial and Applied Mathematics, Jan., 1987. 75

[29] S. J. Wright, *Applying new optimization algorithms to model predictive control*, Chemical Process Control-V., Vol. 93(316):147155, 1997. 76

[30] A. Bemporad, M. Morari, Robust model predictive control: A survey(1999), In A. Garulli, A. Tesi, A. Vicino (Eds.), *Robustness in identification and control*, Lecture Notes in Control and Information Sciences, Vol. 245, pp. 207226, Berlin: Springer. 74

[31] N. Venkat, J.B. Rawlings, S.J. Wright, *Distrtbutedmodel predictive control of large-scale systems*, In Assessment and Future Direcrtons of Nonlinear Model Predictive Control, pp. 591–605. Springer, 2007. 75

[32] Y. Wang, S. Boyd, *Fast Model Predictive Control Using Online Optimization*, IEEE Transaction on Control System Technology, Vol. 18(2):267–278, mar. 2010. 76

[33] A. Bemporad, Explicit Model Predictive Control, Slides of SIDRA doctorate school, 2012. 76

Chapter 4

MPC thermal controller for MPSoCs

In this chapter the basic distributed MPC solution is presented. First, the focus will be devoted to highlight the importance of the model for MPC accuracy. In this context some methods to obtain accurate and reduced order models of the system are illustrated. Then, the centralized and distributed MPC control schemes will be accurately described showing the strengths of the latter solution. Finally, the feasibility property will be proved for centralized and distributed controllers.

4.1 The prediction model

In the previous chapter we showed how MPC schemes strongly rely on the dynamic models used to forecast the future behaviors of the system. In order to build an efficient and effective control solution, the properties the model must satisfy are *simplicity* and *accuracy*. The former is necessary for reducing computational complexity and guaranteeing to find a control decision before the ending of the sampling interval. The latter, instead, affects the optimality of the control decision, which strongly depends on predictions.

When we build a model, the parameters that affect computational complexity and accuracy are essentially three: the model type, its order and the number of inputs.

Although there is nothing in the theory of MPC schemes against the use of nonlinear models, linear models (if correctly describe the phenomena) are usually preferable because they are easier to identify and enable the use of more reliable and efficient algorithms to solve the optimization problem, guaranteeing a global minimum solution (nonlinear MPC schemes are

generally non-convex)(1) (2). However, linear model are usually approximation of the natural phenomena which usually are nonlinear.

The model order represents the state dimension of the model. The greater the order, the greater the number of system modes that describe the behavior of the system, and hence its accuracy. However, as remarked in Chapter 3, the computational complexity of a MPC scheme is exponentially related to the state dimension and the number of constraints.

Finally, it is important to identify the inputs that have a strong impact on the observed behaviors, while discarding the ones that have a negligible contribution. Indeed, as the number of inputs increases, also the computational complexity increases, since the number of multiplications and additions increase. Moreover, also the design complexity is negatively affected by the increasing of information transmission.

The aim of this section is to understand the motivations that have led to the models we used in our control algorithms and to show the techniques adopted to identify their parameters. Thus, in the follows we will answer to these questions: What type of model? What model order? How many inputs?

Clearly, it is impossible to answer these questions without having any information on the system we want to model. The required model has to describe the thermal behavior of a generic multiprocessor chip (see Appendix B for major details on how a generic multiprocessor may be modeled). The system takes as inputs the core frequencies, the supply voltage and the workload of each core and it returns as output the temperatures of a set of points on the die, which correspond to the measurements from the sensors. In this thesis we reasonably assumed a sensor for each core. As it is possible to note the temperature and the power dissipation of caches are respectively uncontrollable and unmeasurable. Thus, the system is under-actuated, since to keep the temperature of the caches below the constraint value we can only manage the power dissipation of the cores. However, it is worth to remark, that this limitation is not so restrictive because the highest power density are consumed on cores where usually the most dangerous thermal challenges occur (7).

What type of model? As shown in Appendix B, the thermal behavior of a processor can be modeled by a nonlinear mathematical function that depends on five main parameters: the cores frequency, the cores workload (CPI – clocks per instruction), the supply voltage, the chip temperature and the ambient temperature. Such a function can be decomposed in two sub-functions that realize the following “causal chain”:

$$(freq, CPI, V_{dd}, T) \xrightarrow{\mathcal{P}(\cdot)} (P, T_{AMB}) \xrightarrow{\mathcal{F}(\cdot)} T$$

where $freq$ is the vector containing the cores frequency, V_{dd} is the supply voltage, CPI is the workload running on each core, T and P are respectively the vector containing the temperature and the power dissipation of the cores, and T_{AMB} is the ambient temperature.

The first sub-function, $\mathcal{P}(\cdot)$, is a highly nonlinear and can be addressed separately in each core according to:

$$\begin{aligned} P &= P_{dynamic} + Pstatic = \\ & k_A \cdot freq \cdot V_{dd}^2 + k_B + (k_C + k_D\, freq) \cdot CPI^{k_E} + Z \cdot V_{dd} \cdot T^2 \cdot e^{\frac{-q \cdot V_t}{K \cdot T}} \end{aligned} \tag{4.1}$$

This equation, that will be called Power Model in the follows, has been obtained by performing a set of tests on each core of a real general-purpose multicore[1]. Without entering in more details (see the Appendix B and the references therein), we can notice that the core dynamic power depends nonlinearly on the frequency, since V_{dd} is a nonlinear function of the frequency, and sub-linearly on the CPI of the application (CPI and frequency are also coupled). Even if this is a simple empirical model that does not account for many secondary effects, many works in the state-of-the-art show that it can be used as a reliable basis to develop advanced models to effectively estimate the power consumption of different workloads (4) (5) (6).

Differently, the power-to-temperature model $\mathcal{F}$, that we will call Temperature Model in the follows, can be considered with good approximation linear. The temperature variation of each point of the die is affected by the power dissipated by the components on the chip, cores and caches primarily. The model can be obtained by using the well-known analogy between thermal and electrical models. The chip is decomposed into a great number of small cells, each one associated to an electrical RC circuit.

The same idea of decomposing the system model (i.e. the plant) into a linear and a non-linear part can be exploited also to build the prediction model. The central reason for adopting such a strategy relies on the possibility of exploiting the advantages offered by linearity. Indeed, we can design a linear MPC scheme which uses as manipulable variables the power consumptions instead of the frequencies and voltages, and then to convert the power into frequency with the Power Model. Without entering in the details of how the controller is structured, that is the argument of the next section, the Power Model translates the target frequencies, workloads and voltages (requested by an higher layer software as the operating system) in target power consumptions. The controller manages these target powers in order to maximize the performance respecting the temperature constraints and it returns the controlled powers. These latter are

[1] Intel® Xeon® X7350 (3)

given as input to the Power Model properly inverted to obtain again the frequencies. Fig. 4.1 clarifies this concept.

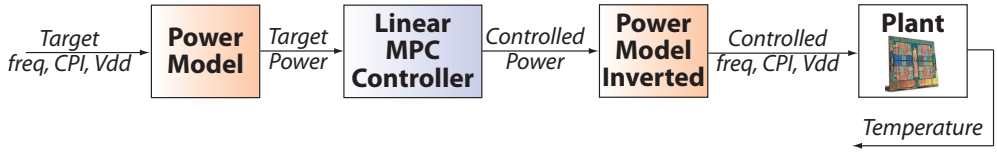


Figure 4.1: Conceptual control scheme

Concluding, the model chosen is a linear model which takes as inputs the power consumption of the cores (only the cores are controllable, that is we have no information of the power dissipated by caches) and it returns the measured temperatures of the cores (only the sensors measurements),

$$\begin{cases} x(k+1) = A \cdot x(k) + B \cdot u(k) \\ y(k) = C \cdot x(k) \end{cases} \tag{4.2}$$

where x is the $n \times 1$ state vector, y is the measured $p \times 1$ outputs vector (thermal sensors readings), u is the $m \times 1$ inputs vector (the power dissipation of each core and other information) and $A_{n\times n}$, $B_{n\times m}$, $C_{p\times n}$ are respectively the dynamical matrix, the input matrix and the output matrix. Assuming N the number of cores $p = N$. Thus, according to the granularity of performance counters, thermal and power sensors, our monitoring capability is at core level and within a core we assume uniform power and temperature distributions. We can abstractly visualize our model as a chip only composed by cores (see Fig. 4.2).

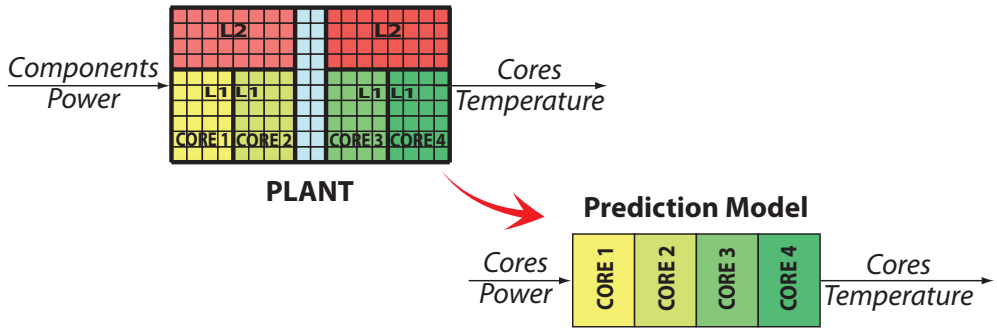


Figure 4.2: Abstract view of the model

What model order? The results in (8) highlight that the thermal dynamics of each core is characterized by two time constants: a faster one, at a few ms, is related to the silicon surface,

whereas the slower one, at a few seconds, is related to the heat spreader. This behavior, needs to be carefully accounted in model identification and control design. Therefore, our model should have at least two states for each core. On the other hand, the computational complexity requirements imposes a low order for the model. For this reason we chose as first attempt a second order model. In this case, the state dimension $n = 2 \times N$ where N is the number of cores.

How many inputs? The inputs represent the actions of the surrounding environment on the system and they can be classified in manipulated or not manipulated. The first can be modified, by the user or the control algorithm, in order to change the behavior of the system (e.g. the power dissipation of the cores in our case), whereas the second represents a "measured disturbance", that is an information that modifies the system behavior but that cannot be controlled arbitrarily (e.g. the ambient temperature). The greater is the number of inputs used to model a fixed behavior, the greater is the accuracy, but also the complexity.

It is also important to note that a measurable attribute can be defined as input for a system, but it could not be for another one, depending on what we are interested to model. As an example, if our goal is to find a unique thermal model for the whole chip, that we will call *"global model"* hereafter, then the inputs will be the power of the cores and the ambient temperature. In this case all these inputs are necessary to have an accurate model. Otherwise, if our goal is to model the temperature behavior of a single core, then the inputs will be the power of the core, the ambient temperature and the powers and temperatures of the other cores. In the following of the thesis we will refer to the model composed by the set of the single-core models as *"modular model"*. In this latter case the number of inputs increases with the number of cores. Thus, in order to understand which inputs are necessary and which are negligible we performed empirical tests. First we discarded the power of the other cores as possible inputs, due to the negligible contribution to the final temperature of the core. Recalling Fourier's law, the temperature of each core can be assumed dependent on its own dissipated power, ambient temperature and adjacent cores temperatures and powers (boundary conditions). This assumption is actually straightforward only for continuous time models. Focusing on discrete-time models, a larger coupling among cores has to be considered to account for the "chain of interactions" taking place during the blind intervals among samplings. Recalling again Fourier's law, the coupling among two cores will be inversely related to their distance and directly related to the sampling time period. Hence, the "equivalent neighborhood" of a core depends on the floorplan combined with the adopted sampling period. To verify this assumption we took

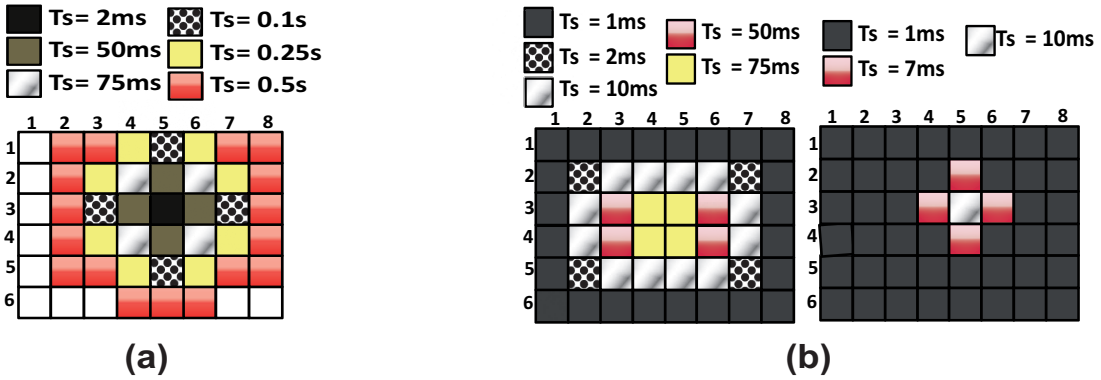


Figure 4.3: a) Single core thermal impact range, at different time windows; b) Multi cores thermal impact range, at different time windows;

an example loosely correlated with the Intel® SCC experimental architecture (11) (see also Appendix B). The floorplan is fully tiled with 48 core/regions, each with an area of $11.82mm^2$ and a maximum power consumption of 2.6W. We used this set-up with the HotSpot thermal analysis tool (12), and we made the following test. We stimulate with a power step the central core $(5,3)$ ("thermal attacker" core) while keeping all the other cores at zero power consumption. As result of the power step, the temperature of the attacker core increases, inducing a temperature raise in the neighbor cores. We are interested in measuring the range of thermal impact of the "attacker" core. To do that we look at the surrounding cores that raise their temperatures as consequence of the attacker. We call them "victims". We consider as victim only a core that raises its temperature of 1% the attacker temperature increment. Fig. 4.3a shows in black the attacker core and in different colors the victims after different time intervals. We can notice that the radius of thermal influence of the central core increases with the time interval: within 50ms it impacts only the closest core along the four cardinal directions, at 0.75s the majority of them is affect.

In order to test the behavior of the system when more cores are triggered we considered an increasing numbers of attackers starting from the perimeter cores going close to the victim one (now core $5,3$), after different timing intervals, we check the cores that increase their temperature more than the 1% of the attackers increment. In Fig. 4.3b we can see the results. We can notice that the neighborhood composed of one core in each direction is enough to prevent the core victim to be sensitive to the rest of the core temperatures within 10ms of time interval.

Technique	Type	Modular	Order
ARX	Linear	Yes	2 per core
H_∞	Linear	Yes	2 per core
POD	Linear	No	5 per 8 cores

Table 4.1: Prediction model characteristics vs. identification technique used to find it.

On the basis of these tests the power consumption of the core, the ambient temperature and the temperature of the adjacent cores constitute the inputs of the single-core model.

Concluding, our MPC solution will use a linear power-to-temperature model of the second order. For the global model we have chosen as inputs the powers of the cores and the ambient temperature, whereas for each single-core model the own power consumption, the ambient temperature and the temperatures of the adjacent cores (assuming a sampling time lower than 10ms).

In the following three subsections we show the techniques used in this thesis to extrapolate the prediction model. The table 4.1 shows the characteristics of the three techniques.

4.1.1 Distributed ARX identification

The first, and simplest, approach, used to find the prediction model, relies on the well-known ARX identification techniques (9) (10). An ARX discrete model can be written as,

$$y(t) = \alpha_s \cdot y(t-1) + \cdots + \alpha_1 \cdot y(t-s) + \beta_s \cdot u(t-1) + \cdots + \beta_1 \cdot u(t-s) + e(t) \tag{4.3}$$

where s is the order, y is the output, u is the input, α and β are constant parameters and e is a stochastic white process with null expected value.

However, the MPC controller will use the model as a predictor for the future output, therefore the equation (4.3) can be rewritten as,

$$y(t|t-1) = \alpha_s \cdot y(t-1) + \cdots + \alpha_1 \cdot y(t-s) + \beta_n \cdot u(t-1) + \cdots + \beta_1 \cdot u(t-s) \tag{4.4}$$

where $y(t|t-1)$ is the predicted output for the future time t based on the information available at time $(t-1)$. The main difference between a predictor and a simulation model is that the predictor uses the past measurements to estimate the future output, whereas the simulation model uses the past estimations.

The main idea of the ARX identification technique consists in learning the model parameters (α and β) by solving a least square problem that minimizes the prediction error. The

prediction error is the difference between the system response after we have applied proper training stimuli and the response estimated using past measurements.

$$min_{\alpha,\beta} \frac{1}{\#_{sample}} \sum_{t=s+1}^{s+\#_{sample}} (y(t) - y(t|t-1))^2 \tag{4.5}$$

where $\#_{sample}$ is the number of measurements used for identification, s is the model order, $y(t)$ is the measured output at time t and $y(t|t-1)$ is the estimated output at time t obtained using the equation (4.4).

As already discussed in the previous section, we chose a order $s = 2$ for the model. Moreover, we called this technique distributed because it solves N least square problems (one for each core) in order to identified the N single-core models forming the modular model. Two are the main advantages of this approach:

- it offers a low complexity solution to counteract the system identification computational cost in large multi-core systems. Indeed, in MIMO model the complexity for solving the least square problem explodes with the number of inputs.
- it perfectly fits with a distributed control solution, since each local regulator can directly exploits a single-core identified model.

As in the equation (4.4), the $i-th$ single-core model is a simple MISO model with a single output and multiple inputs,

$$\begin{aligned} T_i(t|t-1) = {} & \alpha_2 \cdot T_i(t-1) + \alpha_1 \cdot T_i(t-2) + \beta_{1,2} \cdot P_i(t-1) + \beta_{1,1} \cdot P_i(t-2) + \\ & + \beta_{2,2} \cdot T_{AMB}(t-1) + \beta_{2,1} \cdot T_{AMB}(t-2) + \beta_{3:3+dim(NEIGH_i),2} \cdot \\ & \cdot T_{NEIGH_i}(t-1) + \beta_{3:3+dim(NEIGH_i),1} \cdot T_{NEIGH_i}(t-2) \end{aligned} \tag{4.6}$$

where T_i is the temperature of the $i-th$ core, $P_i(\cdot)$ is the dissipated power of the $i-th$ core, T_{AMB} is the ambient temperature, $NEIGH_i$ is the set of neighbors of the $i-th$ core and T_{NEIGH_i} represent their temperatures. $\alpha_{1:2}$ and $\beta_{1:3+dim(NEIGH_i),1:2}$ are the identified parameters.

As already mentioned the core power consumption can be estimated from the core operating point and from the current workload characteristic using the Power Model (see (4.1) or it can be directly measured from power sensors present in recent MPSoC (11)).

The first step for the identification process is collecting data from the real system. The input sequence must be persistently exciting in order to ensure identifiability. We forced a Pseudo-Random Binary Sequence (PRBS) power input to each core, while probing the core

temperature. Then, the parameters α and β are derived by solving a least square problem that minimizes the prediction error, as the one shown in equation (4.5),

$$min \frac{1}{\#_{sample}} \sum_{t=s+1}^{s+\#_{sample}} (T_i(t) - T_i(t|t-1))^2 \tag{4.7}$$

Often it is convenient to translate the model just found in a state-space form,

$$\begin{aligned} x_i(t+1) &= A_{ARX,i} \cdot x_i(t) + B_{ARX,i} \cdot \begin{bmatrix} P_i(t) \\ T_{AMB} \\ T_{NEIGH_i} \end{bmatrix} \\ T_i(t) &= C_{ARX,i} \cdot x_i(t) \end{aligned} \tag{4.8}$$

Unfortunately, the identified models states do not have a physical meaning. To match the core temperature with the first state of each model we apply a change of coordinate transformation to obtain a matrix $C_{ARX,i} = [I_s \mid 0_s]$ where I_s is the s-dimensional identity matrix. We achieve that by representing our system in the equivalent observer canonical form. We use the observability matrix $O = [C_{ARX,i}; C_{ARX,i} \cdot A_{ARX,i}]$ as linear transformation to change the coordinates of the $(A_{ARX,i}, B_{ARX,i}, C_{ARX,i})$ model as shown below.

$$\begin{cases} A_i = O^{-1} \cdot A_{ARX,i} \cdot O \\ B_i = O^{-1} \cdot B_{ARX,i} \\ C_i = C_{ARX,i} \cdot O \end{cases} \tag{4.9}$$

Since each thermal model is a second-order model and each element of A_i, B_i, C_i can be expressed as the composition of a finite algebraical operation of the element of $A_{ARX,i}$, $B_{ARX,i}$, $C_{ARX,i}$, the above computation is negligible.

In a real system we expect to run the distributed ARX identification procedure as a self-calibration routine executed by each core. First, during the start up phase and then, on-line, each time the model behavior differs from the plant one.

We tested the performance of our technique using a high dimensional and accurate plant developed in Matlab using a finite elements approach (see Appendix B). In Fig. 4.4a we showed the comparison between the plant and the model temperature of core 1 obtained applying PRBS signals different from those used for the self-calibration routine. In Fig. 4.4b instead we validated our technique on plants with increasing number of cores. First, we have applied PRBSs on each core and then we have computed the model. Finally, we have measured the mean absolute error between the temperature of the system and the temperature of the model, both running PARSEC 2.1 benchmarks traces (13) (see Appendix B for more details). The resulting errors are lower than $1°K$ on average.

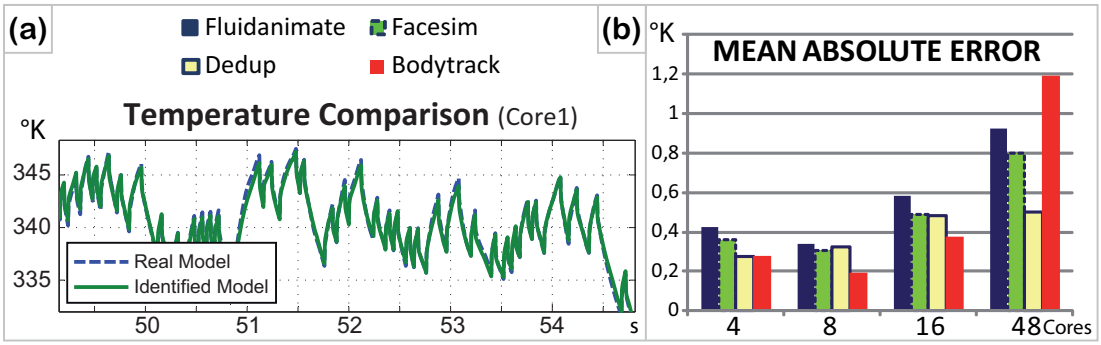


Figure 4.4: Self-calibration routine results

Notice that the modular model can be easily translated into a global model by composing the single-core models contributions.

4.1.2 H_∞ identification

We have seen that a common procedure to find the prediction model is the system identification. Basing on the observed data, i.e. inputs and outputs, the system identification approach finds the optimal parameters of the model minimizing a certain objective function. The model order chosen and the number of inputs determine the number of these parameters. Their values instead depends on the objective function chosen. In the previous section, for each core we minimized the mean square of the error between the measured and the estimated temperatures of the core. Also the H_∞ technique search for a modular model of the second order, but it uses a different cost function that favors the development of a more reliable and efficient MPC controller. For each single-core model it finds the parameters that minimizes the infinite norm of the error, i.e. the maximum error, but imposing the estimated temperature to be always greater than the measured one. This latter constraints allowed us to increase the robustness of the controller. Indeed, the MPC controller, which exploits the model, will always forecast a temperature value higher or equal than the real one. Therefore, at any time, the control decision returned by the controller will be either the optimal one or a less performing one. This optimization problem can be formalized as,

$$\begin{aligned} &\min_{s,\alpha_1,\alpha_2,\beta_1,\beta_2} \quad s \\ &s.t. \\ &T_i(t) - T_i(t|t-1) \geq -s \\ &T_i(t) - T_i(t|t-1) \leq 0 \end{aligned} \tag{4.10}$$

where $T_i(t|t-1)$ results from the second order input/output model,

$$T(t|t-1) = \alpha_2 \cdot T(t-1) + \alpha_1 \cdot T(t-2) + \beta_2 \cdot u(t-1) + \beta_1 \cdot u(t-2),$$

u is the inputs vector and β_2 and β_1 are vectors of appropriate dimensions. Notice that the two constraints impose the error to be negative or at least equal to 0.

Finally, if necessary, we may convert the model from the I/O space to the state space observable canonical form,

$$\begin{aligned} \begin{bmatrix} x_1(t+1) \\ x_2(t+1) \end{bmatrix} &= \overbrace{\begin{bmatrix} \alpha_2 & 1 \\ \alpha_1 & 0 \end{bmatrix}}^{A} \cdot \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \overbrace{\begin{bmatrix} \beta_2 \\ \beta_1 \end{bmatrix}}^{B} \cdot u(t) \\ T(t) &= \underbrace{\begin{bmatrix} 1 & 0 \end{bmatrix}}_{C} \cdot \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} \end{aligned} \tag{4.11}$$

It is worth to note that the parameters result from the data collected for a particular benchmark. However, it could exist particular benchmarks for which the temperature estimated is lower than the real one. For this reason we use a PRBS and check for the most typical benchmarks of the package PARSEC 2.1 (13).

In this thesis we also experimented an iterative procedure that uses the H_∞ approach. The aim is to find the model with the appropriate inputs. The algorithm starts considering all the possible inputs for the single-core model (the power of all the cores, the ambient temperature, and the temperature of all the cores), then it repeats the identification approach discarding at each step the inputs with negligible contributions or giving rise to incoherent results. As an example it is expected that the temperature contribution of a core decrease with distance. This approach could be necessary to cover the scenarios missed by the test shown in Fig. 4.3.

4.1.3 POD approach

In control theory it is extremely important to find a low-order model that approximates the behavior of the real system without impacting on the computational cost. The Proper Orthogonal Decomposition (POD) is an elegant technique for finding low-dimensional approximation of large-scale dynamical systems and data sets. The POD is also known as Principal Component Analysis (PCA), the Karhunen-Loève Decomposition (KLD), and the single value decomposition (SVD). It provides the optimal orthonormal basis for the modal decomposition of an ensemble of functions, such as data obtained in the course of experiments (14). Combined

with the Galerkin projection procedure, we can obtain a lower dimensional models of dynamical systems that have a very large or even infinite dimensional phase space.

Assume we have a very accurate model of the system obtained by using a finite elements decomposition approach, that is discretizing in space the infinite dimension system. This model has usually a high order and therefore it is prohibitive for the development of a controller. We may think to have the model we used as plant in our simulations (see Appendix B) that counts 360 states for a four-cores processor. We call as $T(s,t)$ the function which describes the temperature of the discretized system: s represents the s-th elementary volume (suppose they are K), and t is the time sample. It is always possible to express $T(s,t)$ as an infinite sum of coefficients $\Gamma = [\gamma_1, \ldots, \gamma_\infty]$ multiplied by the vectors of the orthonormal basis $\Phi = [\phi_1, \ldots, \phi_\infty]$,

$$T(s,t) = \sum_{k=1}^{\infty} \gamma_k(t) \cdot \phi_k(s) \tag{4.12}$$

An approximation $\hat{T}_(s,t)$ of $T(s,t)$ can be obtained using a basis Φ_M containing only M vectors. The POD technique find the M terms basis that gives the best approximation in a least square sense. In particular we want to minimize the distance of the data respect to their approximation, expressed as,

$$min \int_0^{Time} \|T(s,t) - \hat{T}(s,t)\|_2 \, dt \tag{4.13}$$

Solving this problem is equivalent to solving the eigenvalue problem,

$$Corr \cdot \Phi = \Phi \cdot \Lambda \tag{4.14}$$

where, according to the method of Sirovich, we may find the correlation matrix $Corr$ as,

$$Corr = \frac{1}{N_{sample}} \cdot T_{SNAP}^T \cdot T_{SNAP} \tag{4.15}$$

and T_{SNAP} is the $K \times N_{sample}$ matrix, the *snapshot matrix*, obtained collecting the temperature values of the K elementary volumes composing the model for N_{sample} time samples, i.e.

$$T_{SNAP} = [T(1)\ T(2)\ \ldots\ T(N_{sample})] \tag{4.16}$$

$T(i)$ is a column vector with K elements and $\Lambda = diag(\lambda_1, \ldots, \lambda_{N_{sample}})$ is the diagonal eigenvalues matrix.

The basis corresponds to Φ and it contains K eigenvectors of $Corr$. The correlation between the data and a generic ϕ_i is represented by the eigenvalues: the greater is the eigenvalue λ_i the greater is the ability of ϕ_i to approximate the data collected. Thus, we can choose M (the basis

dimension) by taking the ϕ_i $i = 1,\ldots,M$ vectors with the greatest eigenvalues. As an example we could find M using the function,

$$P_M = \frac{\sum_{i=1}^{M} \lambda_i}{\sum_{i=1}^{N_{sample}} \lambda_i} \tag{4.17}$$

Assuming the eigenvalues are sorted in a descending order, we start with $M = 1$ and then we increase M until we reach $P_M = 0.99$. The reduced basis is $\Phi_M = [\phi_1,\ldots,\phi_M]$ and $T(s,t)$ can be approximated with $\hat{T}(s,t) = \Phi_M \cdot \Gamma_M(t)$ where $\Gamma_M(t) = [\gamma_1(t),\ldots,\gamma_M(t)]$

Once we found the basis functions, we apply to the plant the Galerkin projection to find the low-dimensional model. Suppose our plant is a linear discrete-time model obtained by discretizing a partial differential equation via finite elements or finite differences,

$$\begin{aligned} \dot{x}(t+1) &= A \cdot x(t) + B \cdot u(t) \\ T(t) &= C \cdot x(t) \end{aligned} \tag{4.18}$$

where $C = I_{N_{sample}}$ and hence $x(t) = T(t)$. We obtain the reduced order model by substituting the $T(t)$ with its approximation $\hat{T}(t)$ and projecting the system onto the subspace defined by Φ_M by multiplying the matrices A, B, C by Φ_M. The final matrices of the reduced model are:

$$A_R = \Phi_M^T \cdot A \cdot \Phi_M \quad B_R = \Phi_M^T \cdot B \quad C_R = C \cdot \Phi_M \tag{4.19}$$

In Fig. 4.5 it is reported a simulation test where the distributed ARX and the POD approaches have been compared. We modeled the thermal behavior of a simple four-cores processor exploiting the well-known equivalence between thermal and electrical systems, as shown in Appendix B. We split the volume of the processor in two layers and then we decomposed each layer into a large number of cubic cells. To each cell we assigned an equivalent electrical circuit obtaining a model as in equation (4.18). The total number of cells is $K = 1728$ then the dimension of the state matrix, A, is 1728×1728.

The blue line represents the real temperature measured on the simulator, the green line is the temperature estimated by the low order model obtained with the distributed ARX approach. The single-core models has been composed to obtain the global model that counts 8 states (2 per core). Finally, the red line represents the model obtained with the POD technique. Despite it has a lower number of states, a total of 5, the temperature estimation are better. The mean value of the error for the POD respect to the real temperature is $0.44°C$, whereas the one for the ARX is $4.7°C$.

From these results the POD seems to be the preferable approach to use in MPC control solutions. However, it present two main disadvantages that made us lean towards the ARX

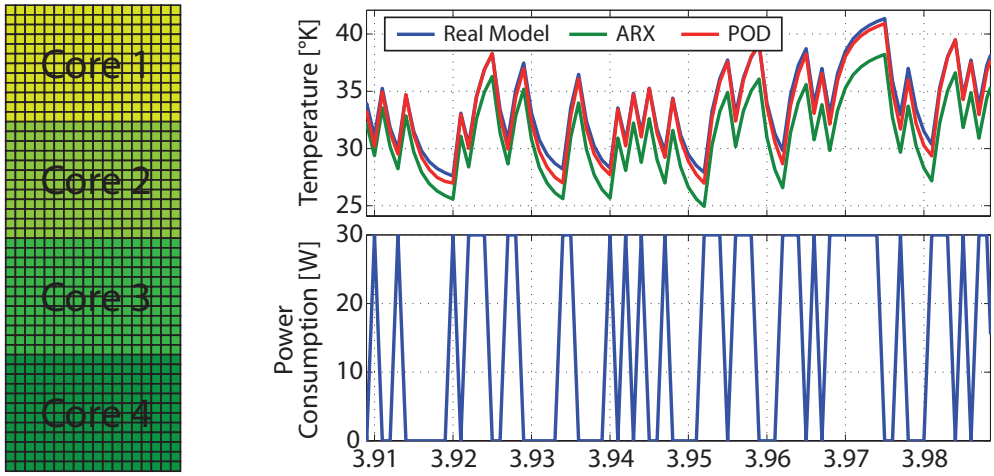


Figure 4.5: (a) Simulated processor, (b) Thermal and power Response of the core 1

and H_∞ approaches. First, this approach requires the knowledge of the thermal behavior of the real system in all its points, but this is usually impossible since on a chip are present only few sensors, say one per core. Also the use of softwares for the finite element decomposition analysis do not represent a viable way. Indeed, it is difficult to tune all the parameters to have the identical response of the real system. Moreover, preparing such a model is time consuming and the possibility of reuse the model for other processors is very low. Second, our MPC solution has the characteristic of being distributed, therefore it fits well with the distributed ARX and H_∞ approaches.

It is also worth to note that the ARX average error of $4.7°C$ is not significative since is obtained using the model as a simulator. Inside the controller, instead, it is used as a predictor, thus, starting from the real temperature value it has to forecast the future temperature for few time instants. In this scenario the ARX results accurate.

The code used for the POD method is shown in Appendix C

4.2 The Distributed Thermal Controllers

In this section we present the main contribution of this thesis, that is the distributed thermal manager designed using the MPC approach. The main idea of this solution relies in decomposing the MPC controller into a set of *local MPC controllers* each one supervising the temperature of a group of cores. Notice that the number of cores supervised by each controller can be different. In the follows we considered a fully distributed solution, that is each core is supervised

by a local controller.

The block diagram in Fig. 4.6 shows its global architecture, focusing on a single local controller which basically consists of four blocks: *f2P converter*, *P2f converter*, *MPC Controller* and *Observer*.

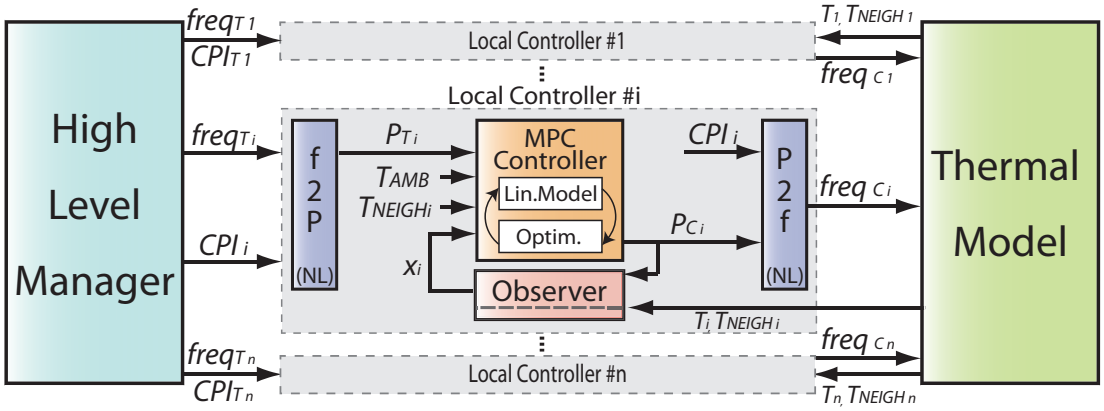


Figure 4.6: Thermal Controller structure

f2P and P2f converter blocks. These two blocks perform the conversion respectively from frequency to power and from power to frequency using the workload as additional information. The main role of the blocks is to encapsulate the nonlinear part of the frequency to temperature relation. The main advantage of this separation is the possibility of using a linear MPC controller instead of a nonlinear one which allows the use of more efficient and reliable algorithms for computing the optimal control solution. Both the conversion rely on the empirical relation (4.1). The *f2P converter* block accepts as inputs the target frequency and the workload coming from a high level manager. It takes as inputs the target core speed and the workload, respectively defined as $freq_{T,i}$ and CPI_i, and it returns as output the correspondent power consumption ($P_{T,i}$). Notice that in Fig. 4.6 the V_{dd} does not belong to the inputs set. However, as we have already mentioned, the voltage can be substituted by a nonlinear function of the frequency. The *P2f converter* block is the dual of f2P one. It receives as inputs the optimal $P_{C,i}$ and the workload of the core, and converts them to a consistent frequency value $f_{C,i}$. This optimal frequency is then applied to the core. The P2f conversion is obtained by inverting the (4.1). Unfortunately, the function is nonlinear, so for finding the root we need to use an iterative numerical method. We have chosen Brent's method (15) that combines the stability of bisection with the speed of a higher-order methods. In particular it uses the secant method

or inverse quadratic interpolation if possible and the more robust bisection method if necessary.

MPC controller block. It represents the center of the local controller. It uses the input measurements T_{AMB}, T_{NEIGH_i} and x_i to forecast the core temperature obtained by consuming all the requested powers $P_{T,i}$. As output, the block returns the controlled power $P_{C,i}$ which is equal to $P_{T,i}$, if the predicted temperature meets the temperature constraint (T_{MAX}), otherwise it is reduced. Clearly, the reduction must be as small as possible to maximize the performance. In order to define a local MPC controller able to manage this problem, two main elements must be designed: a model used for computing predictions and an optimization problem to find the optimal control decisions. The model estimate at each time instant, and starting from the current system temperature, the temperature of the core for a future time window. We have already discussed about the characteristics of the model and how to set its parameters. As an example this model could be identified by using a self-calibration routine, as shown in Section 4.1.1. Let us assume to have for each controller a single-core model of the form,

$$\begin{aligned} \begin{bmatrix} T_i(t+1) \\ x_{2,i}(t+1) \end{bmatrix} &= A \cdot \begin{bmatrix} T_i(t) \\ x_{2,i}(t) \end{bmatrix} + B \cdot \begin{bmatrix} P_i(t) \\ T_{AMB}(t) \\ T_{NEIGH_i}(t) \end{bmatrix} \\ T_i(t) &= C \cdot \begin{bmatrix} T_i(t) \\ x_{2,i}(t) \end{bmatrix} \end{aligned} \tag{4.20}$$

where T_i, P_i, T_{NEIGH_i} are respectively the temperature, the power consumption, and the temperature of the neighbors of the $i-th$ core, whereas $x_{2,i}$ represents the unmeasurable state of the model since we consider a second order model for each core.

The obtained forecasts are finally used to define the optimization problem:

$$\min_{P_{C,i}} \sum_{k=0}^{h_p-1} \|P_{C,i}(t+k|t) - P_{T,i}(t+k)\|_Q^2 \tag{4.21a}$$

$$s.t.$$

$$T_i(t+k+1|t) \leq T_{CRIT} \quad \forall k = 0,\ldots,h \tag{4.21b}$$

where $T_i(t+k+1|t)$ represents the temperature of the core predicted for time $t+k+1$ based on the information available at time t. The inequality (4.21b) imposes a hard constraint on the core temperature (T_i), while (4.21a) ensures the maximization of performance, minimizing the difference between the target power ($P_{T,i}$) and the power really assigned to the core ($P_{C,i}$). h_p represents the dimension of the prediction time window in sample instants (i.e. the prediction

horizon) and Q_i is a matrix that weights the importance of the square error elements. In our implementation we have chosen Q_i equal to the identity matrix and $h_p = 1$, guided by the characteristics of the system. Each sampling time, the solver yields the optimal solution, $P_{C,i}$, that minimizes the cost function and meets the constraints.

Hereafter, we briefly recalled two possible alternatives to solve the optimization problem presented above. The first is called *implicit* and provides an iterative algorithm that solves on-line the optimization problem at each time instant. The second, instead, performs the optimization off-line and it is called *explicit* (16). Both the methods have been introduced in Chapter 3. Anyway, both the approaches need for a reformulation of the problem in a standard quadratic programming (QP) form. The steps shown below transpose the approach used in Appendix A to our specific problem.

$$\min_{w_i} \frac{1}{2} \cdot w_i^T(t) \cdot H_i \cdot w_i(t) + g_i^T \cdot w_i(t) \tag{4.22a}$$

$$s.t.$$

$$M_i \cdot w_i(t) \leq b_i \tag{4.22b}$$

This problem is exactly equivalent to the optimization problem previously defined. Whereas $w_i(t)$ is the solution vector, the values of matrices and vectors H_i, M_i, g_i and b_i can be found starting from (4.21a) and (4.21b). Below are presented the mathematical manipulations to obtain them.

From (4.21a) we find H_i and g_i. The function can be rewritten in the vector form:

$$J = (P_{C,i} - P_{T,i})^T \cdot R_i \cdot (P_{C,i} - P_{T,i})$$

where $P_{C,i} = [P_{C,i}(t|t) \ \ldots \ P_{C,i}(t+k|t) \ \ldots \ P_{C,i}(t+h_p-1|t)]'$ and $P_{T,i} = [P_{T,i}(t) \ \ldots \ P_{T,i}(k+i) \ \ldots \ P_{T,i}(t+h_p-1)]'$ and R_i is the weight matrix (for example an identity). Note that we set $h_p = 1$, hence $P_{C,i} = P_{C,i}[k|k]$ and $P_{T,i} = P_{T,i}[k]$. Computing the products we have:

$$J = P_{C,i}^T \cdot R_i \cdot P_{C,i} - P_{C,i}^T \cdot R_i \cdot P_{T,i} - P_{T,i}^T \cdot R_i \cdot P_{C,i} + P_{T,i}^T \cdot R_i \cdot P_{T,i} \tag{4.23}$$

Using the matrix rule $(A \cdot B)^T = B^T \cdot A^T$ we can rewrite the previous equation as:

$$J = P_{C,i}^T \cdot R_i \cdot P_{C,i} - P_{T,i}^T \cdot R_i \cdot^T P_{C,i} - P_{T,i}^T \cdot R_i \cdot P_{C,i} + P_{T,i}^T \cdot R_i \cdot P_{T,i} \tag{4.24}$$

The searched value is $P_{C,i}$, hence $w_i(t) = P_{C,i}$. Then:

$$H_i = R_i \qquad g_i = -P_{T,i}^T \cdot (R_i^T + R_i)$$

The term $P_{T,i}^T \cdot R_i \cdot P_{T,i}$ can be omitted since it is constant at each time step.

Now we can manipulate the constraint inequality for obtaining M_i and b_i. Remembering that $h_p = 1$, (4.21b) becomes:

$$T_i(t+1|t) = C_i \cdot x_i(t+1) = C_i \cdot (A_i \cdot x_i(t) + B_i \cdot u_i(t) \leq T_{CRIT}$$

where $u = [P_{C,i}\ T_{AMB}\ T_{NEIGH_i}]'$. Making explicit $P_{C,i}$ we have:

$$C_i \cdot B_{1,i} \cdot P_{TC_i} \leq T_{CRIT} - C_i \cdot A_i \cdot x_i - C_i \cdot B_{2,i} \cdot [T_{AMB}\ \ T_{NEIGH_i}]'$$

where $B = [B_{1,i}\ B_{2,i}]$, $B_{1,i}$ is the column of B_i related to the input $P_{C,i}$ and $B_{2,i}$ is the completion of B_i. From the previous equation:

$$M_i = C_i \cdot B_{1,i}$$

$$b_i = T_{CRIT} - C_i \cdot A_i \cdot x_i[k] - C_i \cdot B_{2,i} \cdot [T_{AMB}\ T_{NEIGH_i}]'$$

It is worth to note that we could also consider the case of a lower bound on the power ($P_{MIN} \leq P_{C,i}$) writing:

$$M_i = \begin{bmatrix} C_i \cdot B_{1,i} \\ -1 \end{bmatrix}$$

$$b_i = \begin{bmatrix} T_{CRIT} - C_i \cdot A_i \cdot x_i[k] - C_i \cdot B_{2,i} \cdot [T_{AMB}\ T_{NEIGH_i}]' \\ -P_{MIN} \end{bmatrix}$$

In some of our simulations we do not take into account this power constraint since it is reasonable to assume that the real chip is designed to dissipate a power higher than P_{MIN} without incurring in thermal issues.

Once the QP formulation is obtained, one of the two approaches can be used to find the optimal solution. The implicit approach uses a quadratic programming solver. It is worth to note that the matrix b_i depends on the current state $x_i(t)$ that is time-variable. The same goes for $g_i(t)$ that depends on the requested power $P_{T,i}$ which is also time varying. Clearly, since the QP problem changes over time, the solving algorithm must be applied on-line at each time instant. Although efficient QP solvers based on active-set methods and interior point methods are available, the computational overhead for finding the solution demands significant on-line computation effort. Assuming that the solution of the QP problem does not change much from the solution obtained at the previous iteration, we can reduce this effort by using an active set algorithm capable of finding the new solution starting the search from the previous one (hotstart). This algorithm is implemented in the open-source library qpOASES (17).

As previously mentioned in Chapter 3 and in Appendix A, another way to reduce computational burden is to use an explicit approach that solves the QP problem off-line for all possible values of $x_i(t)$. The problem solved in this way is commonly called multi-parametric QP (mp-QP). By treating $x_i(t)$ as a vector of parameters, the optimal solution is an explicit function of the state ($P_{C,i}(x_i(t))$) with the property of being continuous piecewise affine (16). In other words it is possible to partition the state-space into convex polyhedral regions, each one with its own optimal linear control law. At each time instant, $x_i(t)$ lies in one and only one of these regions. Similarly to a Look-Up Table (LUT), knowing the current state and by checking region boundaries it is possible to find the solution using the linear expression:

$$P_{C,i}(x) = F_{r_i} \cdot x_i(t) + G_{r_i} \tag{4.25}$$

where r is the region index and $F_{r,i}$ and $G_{r,i}$ are the corresponding gain matrices for each core i.

Even though on one hand the use of the explicit solution reduces the computational overhead, on the other hand it increases the memory usage for storing the data. Similarly to the overhead in the implicit solution, the number of regions increases with the problem complexity (18).

Observer. The aim of this block is to estimate the unmeasurable state component ($x_{2,i}$) from the temperature measurements. Indeed in Section 4.1 the model is shown to have two dynamics, but only one thermal sensor per core is available. By knowing the model and taking as inputs the temperature of the core T_i and $P_{C,i}$, the observer estimates the state components $x_{2,i}$. Subsequently, this state will be used by the MPC block as initial state for the prediction of the temperature at the next time instant. We have used for each i-th core a classic Luenberger identity observer defined as:

$$\hat{x}_i(t+1) = A_i \cdot \hat{x}_i(t) + B_i \cdot u_i(t) + K_i \cdot (T_i(t) - C_i \cdot \hat{x}_i(t))$$

where $\hat{x}_i$ is the estimation of the state, A_i, B_i, C_i are the matrices of the prediction model, the same used by the MPC controller and u_i is the input vector containing $P_{C,i}$, T_{AMB}, T_i, T_{NEIGH_i}. K_i is the observer gain matrix that is a degree of freedom for the designer. Its elements are chosen to asymptotically stabilizing to zero the dynamic model of the estimation error $e(t) = \hat{x}_i(t) - x_i(t)$ characterized by the state matrix $(A_i - K_i \cdot C_i)$. To do that we act on K_i to move the poles of the error model inside the unitary circle. In particular we place the poles closer to the

center than the poles of the model to have faster dynamics and thus a faster convergence of the error to zero. The only requirements to arbitrary move the poles is the observability of the pair (A_i,C_i) that is assured by the physics of the system, in fact each dynamic directly or indirectly affects the model output temperatures.

4.3 Design choices motivations

This section aims to show simulation results in order to justify our design choices. The tests have been performed on a Matlab/Simulink thermal simulator, as the one carefully described in Appendix B. This allowed us to be more rapid in switching from a control law to another or simply to change the chip floorplan. The Fig. 4.7 shows the architectures of the chips simulated during the tests.

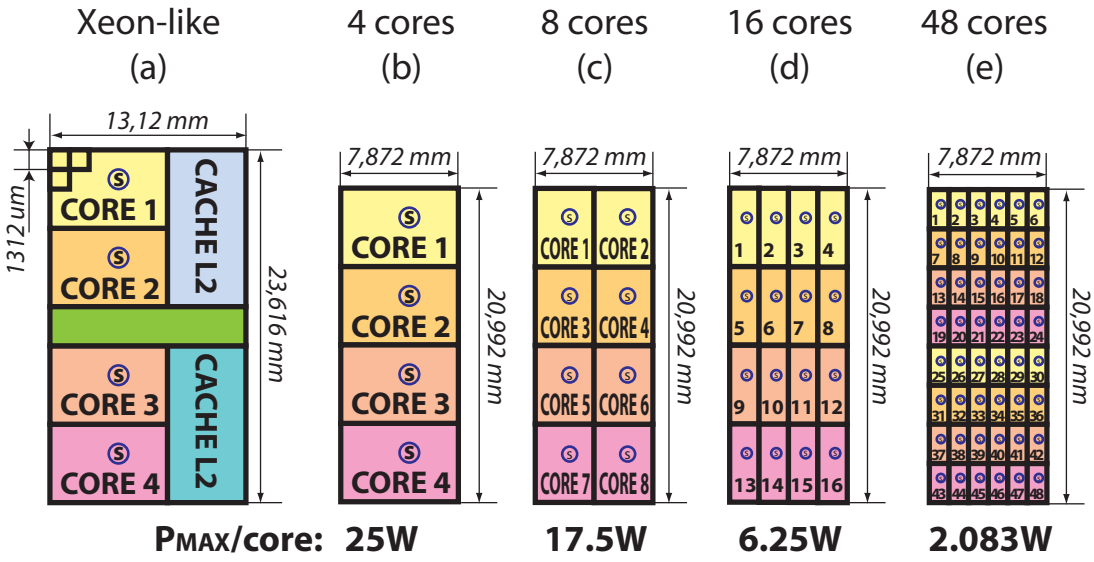


Figure 4.7: Simulation layout of the chip used in the tests

The majority of tests have been performed using as a layout an accurate Xeon® X7350 like four cores model calibrated on real HW (for more details see Appendix B). For simplicity and without loss of generality the tests comprising chips with more than 4 cores were performed on chips without caches. Note that this is not a limitation because all the comparison tests are performed under the same simulation conditions. Finally in the 8, 16 and 48 cores layouts we scaled down the maximum power of each core in order to keep constant the power density on

the chip and we reduced the spatial discretization of the thermal simulator to keep constant the number of finite elements per core.

In the following subsections, we show:

1. a comparison between our distributed solution and a centralized solution;
2. the importance of an accurate prediction model;
3. the importance of an accurate power model ;
4. a comparison between our distributed solution and a PID based solution

4.3.1 Distributed solution vs. Centralized solution

Recently, literature presented many variants of MPC-based DTM schemes for managing the thermal issue of MPSoCs. Despite the use of different formulations and architectures, see e.g. (18)(19), a quite common idea is to modify the frequency of each core with the twofold objective of meeting the temperature constraint and tracking the target frequency requested by a high level SoC manager. This purpose can be obtained by solving an optimization problem whose prototype is:

$$min \sum_{k=0}^{h-1} \| f_T(t+k|t) - f_C(t+k|t) \|_Q^2 \tag{4.26a}$$

$$s.t.$$

$$T_j(t+k+1|t) \leq T_{CRIT} \quad \forall j = 1,\ldots,p \quad \forall k = 0,\ldots,N \tag{4.26b}$$

where $f_C = [f_{C,1}, \ldots, f_{C,N}]$ is the set of frequency assigned to the cores, $f_T = [f_{T,1}, \ldots, f_{T,N}]$ is the set of target frequency, $Q_{N\times N}$ is the weight matrix, and T_{CRIT} and T_j, $j = 1,\ldots,p$ are respectively the critical temperature and the temperatures on the p points selected to represent the thermal state of the die (i.e. the sensors). The notation $T_j(t+k+1|t)$ means the temperature estimated for the future time $(t+k+1)$ based on the information available at time t which implies the existence of a thermal model relating the frequency of all the cores (and the ambient temperature) with future cores temperatures. Notice that also in this case according to (4.1), it is possible to replace respectively the target (f_T) and the current (f_C) frequency with the target (P_T) and the current (P_C) power consumption. The controller takes as inputs the *CPI* and the target frequency of all the cores. It translates these two attributes in power consumption

requirements. After having solved the optimization problem, the controller returns the optimal power consumption of all the cores which is subsequently translated in frequency inputs for the thermal simulators.

This control approach is usually referred to as *"Centralized"* because the control decision, i.e. the frequency assigned to each core, results from the solution of a unique problem running on a single core. However, this solution is problematic to execute on a large scale system because of the computational complexity. As the number of cores increases (this is the expected trend for the future), the time necessary to solve the problem (4.26) exponentially evolves, causing a loss of controllability on the system, maintenance problems and a reduction of fault-tolerance to sensors and actuators failures (20).

These considerations motivate the use of our distributed solution characterized by a set of local controllers each one supervising the temperature of a group of cores (that is a single core in the problem described in (4.21)).

We have tested the performance of the distributed and centralized solutions using this latter as yardstick, since it guarantees the maximum achievable thermal controller performance, reflecting in minimal QoS degradation due to thermal constraining. The first test consists in measuring the performance of the two solutions as maximum overshoot respect to the critical temperature $T_{CRIT} = 330°K$ and percentage of time spent over the aforementioned threshold (we consider the limit violated when temperature exceeds $330.1\,°K$). We performed these tests using the Xeon-like chip architecture (see Fig. 4.7a) and applying to the cores different PARSEC 2.1 (13) benchmarks. The same test has been also repeated on Simics a full-system virtual platform to gauge with high accuracy the complex interdependencies between control actions and workloads, which are lost when using trace-based workload models (see Appendix B). Fig. 4.8 shows that the distributed and centralized solutions have comparable performance. In

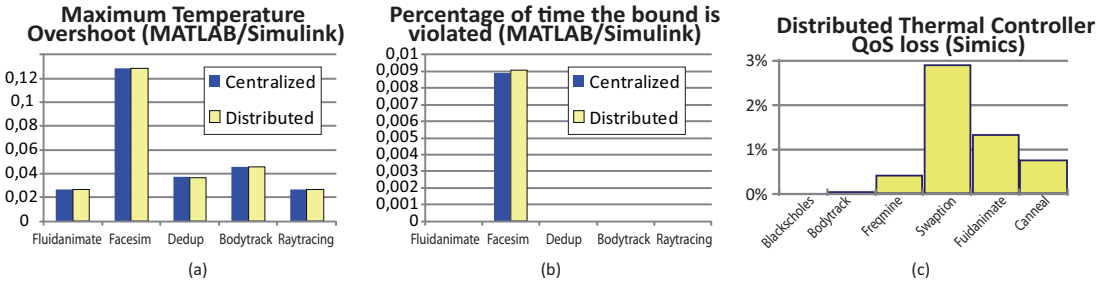


Figure 4.8: Centralized vs. Distributed performance comparisons: (a) Maximum overshoot, (b) Percentage of time the bound is violated, (c) Distributed solution QoS Loss

Centralized MPC Complexity

	MPC_explicit	MPC_implicit
4	81	7,70
8	6561	9,00
16	OUT	24,20
48	OUT	85,50
	# regions	time (us)

(a)

Distributed MPC Complexity (single core)

f2P	0,061	µs (time)
Observer	0,743	µs (time)
MPC (Impl)	4,690	µs (time)
MPC (Expl)	2	# regions
P2f	1,188	µs (time)

(b)

Figure 4.9: Scalability and complexity reduction results

Simics test we use a QoS Loss metric that quantifies the controller quality of service (QoS) degradation due to thermal constraint. We decided to compute it as the mean squared error between the target frequency (f_T) and the one applied to the system by the controller (f_C). Even though performance obtained by the centralized solution represents an upper bound for the distributed one, in all the tests we attained a close approximation of the optimal control actions computed by the centralized controller.

Previous tests showed that our distributed solution performs as the centralized one in terms of controller quality. However, its main properties remain hidden. Hereafter, we provided some results to highlight its main benefits in terms of computational complexity and scalability.

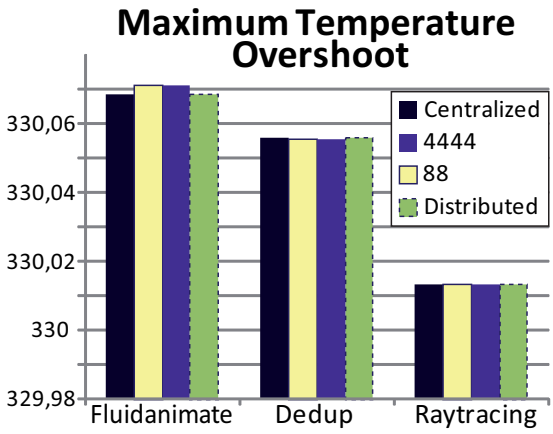
The tests consist in comparing the distributed and the centralized solutions for increasing number of cores (we used the Fig. 4.7b-e models). To evaluate the complexity we use as metrics both the execution time, for the implicit formulation, and the number of regions for explicit one. As a matter of fact, (18) shows that the number of operations depends logarithmically on the number of regions. Instead, for the implicit solution, we provide the mean computational time necessary to solve the QP problem at each iteration. This time has been obtained running a C/C++ code version of the control algorithm on a 2.4GHz dual-core processor. The Fig. 4.9a shows the relation between the computational complexity and the number of cores in the centralized MPC solution. The time spent to solve the QP problem and the number of regions exponentially increases with the number of cores in the worst case. In particular, as asserted in (16) and in Section 3.3, the regions number increases with the number of states, decision variables and constraints. For a chip with 16 or 48 cores we have been unable to compute them for memory limitations.

Fig. 4.9b instead shows the complexity of our distributed solution: whereas the centralized solution grows exponentially, the distributed one globally increases linearly. Moreover, the

complexity of a single controller node remains constant regardless the number of cores. Indeed, each one has only two regions and spends on average only $4,69\ \mu s$ to solve the QP problem. The same figure also shows the execution time for the frequency-to-power conversion and the observer estimation. Notice that their impact on the global execution time is negligible ($< 2\mu s$).

In addition, we need to consider that our distributed controller is naturally parallel (i.e. one local controller running on each core), while parallelization of the centralized controller is far from obvious. As consequence of our distributed implementation, each thermal controller can be stopped autonomously without impacting on other controllers performance. This avoids periodically waking-up of idle cores to execute the controller routine. Moreover, the small power consumption of cores, when power gated, ensures that no thermal emergencies can occur on their surface. Consequently, they do not need to be thermally controlled. As the core is waken up, the controller will pay a temporally QoS loss due to the partially valid initial state vector. Properties in the Luenberger observer ensure this error fast converges to zero. This is not the case of centralized MPC where the core in which the controller is running always need to be periodically waken-up since other cores might be under thermal emergencies. This introduces extra energy penalties in centralized solution applicability.

The set of tests performed in order to compare the centralized and the fully distributed solutions conclude with the exploration of intermediate solutions. Indeed, it could be convenient to group cores in set with a cardinality greater than one. The data in Fig. 4.10 refers to a 16 cores chip where different degrees of control distribution are applied. In detail, the comparison is between the centralized solution, the fully distributed solution and two semi-distributed solutions, in which local controllers manage respectively groups of four and eight cores. As expected, results shows comparable performance in both scenarios: the maximum constraint violation differs by few one hundredths of degree (see Fig.4.10a). The percentage of time for which the temperature violates the constraint is omitted since the results are all zeros. Fig.4.10b, instead, shows the controller complexity of each single group. We can notice that the number of regions exponentially increases with the number of cores supervised by each controller. Also the time spent to solve a QP problem increases as well. Moreover, even though the sum of solving times of each local controller in distributed or semi-distributed solutions are higher than the centralized one, due to its parallel execution, the completion time of the distributed implementations is equal to the one of a single local controller.



Complexity per single controller

	time *(us)*	# regions
Centralized	*29.119*	*OUT*
88	*15.4162*	*6463*
4444	*8.65926*	70
Distributed	*4.55575*	2

(b)

Figure 4.10: Scalability by grouping

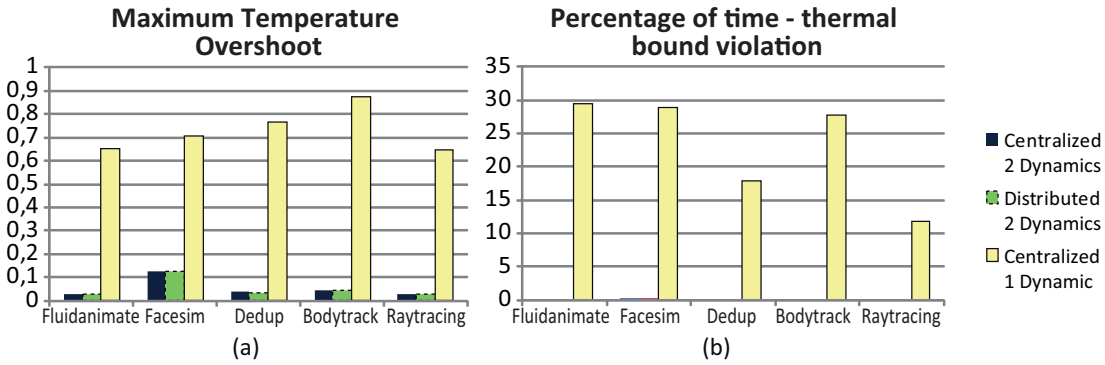


Figure 4.11: Prediction model: 1 dynamic per core vs 2 dynamics per core

4.3.2 Model accuracy

We have mentioned many times the importance of the model accuracy in order to have good predictions. In this subsection, we tested how performance worsen as the model change. We have compared the centralized solution with a second order global model with the same solution, but with a simpler first order global model as in (19). Notice that in this case we used a centralized solution instead of the distributed solution for the sake of simplicity. Anyway, we have already shown that the performance of the centralized and distributed solutions are similar, therefore the results of the test hold also for the distributed solution.

The results in Fig. 4.11 show, as we expected, a worsening of the controller performance. The first order model we use considers only the slower dynamics, that are the dominant ones. The absence of faster dynamics causes oscillations in the controlled temperature that causes a

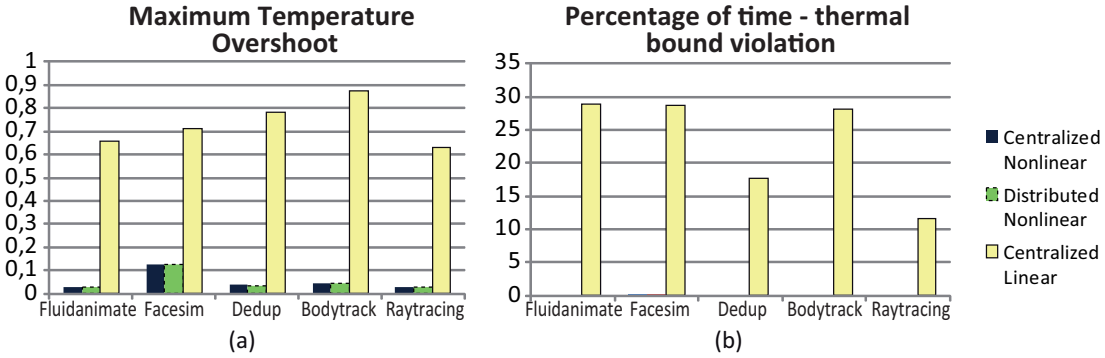


Figure 4.12: Nonlinear vs. linear power model function (P2f and f2P)

large increase in the percentage of time the constraint is violated. The controller sees that the model behavior is characterized by slow dynamics so in proximity of the temperature limit it gives a strong control action to rapidly decrease the temperature. Differently, the plant, having both slow and fast dynamics, responds with larger temperature decreasing. The controller senses this unexpected drop in the temperature and, again, it reacts with a control action larger than what needed. This gives rise to a limit-cycle oscillating behavior which is far from ideal.

4.3.3 Power Model accuracy

The accuracy of the $f2P$ and $P2f$ converters is extremely important to have a reliable and efficient controller. A wrong estimation of the power consumption could lead to performance degradations as consequence of suboptimal control decisions. The tests we have performed show an extremely worsening of the results when the nonlinear Power Model is substituted with its linearization. In the first test we compared the centralized solution with a similar one where a linear function has been used for $f2P$ and $P2f$ conversions. As in (19) we use the best first degree polynomial fitting the nonlinear function. The test has been performed on the Xeon-like processor since the number of cores does not impact on the results, but only emphasizes the results attained.

The results in Fig. 4.12 show that the MPC with linear $f2P$ function shows globally lower controller performance. It has bigger maximum overshoot, and the time percentage the constraint is violated is significantly higher. This is because the linear function transforms the controller output power, P_C, into an input frequency for the plant model which is higher or lower than the frequency really derived by the controller. Notice that performance is influ-

enced by the previous results since the frequencies are maintained closer to the input target frequencies because the temperature constraint is violated more than the other cases.

A reader could complain about the fact that the Power Model we used as comparison meter with the linearized one is the same used inside the thermal simulator. Moreover, the Power Model has been identified through empirical experiments, hence there are no guarantees about its accuracy. The next test shows that even stressing the nonlinear function inside the controller with parameters errors the performance slightly degrades.

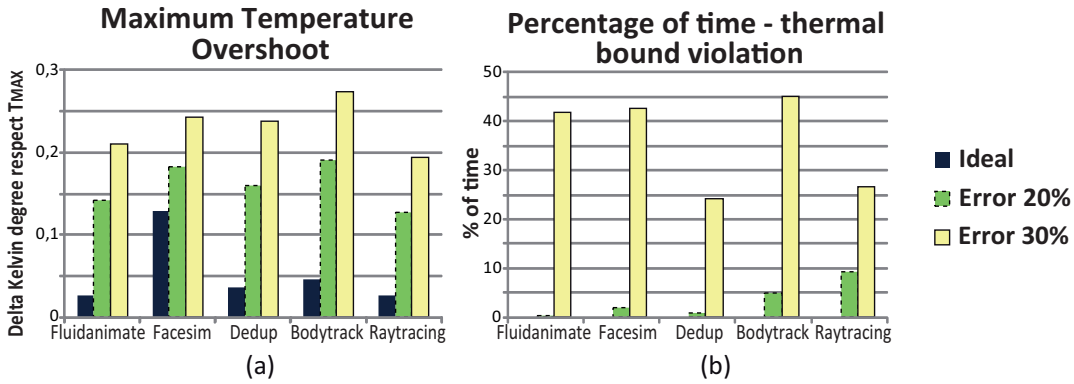


Figure 4.13: Sensitivity test on the Power Model

In order to quantify how the accuracy of the power model impacts on controller performance, we have simulated different accuracy levels by introducing artificial errors in the Power Model parameters. This is done while keeping the simulation model unchanged. Same errors are applied both in $f2P$ and $P2f$. Fig.4.13 shows the performance of the distributed controller under two different levels of accuracy. Respectively we introduced a 20% and a 30% of error on all the parameters of $f2P$ and $P2f$ functions. The results show that despite a worsening in performance, our controller is robust to power model accuracy lack. Indeed, with 20% of error on internal parameters the maximum overshoot is still lower than $0.2^{\circ}K$ and the percentage of time the temperature violates the constraint is below the 10%. Moreover, these results highlight the importance of having an accurate power model for control performance. This reflects in the expectation that in future chip releases, manufacturers will provide detailed information on the f2P relation or will embed accurate power sensors in the final HW to facilitate its identification and its recursive recalibration.

It is also important to note that a recursive recalibration at run time of the linear part of the model would help to improve performance compensating also the uncertainties on this

function, for instance leakage power to temperature dependency.

4.3.4 Distributed solution vs. PID solution

In our control solution we have used a sophisticated approach based on MPC, but is this better than a classical PID one?

In this subsection we provided a comparison between our distributed MPC solution and a distributed PID-based solution. The test has been performed on the Xeon-like processor. We designed the PID solution taking the cue from the controller developed in (21) for a single processor (for the sake of simplicity and without loss of generality we control it in power and not in frequency). By using classical design rules, we approximated the model of each core with a first order system with delay. Then, we developed the PID controller for each core by using Ziegler-Nichols and Cohen-Coon tables. The single PID has the time-continuous form:

$$PID_i = K_c \cdot (1 + \frac{1}{T_i \cdot s} + T_d \cdot s)$$

where K_c, T_i and T_d are respectively the constants of the proportional, integral and derivative actions. As shown in Fig. 4.14c, the PID solution is power driven and not frequency driven; each PID controller takes as inputs the error between the real temperature and T_{CRIT} and returns as output the correction that needs to be applied to the target power (P_T) to meet the temperature constraint.

Notice that positive corrections should be avoided since the core would run faster than the frequency suggested by the high level SoC manager, causing a useless power consumption. For this purpose a saturation $[-inf, 0]$ has been introduced to the control action outgoing from the PID. Moreover, to prevent the negative effect of the saturation on the integral action we force this latter to freeze when controller output saturates the actuator. The final value for our Ziegler-Nichols based PID are: $K_c = 1.048e+003$, $T_i = 1e-003$, $T_d = 2.5e-004$ discretized with a sampling time $T_s = 0.5ms$. It is relevant to remark that we chose a sampling time finer than the MPC solution one that is $1ms$ in order to reflect the lower internal complexity of the PID solution. Moreover, the PID controller does not include the quantization. The Fig. 4.14a-b use the usual metrics to show the benefits of a MPC-based solution on performance. This results are even more relevant if we consider that the sampling time of the MPC controller is higher than the one used to discretize the PID.

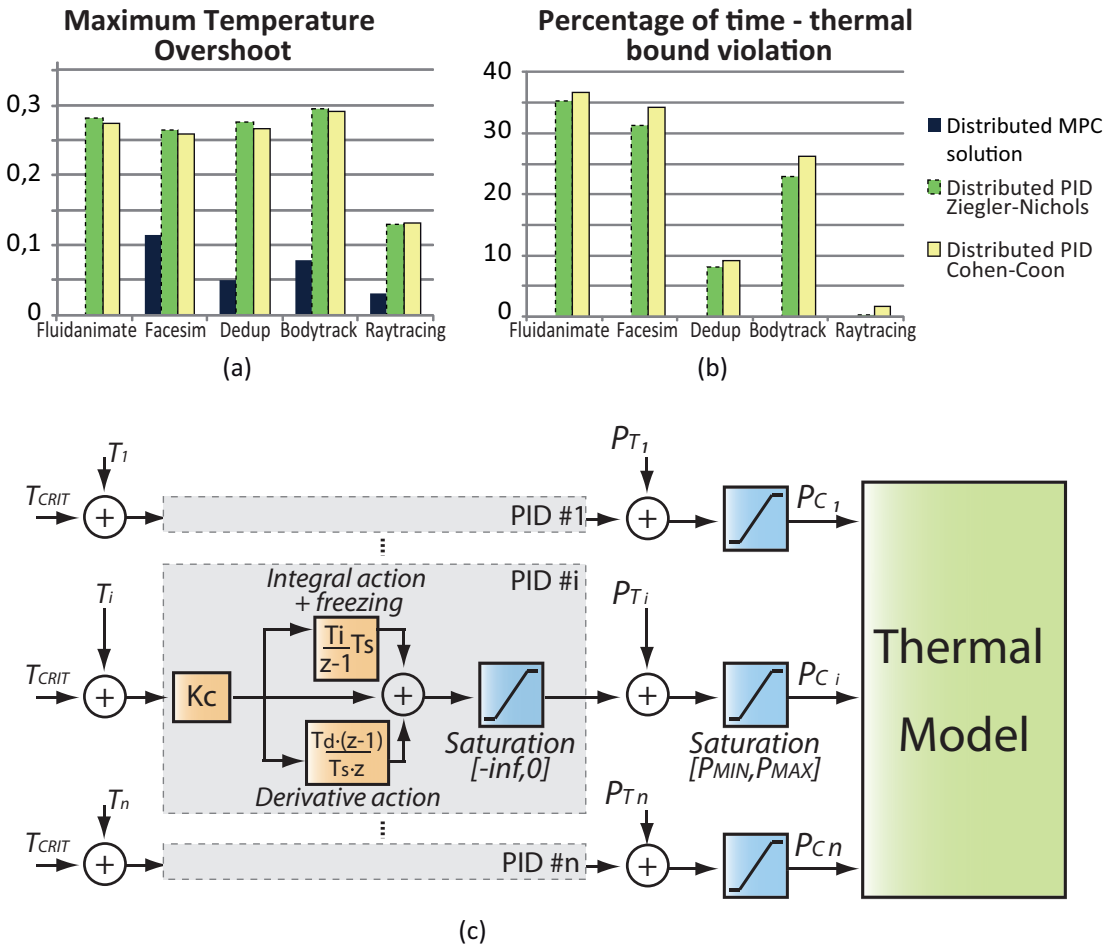


Figure 4.14: Distributed MPC solution vs. distributed PID solution

4.4 Control feasibility and other properties

In order to build an effective MPC scheme, model properties should be carefully checked to test the feasibility of the considered control problem (centralized or distributed) and simplify the design of the controller.

The aim of this section consists in verifying feasibility of the control solution over different prediction horizons (h_p), taking into account that the constraint on the critical temperature must be fulfilled by all the points on the die. The feasibility problem is usually disregarded in the specific literature on the thermal control of MPSoC and no formal proofs guaranteeing the meeting of the constraints are stated. For this reason we studied the properties of the thermal models starting from the general Partial Differential Equation (PDE) which represents the heat conduction in a volume (the so called *heat equation*) (22) (23). Then, we used the properties of the PDE thermal model to obtain general results, which are not affected by possible side-effects due to temporal and spatial discretization.

However, it is worth to remark that the model is heavily influenced by uncertainties and unavoidable simplifications required to make the problem treatable (e.g. the reduction of the originally infinite dimension model to a handy finite dimension). These simplification negatively affect the model accuracy threatening feasibility and performance. Nevertheless, the spatial and temporal discretizations, usually adopted in control/simulation-oriented modeling, are necessary for model simplification even though they introduce an accuracy worsening. The model has to be simple and accurate to reduce the computational effort and capture all the die temperatures without missing the maximum one. Thus, rules for model discretization must be stated. We exploited physical properties of thermal systems to capture the maximum temperature on the die, keeping the problem size as low as possible for control and simulations.

4.4.1 The thermal problem

In order to provide a general dissertation on the feasibility problem valid for the whole class of thermal systems, we assume to have a very accurate model based on PDEs, which perfectly describes the thermal system behavior. The PDE we considered is the well-known heat equation that models the heat conduction in a solid body. It belongs to the family of the second order linear PDEs, and in particular it is a parabolic PDE. The heat equation derives from two important postulates. The first states that the internal energy u $[J]$ of a body depends on the

spatial distribution of temperature and the thermal capacity of the material as follows:

$$u(x,t) = u(T(x,t),x) = c_T(T(x,t),x) \cdot T(x,t) \tag{4.27}$$

where for the sake of simplicity the thermal capacity c_T is assumed constant.

The second is the so-called Fourier's law of heat conduction stating that the heat transfer rate per unit area is proportional to the normal temperature gradient, i.e.

$$\vec{J_q}(x,t) = -k(T(x,t),x) \cdot \nabla T(x,t) \tag{4.28}$$

where $\vec{J_q}(\cdot)$ $[W]$ is the heat flow, $\nabla T(\cdot)$ is the temperature gradient and $k(\cdot)$ is a positive proportional term dependent on material conductivity and temperature. Note that the minus sign is a consequence of the Second Principle of Thermodynamics which states that heat must flow downhill on the temperature scale or equivalently the entropy of a closed system always increases or remains constant.

In agreement with the First Principle of Thermodynamics, energy is conserved, thus the total system energy is equal to the amount of energy transferred across its boundary by means of heat and work ($\partial u = \partial Q - \partial W$). Assuming no working energy is transferred, the variation of internal energy of the body will depend only on the heat flowing into the body through the boundary and on the thermal power generated by internal sources $\overline{q}(\cdot)$:

$$\frac{\partial u}{\partial t} = -\nabla \cdot J_q + \overline{q}(x,t) \tag{4.29}$$

Putting together constitutive equations (4.27) and (4.28) with energy balance equation (4.29) and assuming, according to common approaches, c_T and k as constants, we obtain the heat equation:

$$c_T \cdot \frac{\partial T(x,t)}{\partial t} = k \cdot \nabla^2 T(x,t) + \overline{q}(x,t) \tag{4.30}$$

where ∇^2 is the divergence of the gradient (Laplacian) of the temperature.

To completely define a thermal system this equation is not enough. First the boundary of the region of interest must be specified. Let the volume $V \subset \mathbb{R}^3$ be the region we want to study, and let ∂V be its boundary surface (such that $\partial V \cup V = \overline{V}$). Secondly, the initial condition $T(x,0)$, representing the system state at $t = 0$ must be set. Finally, the boundary conditions must be defined. In this regard, the most common choices are Dirichlet boundary conditions (DBCs), where $T(x,t)$ is known on the boundary (i.e. $\forall x \in \partial V, \forall t \geq 0$) and Neumann boundary

conditions (NBCs), where the normal derivative $\partial T/\partial n = \hat{n} \cdot \nabla T(x,t)$ is specified ($\hat{n}$ is the normal to the boundary). A general heat problem is given by,

$$\begin{cases} \frac{\partial T(x,t)}{\partial t} - \alpha \cdot \nabla^2 T(x,t) = q(x,t) & x \in V \quad t \in [0,\tau[\\ T(x,0) = T_0(x) & \forall x \in V \\ \begin{array}{ll} & T(x,t) = T_{\partial V}(x,t) \\ or & \hat{n} \cdot \nabla T(x,t) = -J_{\partial V}(x,t) \end{array} & \forall x \in \partial V \quad \forall t \in [0,\tau[\end{cases} \tag{4.31}$$

where$[0,\tau[$ is the time interval, $\alpha = k/c_T$ is a constant and $q(\cdot)$ is equal to $\bar{q}(\cdot)$ except for a multiplicative constant value. The second and third equation define respectively the initial temperature and the conditions on the body boundary, i.e. the DBCs and the NBCs. Moreover, in both formulations, initial and boundary conditions need to be set according to the Third Principle of Thermodynamics, i.e. forcing the temperature evolution to be always positive. Hence, expressing the temperature in Kelvin degrees, the following further constraints have to be considered:

$$\begin{array}{lll} T_0(x) \geq 0 & \forall x \in V\ , & \\ \begin{array}{l} \\ or \end{array} & \begin{array}{l} T_{\partial V}(x,t) \geq 0 \\ J_{\partial V}(x,t) \leq 0 \end{array} & \begin{array}{l} \forall x \in \partial V,\ \forall t \in [0,\tau[\\ \forall x \in \partial V,\ \forall t \in [0,\tau[\ \ s.t.\ \ T(x,t) = 0 \end{array} \end{array} \tag{4.32}$$

where the second equation is for DBCs, while the third one is for NBCs.

Remark 1. *It is also worth to note that the classical Fourier-based heat equation, used in this work to model the thermal systems, assumes an infinite propagation speed for the heat. To take into account the finiteness of heat propagation speed we should upgrade the equation from the parabolic to an hyperbolic form considering the second time derivative, or to use a nonlinear parabolic equation like the Porous Medium Equation. Anyway, considering the transmission speed of the heat in silicon and the small sizes of chips, the classical linear equation still remains a good approximation.*

In order to better understand the latter remark, consider an arbitrarily long bar in a monodimensional space. If we apply a heat pulse in its middle, according to Fourier equation, temperature change at the borders instantaneously. However this behavior contrast with the relativity theory because it assume that the speed of information propagation is faster than the speed of light in vacuum. This explains why a thermal system modeled with the Fourier equation does not present an overshoot in correspondence of a pointwise heat reduction when temperature are at the equilibrium. In simpler words, consider the same bar with a constant initial temperature distribution a ε below the critical threshold and assume that the boundary conditions are known (see Fig.4.15). At a particular instant we put a small power in the middle of the bar. From the

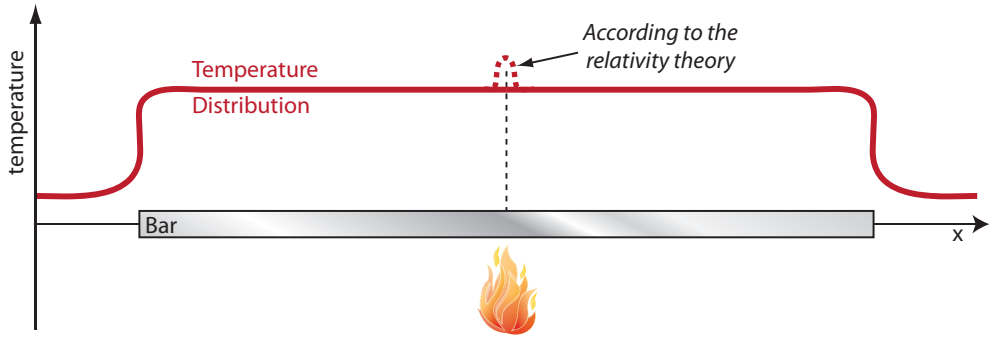


Figure 4.15: The bar example

Fourier heat equation we would have $\dot{T}(x,t) = q(x,t)$ since $\nabla^2 T(x,t) = 0$, that means the temperature increases in that point before decreasing, but if we consider a infinite heat propagation speed then this overshoot is infinitely small and it tends to zero.

4.4.2 Thermal system physical properties

In the following we mention two properties of thermal systems, useful for successive proofs. The first is the Maximum Principle that allows one to infer the position of the maximum temperature in a generic open volume V.

Maximum Principle (22) (23): Assume the parabolic cylinder and the parabolic boundary respectively defined as:

$$\Omega_\tau := V \times (0,\tau] \,, \quad \Gamma_\tau := \bar{\Omega}_\tau - \Omega_\tau = \bar{V} \times [0,\tau] - \Omega_\tau$$

where the bar indicate the closure set, and suppose $T(x,t)$ sufficiently smooth ($T(x,t) \in C^2(\Omega_\tau) \cap C(\bar{\Omega}_\tau)$) and $T(x,t)$ solves the heat equation in Ω_τ. Then,

1. (*weak maximum principle*)

$$\max_{\bar{\Omega}_\tau} T(x,t) = \max_{\Gamma_\tau} T(x,t) \;\; i.e. \;\; \Gamma_T \cap [arg \max_{(x,t)\in\bar{\Omega}_\tau} T(x,t)] \neq \emptyset$$

2. (*strong maximum principle*) if V is connected and there exists a point $(x_m,t_m) \in \Omega_\tau$ such that,

$$T(x_m,t_m) = \max_{\bar{\Omega}_\tau} T(x,t) \;\; i.e. \;\; \Omega_\tau \cap [arg \max_{(x,t)\in\bar{\Omega}_\tau} T(x,t)] \neq \emptyset$$

then $T(x,t)$ is constant in $\bar{\Omega}_\tau$.

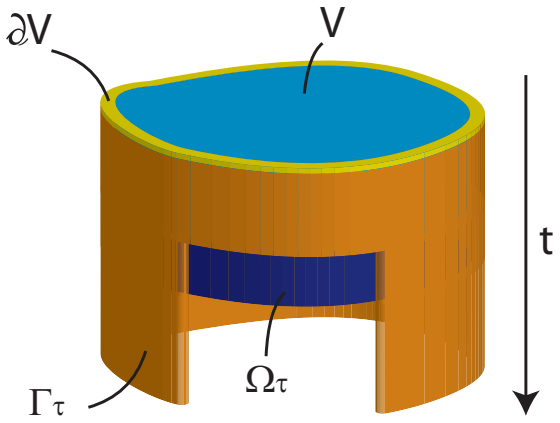


Figure 4.16: Parabolic cylinder for the 2D volume V

In other words the maximum temperature is on the top ($t = 0$) or on boundaries of the parabolic cylinder.

The second properties states that:

Proposition 4.4.1. *The temperature norm rate is the sum of three terms representing respectively, the internal fluttering, the effect of the boundaries and the effect of internal sources.*

Proof. The norm and its derivative can be written as:

$$\|T(x,t)\|_{n,V} \triangleq \sqrt[n]{\int_V |T(x,t)|^n \, dv}$$

$$\frac{d}{dt}\|T\|_{n,V} = \frac{1}{n} \cdot \left(\int_V |T|^n \, dv \right)^{\frac{1}{n}-1} \cdot \frac{d}{dt} \int_V |T|^n \, dv. \tag{4.33}$$

The derivative term on the right hand side of (4.33) can be written as:

$$\frac{d}{dt} \int_V |T|^n \, dv = \int_V n \cdot T \cdot |T|^{n-2} \frac{\partial T}{\partial t} \, dv. \tag{4.34}$$

Then, substituting the heat equation $\partial T/\partial t = \alpha \nabla^2 T + q$ in (4.34) we obtain:

$$\frac{d}{dt} \int_V |T|^n \, dv = \int_V nT|T|^{n-2} \alpha \nabla^2 T \, dv + \int_V nT|T|^{n-2} q \, dv. \tag{4.35}$$

Recalling Green identity:

$$\int_V f \cdot \nabla^2 g \, dv = \int_{\partial V} f \cdot (\nabla g \cdot \hat{n}) \, ds - \int_V \nabla f \cdot \nabla g \, dv$$

we rewrite the first term of (4.35) on the right hand side as:

$$\begin{aligned}\int_V nT|T|^{n-2}\alpha\nabla^2 T\, dv = \alpha\int_{\partial V} nT|T|^{n-2}(\nabla T\cdot\hat{n})\, ds + \\ -\alpha\int_V n(n-1)|T|^{n-2}\|\nabla T\|_2^2\, dv\end{aligned} \tag{4.36}$$

Putting together (4.35) and (4.36), (4.33) becomes:

$$\begin{aligned}\frac{d}{dt}\|T\|_{n,V} = \gamma\cdot\Big[\underbrace{\alpha\cdot\int_{\partial V} nT|T|^{n-2}(\nabla T\cdot\hat{n})\, ds}_{\text{EFFECT OF THE BOUNDARIES}} + \\ \underbrace{-\alpha\int_V n(n-1)|T|^{n-2}\|\nabla T\|_2^2\, dv}_{\text{INTERNAL FLUTTERING}} + \underbrace{\int_V nT|T|^{n-2}q\, dv}_{\text{INTERNAL SOURCES}}\Big]\end{aligned} \tag{4.37}$$

□

4.4.3 The constraint reduction property

The central target of the predictive controller is limiting the temperature of the volume V under a specified threshold. This goal translates into having an optimization problem with an infinite number of constraints, one for each infinitesimal volume element. Starting from the Maximum Principle, in this section we have shown how to reduce these constraints to a finite number, making the controller implementable and reducing its complexity[1].

Proposition 4.4.2. *Consider the problem (4.31) with $q \neq 0$ and DBCs or NBCs. The maximum temperature is either at initial time, or on the boundary or on sources.*

Proof. Since (4.31) is a Cauchy problem, it admits a unique solution $\bar{T}(x,t)$. Assuming $q \neq 0$ only in a finite sum of compact and connected sets $V_{S,i}$ with $i = 1,2,\ldots,n$, it is possible to find the temperature $\bar{T}(x,t)$ on the boundary of each source (i.e. $\forall x \in \cup_i \partial V_{S,i}$). Calling $\mathcal{V}_S = \cup_{i=1}^n V_{S,i}$ and $\partial\mathcal{V}_S = \cup_{i=1}^n \partial V_{S,i}$, the Cauchy problem can be rewritten as an equivalent problem without any internal heat sources:

$$\begin{cases}\frac{\partial T}{\partial t} - \alpha\cdot\nabla^2\cdot T = 0 & x\in V\setminus\mathcal{V}_S\ ,\ t\in[0,\tau[\\ T(x,0) = T_0(x) & \forall x\in V\setminus\mathcal{V}_S \\ T(x,t) = T_{\partial V}(x,t)\ \ or\ \ \hat{n}\cdot\nabla T = J_B & \forall x\in\partial V,\ \forall t\in[0,\tau[\\ T(x,t) = \bar{T}(x,t)\ \ or\ \ \hat{n}\cdot\nabla T = \hat{n}\cdot\nabla\bar{T} & \forall x\in\partial\mathcal{V}_S, \forall t\in[0,\tau[\end{cases} \tag{4.38}$$

[1] The number of constraints as well as the model order determine the complexity of the optimization problem.

Applying the maximum principle the maximum temperature is at $t = 0$ or on the augmented boundary $(\partial V \cup \partial \mathcal{V}_S)$ that is:

$$\max_{(x,t)\in V\setminus\mathcal{V}_S\times[0,\tau]} \bar{T}(x,t) = \max_{(x,t)\in\{\{V\setminus\mathcal{V}_S\}\times\{0\}\}\cup\{\{\partial V\cup\partial\mathcal{V}_S\}\times[0,\tau]\}} \bar{T}(x,t)$$

hence:

$$\max_{(x,t)\in\Omega_\tau} \bar{T}(x,t) = \max_{(x,t)\in\{V\times\{0\}\cup\partial V\times[0,\tau]\cup\mathcal{V}_S\times[0,\tau]\}} \bar{T}(x,t)$$

□

The above result directly leads to the following proposition.

Proposition 4.4.3. *Under the conditions that the initial and boundary temperatures are lower than T_{CRIT}, a necessary and sufficient condition for $T(x,t)\leq T_{CRIT}$ $\forall x\in V$, $\forall t\geq 0$ is that the maximum temperature on sources is always lower than T_{CRIT}.*

Fig. 4.17 illustrates the temperature distribution of a volume, in two subsequent time instants. In Fig. 4.17a we find the maximum temperatures on the two sources, in which we applied a 20W power. Then, after removing powers (see Fig. 4.17b), the maximum temperature moves to a middle point, not corresponding to any sources, but with a magnitude far lower than the initial one shown in Fig. 4.17a.

The main consequence of latter propositions is that possible constraints violation may happen on sources first, if we assume that the boundary temperature and the initial condition meet constraints. According to this result, and with the further assumption that the number of sources is finite and that each of them is assimilable to a point source, then we can convert the infinite dimensional constraint into the finite dimensional constraint $T(x,t) \leq T_{CRIT}, \forall x \in \mathcal{V}_S$.

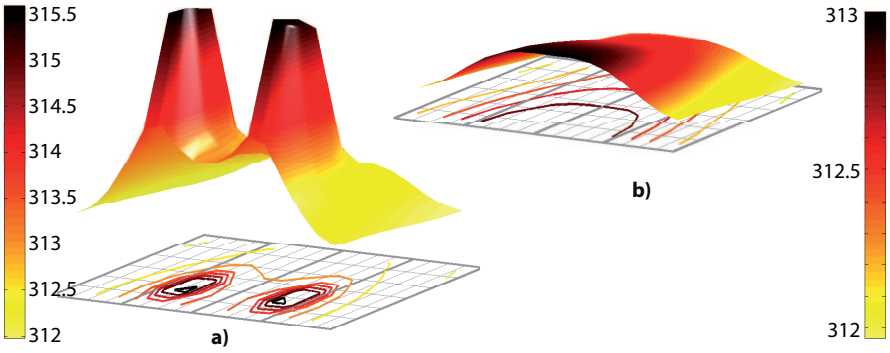


Figure 4.17: Two sources simulation: a) 20W per sources; b) 0W per sources

4.4.4 The feasibility issue

One of the most important property for MPC schemes is feasibility: if a control input sequence meeting the constraints exists at time $t = 0$, then, it will exist also for all $t > 0$.

In classical MPC literature such a property is enforced by using adequate prediction horizons, terminal weighting matrix, invariant terminal sets, etc. (25) (2). However, these are indirect methods that have some limitations of use, they complicate the design of the controller and augment its computational complexity (26) (24). In this Section our aim is to prove that predictive controllers related to the centralized problem (4.26) and the distributed one (4.21), possibly with some adaptations, are intrinsically feasible when applied to whatever thermal system, for any horizon of length greater or equal to 0. The first step is captured by the following proposition, direct consequence of the Maximum Principle,

Proposition 4.4.4. *Consider the system (4.31) under DBCs with no internal sources. If $T_0(x)$ and $T(x,t)$ are lower than T_{CRIT} respectively at time $t = 0$ and on the boundary $\forall t \in [0,\tau]$, then the temperature will never exceed T_{CRIT}, i.e. the set $\mathcal{T} = \{T(\cdot,\cdot) | T(x,\cdot) \leq T_{CRIT}\ ,\ \forall x \in V\}$ is a positive invariant set.*

Proof. According to the maximum principle, the temperature of any points in the parabolic cylinder is lower than the maximum temperature at initial time or on the cylinder boundary. Consequently, $\forall t_1, t_2$ such that $t_2 \geq t_1 \geq 0$:

$$\|T(x,t_2)\|_{\infty,V} \leq \max(\|T(x,t_1)\|_{\infty,V}, \|T(\cdot,\cdot)\|_{\infty,\,\partial V,\, t\in[t_1,t_2]})$$

that proves $\mathcal{T}$ is invariant. □

Thus, if the maximum temperature meets the constraints at time $t = 0$, the temperature on boundary is lower than T_{CRIT}, and no sources are present, then the constraints will never be violated in the future. This has a direct consequence in the MPC context; it means that if constraints are not violated at a given time, then the null input action (i.e. zeroing all of the sources) always ensures constraints satisfaction in the future. This result clearly guarantees the feasibility for the centralized control problem (4.26) for any prediction horizon.

A similar result may be stated also under *NBCs* when there is no heat transfer from the external environment towards the thermal system. In such a case it is possible to prove that,

Proposition 4.4.5. *Consider the problem (4.31) with NBCs, no internal sources ($q = 0$) and no heat coming from the boundaries, i.e. $-J_{\partial V} = \hat{n} \cdot \nabla T \leq 0$. Then $\|T(x,t)\|_{\infty,V}$ decreases along time.*

Proof. With $q = 0$, the equation (4.37) is equal to:

$$\begin{aligned}\frac{d}{dt}\|T\|_{n,V} = \quad & \gamma \cdot \left[-\alpha \int_V n(n-1)\|T\|^{n-2}\|\nabla T\|_2^2 \, dv + \right. \\ & \left. + \alpha \cdot \int_{\partial V} nT\|T\|^{n-2}(\nabla T \cdot \hat{n}) \, ds \right] \leq 0\end{aligned} \tag{4.39}$$

where the right-hand term is negative since the internal fluttering is always negative, the temperature is always positive and the heat flow at the boundary is not incoming. Since $T(x,t)$ is continuous over V, then:

$$\|T(x,t)\|_{\infty,V} = \lim_{n\to\infty} \|T(x,t)\|_{n,V}$$

Hence,

$$\frac{d}{dt} \lim_{n\to\infty} \|T(x,t)\|_{n,V} = \lim_{n\to\infty} \frac{d}{dt}\|T(x,t)\|_{n,V} \leq 0$$

□

Thus, if we provide a null input action to the system the temperature will never increase, proving recursive feasibility in the centralized case. Notice that, on the basis of the above considerations, the infinity norm may be seen as a Lyapunov function for the considered class of thermal systems. It is also worth to note that in the next proofs we will assume DBCs for simplicity reasons and without any accuracy loss since it is always possible to convert one boundary condition to another.

Remark 2. *A physical interpretation of the described property may be given according to the fact that, for the Second Principle of Thermodynamics, heat flow cannot be directed as the temperature gradient. Hence, no inductive storage elements can be present in thermal systems and, consequently, no free resonant or double integrative behaviors can arise when no sources are present.*

For what concerns the feasibility of distributed solutions, related to the problems (4.21), a crucial point is to define the "perimeter" of the local controllers and the information about the rest of the system available to each of them. According to Subsection 4.4.3, a finite number of point-wise sources, $x_{s,i}$ $i = 1 \ldots N_s$, is considered in V (i.e. $x_{s,i}$ are the only points in V where $q(x,t)$ can be larger than zero) and $\mathcal{V}_s$ is defined as the set collecting all of them. By Proposition 4.4.2, the requirement to prevent temperatures larger than T_{CRIT} all over the system can be achieved by preventing overtemperature on the sources (if suitable initial and boundary conditions are present). Then, following the formulation of (4.21), a local controller is assumed for each source $x_{s,i}$; each controller can read its own source temperature and can act on its local

power, while it has to comply with the local constraint $T(x_{s,i}) \leq T_{CRIT}$. At this stage, no information on the rest of the system is considered available. The only assumption that can be formulated at local control level is that all of the other controllers are acting in order to comply with their local constraints. In this framework, taking the cue from the previous result for the centralized problem (4.26), it looks reasonable to expect that shutting down a single core guaranties no overtemperature can take place for that core, just assuming the other cores have no overtemperature as well, but independently of their exact temperature and dissipated power. In other words, whenever a core is approaching harmful temperature, the local decision of reducing its power should be enough to prevent local constraint violation, and contextual reduction of powers of all other cores should not be needed, providing that each of them is meeting its local thermal bound. Indeed, this property, which will be crucial for "distributed" feasibility, is true and captured in the following Proposition.

Proposition 4.4.6. *Consider the system (4.31) under DBCs with $\mathcal{V}_s = \{x_{s,i}\, i = 1 \dots N_s\}$ the set of points in V where the sources are located. Assume that the initial condition and the boundary temperatures satisfy the constraints, i.e. $T_0(x) \leq T_{CRIT}, \forall x \in V$ and $T_{\partial V}(x,t) \leq T_{CRIT}, \forall x \in \partial V, \forall t \in [0, \tau[$. For each source $x_{s,i}$ $i = 1 \dots N_s$, the local decision of imposing $q(x_{s,i}, t) = 0, \forall t \in [0, \tau[$ guarantees $T(x_{s,i}, t) \leq T_{CRIT}, \forall t \in [0, \tau[$, if $T(x_{s,j}, t) \leq T_{CRIT}$ for all of the other sources $j \in \{1 \dots N_s\}$, $j \neq i$ and $\forall t \in [0, \tau[$.*

Proof. As soon as, in a source point $x_{s,i}$, $q(x_{s,i}, t)$ is zeroed, this point becomes equivalent to a no-source point in V, then the result, claimed in the proposition, follows by Proposition 4.4.2. □

In the above analysis, the possibility of imposing null power on sources is a key element to achieve important properties in the path toward feasibility of both centralized and distributed control problems (4.26) and (4.21). In real chips, it is very hard and possibly harmful to have a sudden zeroing of the power consumption (i.e. halting the core activities immediately). The common procedure is to slow down the clock frequency of the cores to a lower bound f_{MIN}, that means reduce the source powers to a minimal value, referred here as q_{MIN}. Clearly, this issue cannot be covered by the results of Propositions 4.4.4 and 4.4.6, but, taking inspiration from them and introducing a suitable thermal constraints review, similar results with non-null minimum power can be obtained as shown in the following, these results will be finally exploited to show the feasibility of the considered centralized and distributed control problems.

Define $T_{EQ}(x)$ as the solution of the following Cauchy problem

$$\begin{cases} -\alpha \cdot \nabla^2 T(x) = q_{MIN}(x) & x \in V \\ T(x) = T_{\partial V,\, max}(x) & \forall x \in \partial V \end{cases} \tag{4.40}$$

where $q_{MIN}(x)$ represents the minimum power which can be dissipated in any source and $T_{\partial V,\, max}(x)$ is the largest environment temperature which can be experienced by the chip on its boundary. Similarly to Proposition 4.4.6, a finite number of point-wise sources, $x_{s,i} : i = 1 \dots N_s$, is assumed, with $\mathcal{V}_s = \{x_{s,i}\, i = 1 \dots N_s\}$.

According to the above definition, $T_{EQ}(x)$ corresponds to the steady state temperature distribution for the system (4.31) under minimum power consumption and worst ambient temperature (i.e. assuming $T_{EQ}(x)$ as initial condition and $T_{\partial V,\, max}(x)$ as constant boundary condition, the solution of (4.31) is constant along time and equal to $T_{EQ}(x)$). Obviously, $T_{EQ}(x) < T_{CRIT}$, $\forall x \in \bar{V}$ is guaranteed by a proper sizing of the chip, otherwise the thermal constraints cannot be met. In addition, by Proposition 4.4.2, it follows that the maximum value of $T_{EQ}(x)$ is located on sources or boundaries, i.e.

$$\left\{ arg\max_{x \in \bar{V}} T_{EQ}(x) \right\} \cap \{\mathcal{V}_s \cup \partial V\} \neq \emptyset$$

Then, defining

$$\Delta T_{CRIT} = \min_{x \in V} (T_{CRIT} - T_{EQ}(x)), \tag{4.41}$$

it results

$$\Delta T_{CRIT} = \min_{x \in \partial V \cup \mathcal{V}_s} (T_{CRIT} - T_{EQ}(x)) > 0, \tag{4.42}$$

At this point, a crucial review of temperature bound needs to be introduced, we define a space-variant $\bar{T}_{CRIT}$ constraint as follows

$$\bar{T}_{CRIT}(x) = T_{EQ}(x) + \Delta T_{CRIT}, \ \forall x \in \bar{V}. \tag{4.43}$$

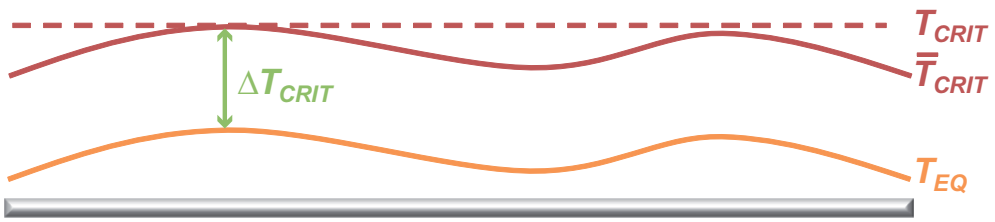


Figure 4.18: Definition of the new bound $\bar{T}_{CRIT}$

In Fig. 4.18, the meaning of this new bound is graphically represented assuming a simple one-dimension thermal system. Clearly, the new bound $\bar{T}_{CRIT}(x)$ is tighter than T_{CRIT}, but this restriction is fundamental to formulate the following propositions, as it will be clear by their proofs.

Proposition 4.4.7. *Consider the thermal system (4.31) under DBCs with $q(x,t) \neq 0$. Assuming $T_0(x) \leq \bar{T}_{CRIT}(x) \forall x \in V$ and $T_{\partial V}(x,t) \leq \bar{T}_{CRIT}(x) \forall x \in \partial V \forall t \in [0,\tau[$, and applying $q(x,t) = q_{MIN}(x)$ the following bound holds $T(x,t) \leq \bar{T}_{CRIT}(x) \forall x \in V \forall t \in [0,\tau[$; i.e. the set $\bar{\mathcal{T}} = \{T(\cdot,\cdot) | T(x,\cdot) \leq \bar{T}_{CRIT}(x)\ , \forall x \in V\}$ is a positive invariant set respect to the constant input $q_{MIN}(x)$.*

Proof. As consequence of the superposition principle the heat equation $\dot{T}(x,t) - \alpha \cdot \nabla^2 T(x,t) = q_{MIN}(x)$ can be rewritten as

$$\Delta\dot{T}(x,t) - \alpha \cdot \nabla^2 \Delta T(x,t) = 0 \tag{4.44}$$

where $\Delta T(x,t) \triangleq T(x,t) - T_{EQ}(x)$ is the displacement from the equilibrium temperature. According to the hypothesis of the Proposition and the definition of $\bar{T}_{CRIT}$, we consider (4.44) under the following initial and boundary conditions, $\Delta T_0(x) \leq \Delta T_{CRIT} \forall x \in V$ and $\Delta T_{\partial V}(x,t) \leq \Delta T_{CRIT} \forall x \in \partial V \forall t \in [0,\tau[$ The Maximum Principle can be applied to such system, since it has the same structure of (4.31), then $\Delta T(x,t) \leq \Delta T_{CRIT} \forall x \in V \forall t \in [0,\tau[$ and, consequently, $T(x,t) \leq T_{EQ}(x) + \Delta T_{CRIT} = \bar{T}_{CRIT}(x) \forall x \in V \forall t \in [0,\tau[$

□

Similarly to Proposition 4.4.4, the above result is useful only to achieve feasibility of the centralized control problem (4.26), but exploiting the approach adopted in Proposition 4.4.6, the following can be derived.

Proposition 4.4.8. *Consider the system (4.31) under DBCs with $x_{s,i}$ $i = 1 \ldots N_s$ the points in V where the sources are located. Assume that the initial condition and the boundary temperatures satisfy the constraints, i.e. $T_0(x) \leq \bar{T}_{CRIT}(x), \forall x \in V$ and $T_{\partial V}(x,t) \leq \bar{T}_{CRIT}(x),\ \forall x \in \partial V,\ \forall t \in [0,\ \tau[$. For each source $x_{s,i}$ $i = 1 \ldots N_s$, the local decision of imposing $q(x_{s,i},t) = q_{MIN}(x_{s,i}),\ \forall t \in [0,\ \tau[$ guarantees $T(x_{s,i},t) \leq \bar{T}_{CRIT}(x_{s,i}),\ \forall t \in [0,\ \tau[$, if $T(x_{s,j},t) \leq T_{CRIT}(x_{s,j})$ for all of the other sources $j \in \{1 \ldots N_s\},\ j \neq i$ and $\forall t \in [0,\ \tau[$.*

Proof. By defining $\Delta T(x,t) \triangleq T(x,t) - T_{EQ}(x)$ and $\Delta q(x,t) \triangleq q(x,t) - q_{MIN}(x)$ (where $\Delta q(\cdot)$, $q(\cdot)$ and $q_{MIN}(\cdot)$ can be non-null in the point-wise sources $x_{s,i}$ $i = 1 \ldots N_s$), the temperature dynamics can be rewritten as follows, again exploiting the superposition principle

$$\Delta\dot{T}(x,t) - \alpha \cdot \nabla^2 \Delta T(x,t) = \Delta q(x,t) \tag{4.45}$$

This equation has the same structure of (4.31), then it inherits all of its features. In addition, according to (4.43), the bound $\bar{T}_{CRIT}(x)$ is mapped into a constant bound ΔT_{CRIT}. Therefore, as soon as, in a source point $x_{s,i}$, $q(x_{s,i},t) = q_{MIN}(x_{s,i})$ is imposed, it means that $\Delta q(x_{s,i})$ is zeroed and this point becomes equivalent to a no-source point in V for (4.45). Then, according to Proposition 4.4.2, the maximum of temperatures $\Delta T(x,t)$ occurs on the remaining sources $x_{s,j}$, $j \in \{1 \dots N_s\}$, $j \neq i$ and/or the initial condition and/or the boundaries. In these places, by the proposition assumptions, $\Delta T(x,t)$ is always lower than ΔT_{CRIT}, therefore the claimed result follows. □

With the above results at hand, the feasibility properties of the centralized and distributed problems, (4.26) and (4.21) can be formally stated as follows, when $q_{MIN}(x)$ is the minimum power which can be dissipated in any point-wise source $x_{s,i} i = 1 \dots N_s$.

Proposition 4.4.9. Centralized Feasibility. *Consider the thermal system (4.31) under DBCs and subject to the constraint $T(x,t) \leq \bar{T}_{CRIT}(x), \forall x \in V, \forall t \geq 0$, according to the control problem (4.26). Assume that $x_{s,i}$ $i = 1 \dots N_s$ are the points in V where the sources are located and define $u(t) \in \mathbb{R}^{+N_s}$ as the centralized control vector, whose i-th element commands the power $q(x_{s,i})$. Let $T_0(x) \leq \bar{T}_{CRIT}(x)$ at time $t = 0$ and $T(x,t) \leq \bar{T}_{CRIT}(x)$ on the boundary $\forall t \in [0,\tau]$. Assume that at time t' a control strategy is computed such that constraints are satisfied in the interval $t' \leq t \leq t' + \Delta t$ where $\Delta t > 0$ is a the prediction horizon. If such a strategy is applied to the thermal system, then for any time $t'' \in [t', t' + \Delta t]$ it is possible to compute a new control strategy that if applied will not violate constraints in the interval $t'' \leq t \leq t'' + \Delta t$.*

Proof. It is enough to note that by using the strategy computed at time t' up to the time $t' + \Delta t$ the constraints at time t'' will not be violated, i.e. $T(x,t'') \leq \bar{T}_{CRIT}(x), \forall x \in V$. Then, the strategy $u_i(t) = q_{MIN}$, $i = 1 \dots N_s, \forall t \in [t'', t'' + \Delta t]$ is always a feasible strategy, since $\bar{\mathcal{T}}$ is a positive invariant set according to Proposition 4.4.7. □

Proposition 4.4.10. Distributed Feasibility. *Consider the thermal system (4.31) under DBCs. According to the control problem (4.21), assume that $x_{s,i}$ $i = 1 \dots N_s$ are the points in V where the sources are located and define, for each i, $u_i(t) \in \mathbb{R}^+$ as the scalar control input, available to the local i-th controller, whose value commands the power $q(x_{s,i},t)$. Let $T_0(x) \leq \bar{T}_{CRIT}(x)$ at time $t = 0$ and $T(x,t) \leq \bar{T}_{CRIT}(x)$ on the boundary $\forall t \in [0,\tau]$. Assume the system (4.31) is subject to the following constraints, again according to (4.21), $T(x_{s,i},t) \leq \bar{T}_{CRIT}(x_{s,i}), \forall t \geq 0, i = 1 \dots N_s$ (these constraints are equivalent to require $T(x,t) \leq \bar{T}_{CRIT}(x), \forall x \in V, \forall t \geq 0$, as stated in Proposition 4.4.2). Finally, assume that at time t' for a generic local controller, $\bar{i}$, a control strategy is computed such that the local constraint $T(x_{s,\bar{i}},t) \leq \bar{T}_{CRIT}(x_{s,\bar{i}})$ is satisfied in the interval $t' \leq t \leq t' + \Delta t$ (where $\Delta t > 0$ is again the prediction horizon), provided*

that all the other local controllers $j \neq \bar{i},\ j \in \{1 \dots N_s\}$ satisfy their own local constraint $T(x_{s,j},t) \leq \bar{T}_{CRIT}(x_{s,j}) \forall t \in [t', t'+\Delta t]$. If such a strategy is applied to the local source $\bar{i}$ and all other constraints are fulfilled in the considered interval, then for any time $t'' \in [t', t'+\Delta t]$ it is possible to compute a new local control strategy for the source $\bar{i}$ that, if applied, will provide the following features; (i) the local constraint will not be violated in the interval $[t'' \leq t \leq t''+\Delta t]$, provided that the other constraints are not violated as well; (ii) the local control strategy applied to $\bar{i}$, or to any subset of sources, will not prevent the other sources from the possibility of meeting their own local constraint exploiting the same strategy. In addition, the above-mentioned local control strategy for $\bar{i}$ can be selected without any detailed information on the conditions of the other controllers, but just exploiting the assumption that all of them are satisfying the other constraints.

Proof. First of all, according to the above assumptions, the constraints at time t'' will not be violated, i.e. $T(x_{s,i},t'') \leq \bar{T}_{CRIT}(x_{s,i})$, $i = 1 \dots N_s$ and, then, $T(x,t'') \leq \bar{T}_{CRIT}(x), \forall x \in V$. Therefore, (i) follows as direct consequence of Proposition 4.4.8 by applying, in the generic local controller $\bar{i}$, $u(t)_{\bar{i}} = q_{MIN}, \forall t \in [t'' \leq t \leq t''+\Delta t]$. Clearly, this control strategy does not require any knowledge on the other local controllers, confirming what stated at the end of proposition. For what concerns (ii), by Proposition 4.4.7 it follows that all of the local controllers can apply $u(t)_i = q_{MIN}, \forall t \in [t'' \leq t \leq t''+\Delta t]$, guaranteeing feasibility. Moreover, by exploiting the approach adopted in the proof of Proposition 4.4.8 it is easy to prove what follows. Considering a generic $j \in [1 \dots N_s]$ and defining two generic disjointed subsets of sources, S_0 and S_s, not including j, but covering all of the sources (i.e. $S_0 = \{n \,|\, n \in [1 \dots N_s], n \neq j\}$, $S_s = \{m \,|\, m \in [1 \dots N_s], m \neq j\}$, $S_0 \cap S_s = \emptyset$, $S_0 \cup S_s = [1 \dots j-1, j+1 \dots N_s]$), if all of the sources in S_0 apply $u_n(t) = q_{MIN}, \forall t \in [t'', t''+\Delta t]$ and all of the sources in S_s meet their own local constraints $T(x_{s,m},t) \leq \bar{T}_{CRIT}(x_{s,m}), \forall t \in [t'', t''+\Delta t]$, then in the source j it can be applied $u_j(t) = q_{MIN}, \forall t \in [t'', t''+\Delta t]$, guaranteeing $T(x_{s,j},t) \leq \bar{T}_{CRIT}(x_{s,j}), \forall t \in [t'', t''+\Delta t]$. Then, also (ii) is proven. □

A remarkable point that can be noted is that the above results hold true for any positive prediction horizon $\Delta t > 0$; therefore, it can be selected arbitrarily small without impairing feasibility properties. It means that there is room to preserve feasibility until the current temperature reach the bound, both in the centralized and distributed approach.

Another relevant point to highlight is the use of $\bar{T}_{CRIT}(x) \leq T_{CRIT}$ instead of T_{CRIT}. This is not just a technical point related to the available theoretical tools, but it accounts for a possible practical trouble in the thermal behavior (even if it can be a bit conservative). As a matter of facts, whenever a core (or any heat source) needs an indirect path through its neighbors to expel

its heat flow and keep a safe temperature (i.e. it doesn't have a direct path toward the external ambient that is efficient enough), the neighbors are required to keep a temperature lower than the maximum to give room to the heat flow coming from the above-mentioned core. Clearly, this characteristic strongly depends on the layout and the thermal resistivity of the considered system and should be carefully addressed in the multi-core design. By the way, an idea on how important is this effect can be easily derived by the analysis of $T_{EQ}(x)$, solution of (4.40); the more uniform is $T_{EQ}(x)$ in V, the less important is "indirect heat flow" to guarantee safe temperature.

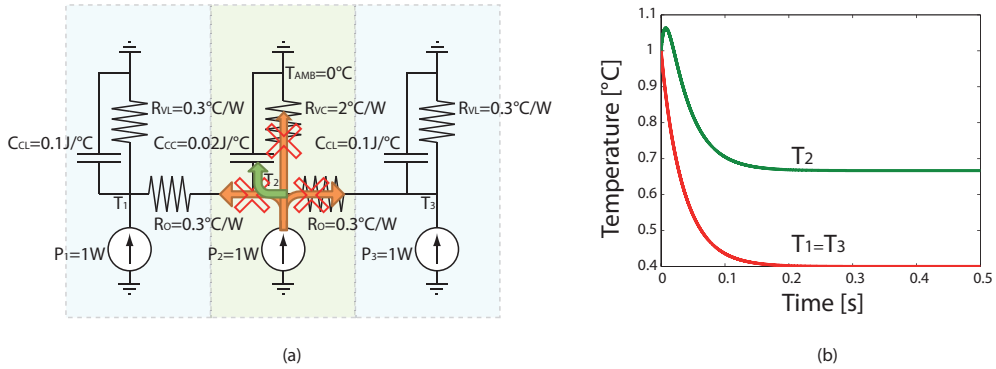


Figure 4.19: (a) simple circuit used for simulating a volume with three point-wise sources; (b) simulation result using a uniform T_{CRIT}.

Fig. 4.19 shows a simple example to clarify the concepts. We consider a volume with three three point-wise sources. We split the volume into three cells associating to each one an equivalent electric circuit (as shown in figure). The power consumption of the sources are $P(x_{s,1})$, $P(x_{s,2})$, and $P(x_{s,3})$, the ambient temperature is $T_{AMB} = 0°C$, the critical temperature is $T_{CRIT} = 1°C$, the minimum power $P_{MIN} = 1W$, and the initial temperature of the cells are $T(x_{s,1}) = T(x_{s,2}) = T(x_{s,3}) = 1°C$. When the three sources dissipate P_{MIN} the lateral cells dissipate all the power whereas the central one dissipates only a small part of the power (due to the resistance value). The source 2 is neither able to dissipate the power through the cells 1 and 3 since the initial temperatures are the same. Therefore, the power goes to increase the temperature causing a violation of T_{CRIT}. This happens because the T_{CRIT} is uniform. When the minimum power is applied the corresponding equilibrium temperature is such that $T_{EQ}(x_{s,1}) = T_{EQ}(x_{s,3}) = 0.4°C$, $T_{EQ}(x_{s,2}) = 0.68°C$. Due to the resistance and capacitance values assigned, we can note a large difference between these temperatures, then a large importance of "indirect heat flow" could be present. Indeed, this is confirmed by simulations. Assuming that $T_{CRIT} =$

$1^{\circ}C$, imposing an initial condition with all the temperatures in the volume V equal to T_{CRIT} and applying q_{MIN} in all of the sources, we can observe a constraint violation in source 2. In contrast, if we start from an initial condition $T(x,0) = \bar{T}_{CRIT}(x)$, no over-temperature takes place.

The above results guarantee feasibility by applying minimum power consumption at each core. The centralized feasibility of Proposition 4.4.9 is essentially based on the observation that, by switching to the minimum power all of the cores, the maximum temperature along the chip will never increase, since no resonant or double integration effects can arise. The distributed feasibility of Proposition 4.4.10 allows to recover this property locally to each source without needing any information exchange, but just assuming, in a local controller, that all of the other are working correctly. This can be achieved at the cost of replacing the uniform bound T_{CRIT} with the variable bound $\bar{T}_{CRIT}(x)$ (probably, this cost can be reduced by allowing proper information exchange between the core and selecting a large enough prediction horizon, but this is not the subject of this work).

At first glance, these results could seem of little interest from a practical viewpoint, since they just provide the trivial solution of switching to minimum power the cores to meet the temperature constraints, but they do not provide any algorithm to find other feasible solutions which maximize the computing power, as reported both in (4.26) and (4.21). To this purpose, in the following, algorithms based on approximated discretized models will be presented, also exploiting information exchange between controllers for the distributed case. Nevertheless, the above feasibility results play a fundamental role for practical applications. Indeed, by using algorithm based on approximated discrete models, temperature constraints violation could be experienced, but this can be effectively prevented, according to the above feasibility results with arbitrary small prediction horizon, by adding a sort of *ultimate temperature capping control layer* on each local i-th source. This control layer, as soon as the $T_{x_{s,i},t}$ approaches $\bar{T}_{CRIT}(x_{s,i})$, has to override the optimizing controller and to impose simply $q(x_{s,i},t) = q_{MIN}$ for a suitable time interval to obtain $T_{x_{s,i},t}$ sufficiently lower tha $\bar{T}_{CRIT}(x_{s,i})$.

4.4.5 Discretization issues

We have already shown the benefit of using a discrete LTI model, as (4.11), instead of the PDE-based one. However, whereas we proved control feasibility for this latter, the same property could not be guaranteed for the identified model due to the time-space discretization process.

In (24) the author showed a simple approach for testing the control feasibility with respect to a discrete model. It consists in designing a bilevel optimization problem - an "outer" optimization problem containing an "inner" optimization problem in the constraints - and checking the optimal value of the outer objective function. Thus, we could discard a model identified with the iterative procedure if the test fails.

We have already shown the main step of this approach in Chapter 3. Hereafter we specialized the approach to our problem. The discrete model used is given by:

$$\begin{aligned} x(t+1) &= A \cdot x(t) + B_1 \cdot p(t) + B_2 \cdot d(t) \\ T(t) &= C \cdot x(t) \end{aligned} \tag{4.46}$$

where we discriminated control inputs, p, from the others, d.

The optimization problem (4.21) can be rearranged in a quadratic programming form,

$$min \ \frac{1}{2} \cdot P(t) \cdot H \cdot P(t) + P(t) \cdot G(t) \tag{4.47a}$$

$$s.t.$$

$$E \cdot x(t) + F \cdot P(t) \leq b(t) \tag{4.47b}$$

where $P(t) = [p(t|t) \ p(t+1|t) \ \ldots \ p(t+N-1|t)]$, $E = C \cdot A$, $F = C \cdot B_1$ and $b(t) = T_{CRIT} - C \cdot B_2 \cdot d(t)$.

Feasibility is guaranteed if for every initially feasible states and for every feasible control sequences the optimization problem remains feasible for all time. Hence, if $x(t)$ is a feasible initial state and $p^{\circ}(t|t)$ is the optimal solution of (4.47), then at the next sample interval,

$$E \cdot (A \cdot x(t) + [B_1 \ B_2] \cdot [p^{\circ}(t|t) \ \ d(t)]^T) + F \cdot P(t+1) \leq b(t) \tag{4.48}$$

According to the Farkas' Lemma, a $x(t)$ exists such that (4.48) is true or a $y(t)$ exists such that $y \geq 0$, $y^T \cdot F = 0$ and $y^T \cdot (b(t) - E \cdot (A \cdot x(t) + [B_1 \ B_2] \cdot [p^{\circ}(t|t) \ d(t)]^T) < 0$.

Thus, starting from a feasible state $x(t)$ and giving an optimal input $p^{\circ}(t|t)$, if a y satisfying these conditions exists, then the control problem is infeasible. According to (24) we designed the following optimization problem to check the consistency of Farkas' conditions.

$$\min_{y,\,x(t),\,d(t)} \; y^T \cdot (b(t) - E \cdot (A \cdot x(t) + [B_1\ B_2] \cdot [p^{\circ}(t|t)\ \ d(t)]^T)) \tag{4.49a}$$

$$s.t.$$

$$y \geq 0\ ,\ \ y^T \cdot F = 0 \tag{4.49b}$$

$$x \in \mathbb{X},\ d \in \mathbb{D} \tag{4.49c}$$

$$P^{\circ} = arg \min_{P(t)} \frac{1}{2} \cdot P(t) \cdot H \cdot P(t) + P(t) \cdot G(t) \tag{4.49d}$$

$$s.t.$$

$$E \cdot x(t) + F \cdot P(t) \leq b(t) \tag{4.49e}$$

$$p \in \mathbb{P} \tag{4.49f}$$

where $\mathbb{X}$, $\mathbb{P}$ and $\mathbb{D}$ are respectively the sets of allowed values for $x(t)$, $p(t)$, $d(t)$, and the cost function of the outer problem is the third condition of the lemma.

Substituting the Karush-Kuhn-Tucker optimality conditions to the inner convex problem (4.49d)-(4.49f) we obtain a easier problem to solve with numerical algorithm:

$$\min_{y,\,x(t),\,d(t),\lambda,P(t)} y^T \cdot (b(t) - E \cdot (A \cdot x(t) + [B_1\ B_2] \cdot [p^{\circ}(t|t)\ \ d(t)]^T)) \tag{4.50a}$$

$$s.t.$$

$$y \geq 0\ ,\ \ y^T \cdot F = 0 \tag{4.50b}$$

$$x \in \mathbb{X},\ d \in \mathbb{D},\ p \in \mathbb{P} \tag{4.50c}$$

$$H \cdot P(t) + G(t) + C \cdot B_1 \cdot \lambda = 0 \tag{4.50d}$$

$$\lambda \geq 0 \tag{4.50e}$$

$$b(t) - E \cdot x(t) - F \cdot P(t) \geq 0 \tag{4.50f}$$

$$\lambda^T \cdot (b(t) - E \cdot x(t) - F \cdot P(t)) = 0 \tag{4.50g}$$

where λ is the array of the Lagrangian multipliers. Farkas' conditions holds if the optimal value of the objective function is negative. In this case at least one state initially feasible can become infeasible. The identification procedure must discard the considered model and recheck the feasibility for a new model solving a new problem. Every time the model change the test must be repeated.

It is worth to note that this approach can be applied both to the centralized and the distributed MPC algorithm. In the first case the state is $x(t) = [T_i(t)\ x_{2,i}]$, $i = 1,\ldots,N$, where T_i is the temperatures of the i-th core and $x_{2,i}$ is the second unknown state of the i-th core. The input $p(t)$ is the vector containing the P_i power of all cores, and $d(t)$ contains only the ambient temperature. In this case $\mathbb{X}$, $\mathbb{P}$ and $\mathbb{D}$ can be assumed equal to $0 \leq T_i(t) \leq T_{CRIT}$, $i = 1,\ldots,N$, $P_{MIN} \leq P_i(t) \leq P_{MAX}$, $i = 1,\ldots,N$, and $0 \leq T_{AMB}(t) \leq T_{CRIT}$.
In the case of a distributed scheme we apply the test to the i-th local controller. The state is $x_i(t) = [T_i(t)\ x_{2,i}]$, the input $p(t)$ is the power consumption of the i-th core (P_i), and $d(t)$ contains the ambient temperature T_{AMB} and the temperature of the neighbors. Then, $\mathbb{X}$, $\mathbb{P}$ and $\mathbb{D}$ can be assumed equal to $0 \leq T_i(t) \leq T_{CRIT}$, $P_{MIN} \leq P_i(t) \leq P_{MAX}$, $0 \leq T_{AMB}(t) \leq T_{CRIT}$, and $0 \leq T_{NEIGH_i}(t) \leq T_{CRIT}$.

In both cases the control solution may be feasible or not. Thus, we should repeat the feasibility test until we identify a feasible model. It is worth to remark that the model obtained by the identification approaches could have not the same physical properties of the real system. However, the distributed MPC introduces some challenges which are not considered by the previously mentioned feasibility test and which cannot occur with a centralized solution. Indeed, the distributed solution could be feasible according to the simplified and discrete-time discrete-space models obtained by identification, but it could be unfeasible for the actual system, owing to the unavoidable model approximations and the time discretization. In order to better understand this concept, consider a distributed MPC solution. The feasibility test is applied to each MPC local controller that uses as prediction model the discrete-time model of a single core. The controller results feasible if all the future states, obtained giving as inputs to the model all possible feasible inputs and starting from any feasible initial states, meet the constraints. However, this approach is based on the assumption that the temperatures of the neighbors are constant during a sampling interval, whereas they change in the real system. Thus, imagine a scenario where all cores have a temperature close, but not equal, to the temperature limit $\bar{T}_{CRIT}$. Each controller computes the optimal decision to bring the temperature of the controlled core close to $\bar{T}_{CRIT}$ without considering that in the meanwhile the temperature of the neighbors cores increase as well. As a result some cores could experience a threshold violation even though the controller had passed the feasibility test. Fig. 4.20 illustrates well this behavior that depends on the use of a discrete-time model instead of continuous-time one. In the centralized case the previously mentioned situation cannot happen because the controller can exploit the information of all the cores and the discrete model of the whole chip to forecast

the future cores temperatures and obtain the optimal control action to make the temperature perfectly equal to $\overline{T}_{CRIT}$.

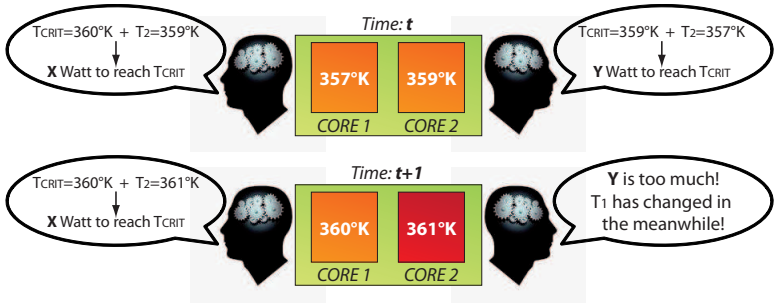


Figure 4.20: Feasibility problem for distributed MPC

Similar issues derive also from the approximation and uncertainties of the model respect to the real system.

In Chapter 5 we proposed a solution able to ensure the feasibility of the system, without the need of a discrete model, avoiding in this way the aforementioned issues.

4.4.6 Notes on stability

The goal of our controller is to regulate the power consumption of the cores to maintain the temperature below a specific threshold. Whereas the feasibility is fundamental to ensure the desired behavior, the stability is unnecessary. By definition a system is stable if it can be bounded arbitrarily close to an equilibrium point. However, in our specific case we are not interested in maintaining the temperature close to a specific value, our priority is constraining the temperature below a threshold. Thus, only the boundedness of the system state must be guaranteed.

The feasibility property is enough for proving the boundedness of the state (note that the feasibility never imply the stability). The proofs stated in Section 4.4.4 ensure the feasibility for a time continuous controller when a partial differential equation based model is considered. When a discrete-time discrete-space model is considered, instead, the controller feasibility can be proved with the feasibility test of Section 4.4.5 (we are not considering the feasibility respect to the real system which is obtained in the Chapter 5 using a hierarchical control structure). The test ensures that the measured part of the state is bounded when limited inputs are applied (the part corresponding to the temperature). However, we have no guaranties that the unmeasurable

part of the state is bounded. This could compromise the feedback behavior of the model. The internal boundedness of the model is ensured by the following proposition,

Proposition 4.4.11. *Let the model*

$$\begin{aligned} x(t+1) &= A_{n\times n}x(t) + B_{n\times m}u(t) \\ y(t) &= C_{p\times n}x(t) \end{aligned} \tag{4.51}$$

be observable and let part of the state be constrained to a bounded value when bounded inputs are applied. Then, the whole state is bounded.

Proof. Because of the observability of the system, the state is uniquely determined after n steps, where n is the order of the model. The state can be found as,

$$x(t) = O_n^{-1} \cdot (Y(t) - U(t))$$

where O_n is the observability matrix $[C;\ CA;\ \ldots; CA^{n-1}]$, $Y(t) = [y(t);\ldots;y(t+n-1)]$ is the output vector containing the outputs over n steps and $U(t) = [CBu(t);\ldots;CA^{t+n-1}Bu(t) + \cdots + CBu(t+n-2)]$. From assumptions, $Y(t) - U(t)$ is bounded (as an example using the 2 norm), therefore,

$$\|x(t)\|_2 = \|O_n^{-1}\| \cdot \|Y(t) - U(t)\|_2$$

where $\|O_n^{-1}\|$ is the induced norm of the observability matrix. □

This proof guarantees that if the controller satisfies the feasibility test, then the inputs needed to constraint the outputs are bounded, and hence, the identified model cannot have unstable dynamics (possibly related to unstable zeros which could occur in the simplified model coming from the identification methods described in Section 4.1).

Bibliography

[1] E.F. Camacho, C. Bordons, *Model predictive control*, Springer, 1999. 82

[2] J.B. Rawlings, D.Q. Mayne, *Model Predictive Control Theory and Design*, Nob Hill Publishing, 2009. 82, 117

[3] N. Sakran et al. *The implementation of the 65nm dual-core 64b merom processor*, In IEEE International Solid-State Circuits Conference, 2007. 83

[4] Xi Chen, Chi Xu, R. P. Dick, Z. Morley Mao. *Performance and power modeling in a multi-programmed multi-core environment*, DAC '10. ACM, New York, NY, USA, pp. 813-818, 2010. 83

[5] R. Bertran, M. Gonzalez, X. Martorell, N. Navarro, E. Ayguade. *Decomposable and responsive power models for multicore processors using performance counters*, ICS '10. ACM, New York, NY, USA, pp. 147-158, 2010. 83

[6] K. Singh, M. Bhadauria, S.A. McKee. *Real time power estimation and thread scheduling via performance counters*, SIGARCH Comput. Archit. News 37, pp. 46-55. July 2009. 83

[7] H.F. Hamann, A. Weger, J.A. Lacey, Z. Hu, P. Bose, E. Cohen, J. Wakil, *Hotspot-Limited Microprocessors: Direct Temperature and Power Distribution Measurements*, IEEE J. Solid-State Circuits, Vol. 4, pp. 5665, Jan. 2007. 82

[8] W. Huang, K. Skadron, S. Gurumurthi, R.J. Ribando, M.R. Stan. *Differentiating the roles of IR measurement and simulation for power and temperature-aware design*, ISPASS, 2009. 84

[9] L. Ljung, *System Identification - Theory For the User*, 2nd ed, PTR Prentice Hall Upper Saddle River, N.J., 1999. 87

[10] R. Guidorzi, *Multivariable system identification*, Bononia University Press, 2003. 87

[11] J. Howard et al, *A 48-Core IA-32 Message-Passing Processor with DVFS in 45nm CMOS*, ISSCC, pp. 108-109, Feb. 2010. 86, 88

[12] Huang Wei, K. Sankaranarayanan, K. Skadron, R.J. Ribando, M.R. Stan, *Accurate, pre-RTL temperature-aware design using a parameterized, geometric thermal model*, IEEE Transactions on Computers, 57(9):1277-1288, 2008. 86

[13] C. Bienia, S. Kumar, J.P. Singh, K. Li, *The PARSEC Benchmark Suite: Characterization and Architectural Implications*, PACT, 2008. 89, 91, 102

[14] P.Holmes, J.L. Lumley, G. Berkooz, *Turbulence, Coherent Structures, Dynamical System, and Symmetry*, Cambridge University Press, Cambridge, 1996. 91

[15] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, *Numerical Recipes in C (2nd Ed.): The Art of Scientific Computing*, Cambridge University Press, New York, NY, USA. 95

[16] A. Bemporad, M. Morari, V. Dua, E.N. Pistikopoulos, *The explicit linear quadratic regulator for constrained systems*, Automatica, Vol. 38(1):3-20, Jan. 2002. 97, 99, 103

BIBLIOGRAPHY

[17] qpOASES Homepage (2011). http://www.qpOASES.org/ 98

[18] F. Zanini, D. Atienza, L. Benini and G. De Micheli, *Multicore Thermal Management with model predictive control*, ECTTD, Vol. 1, 2009. 99, 101, 103

[19] Y. Wang, K. Ma and X. Wang, *Temperature-Constrained Power Control for Chip Multiprocessors with Online Model Estimation*, in Proc. of the 36th annual international symposium on Computer architecture, Austin, TX, USA, June 2009. 101, 105, 106

[20] P.D. Christofidesa, R. Scattolini, D.M. de la Peña, J. Liu, *Distributed model predictive control: A tutorial review and future research directions*, Computers & Chemical Engineering, Vol. 0, pp. - , June 2012. 102

[21] K. Skadron, T. Abdelzaher, M.R. Stan, *Control-Theoretic Techniques and Thermal-RC Modeling for Accurate and Localized Dynamic Thermal Management*, Technical Report, UMI Order Number: CS-2001-27, University of Virginia. 108

[22] Lawrence C. Evans, *Partial Differential Equations*, American Mathematical Society, Feb. 1998. 110, 113

[23] Walter A. Strauss, *Partial Differential Equations:An Introduction*, Wiley, 1992. 110, 113

[24] J. Löfberg, *Oops! I cannot do it again: Testing for recursive feasibility in MPC*, Automatica, Vol. 48(3):550–555, 2011. 117, 126

[25] A. Bemporad, M. Morari, Robust model predictive control: A survey(1999), In A. Garulli, A. Tesi, A. Vicino (Eds.), *Robustness in identification and control*, Lecture Notes in Control and Information Sciences, Vol. 245, pp. 207226, Berlin: Springer. 117

[26] R. Gondhalekara, J. Imura, K. Kashima, *Controlled invariant feasibility - A general approach to enforcing strong feasibility in MPC applied to move-blocking*, Automatica, Vol. 45(12):2869–2875, 2009. 117

Chapter 5

Complex control solutions

In this chapter three complex control solutions are presented. All these solutions involve as common element the distributed thermal controller presented in the previous chapter. In addition they provide extensions to the nominal functionality enabling energy saving in the first example, feasibility and reliability in the second one, and communication properties between cores in the third and last example of this chapter.

5.1 Thermal and Energy management of High-Performance Multi-cores

The control solution we present in this section addresses the scalability challenge for large multi-core platforms with a fully distributed architecture. It combines energy minimization, MPC based thermal control, and thermal model self-calibration.

5.1.1 The Architecture

The Fig. 5.1 depicts the block diagram of the proposed solution. Each *i*-th core runs three main programs, two of them are executed on-line and one off-line:

Local Self-Calibration Routine : it automatically derives, off-line, the local, but interacting, thermal prediction model adopted in MPC-based blocks.

Local Energy Mapper (EM_i): according to the workload characteristics of the incoming task, it selects the minimum frequency ($f_{EM,i}$) that allows energy saving, preserving performance loss within a tolerable bound.

Local MPC-based Thermal Controller (TC_i): it trims the frequency to ensure a safe working temperature. Local controllers jointly optimize global system operation by exchanging a limited amount of information at run-time on a neighborhood basis. This is the solution presented in Chapter 4.

In the next paragraphs we provide a detailed description of these three components.

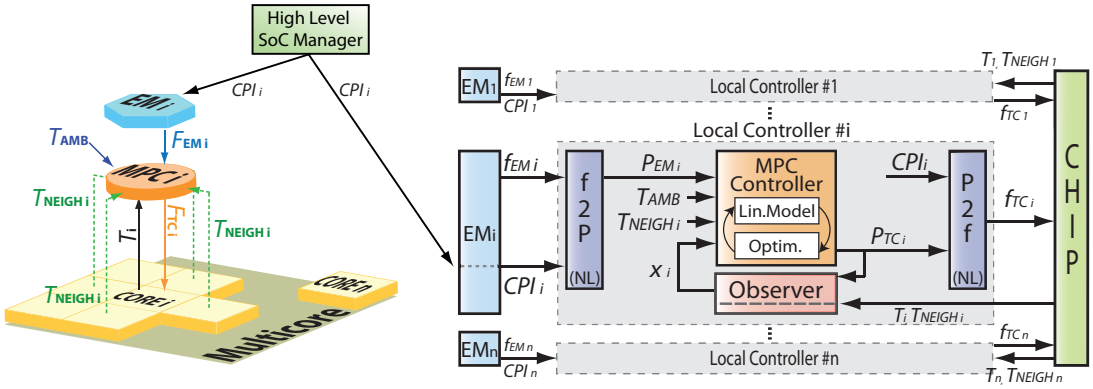


Figure 5.1: General Architecture

5.1.1.1 Local Self-Calibration Routine

The accuracy of the model is of primary importance for the reliability and effectiveness of the control problem. We addressed model uncertainty by self-calibration: each core extracts automatically the local prediction model by applying a set of training stimuli and monitoring the thermal response of the neighborhood area. The distributed controller strategy combined with the distributed thermal model calibration phase allow us to take advantage of the parallelism of the underlying multi-core platform by running different instances of the controller and self-calibration routine in parallel.

The identification mechanism consists in the Distributed ARX approach shown in Chapter 4. We used a second order model,

$$\begin{aligned} x_i(t+1) &= A_i \cdot x_i(t) + B_i \cdot \begin{bmatrix} P_i(t) \\ T_{AMB} \\ T_{NEIGH_i} \end{bmatrix} \\ T_i(t) &= C_i \cdot x_i(t) \end{aligned} \tag{5.1}$$

where i refers to the i-th single-core model, x_i, P_i, T_i, T_{NEIGH_i} are respectively the state vector, the power consumption, the temperature, and the temperature of the neighbors core of the i-th core, T_{AMB} is the ambient temperature.

Notice that the same model has been used to implement the observer block in Fig. 5.1. Indeed, each local thermal controller embeds a Luenberger state-observer to estimate from the core temperature sensors the not-measurable state components. This allows us to use higher order and more accurate prediction models inside the MPC, bringing to significant improvements in the quality of the control with lower overhead compared to a standard Kalman observer. Moreover, the observer uses the local model identified in the self-calibration phase and thus does not incur in errors due to not-in-field calibration.

Finally, it is worth to remark that the Self-Calibration Routine is executed at first off-line during the start up phase. Then, it should be executed every time the model behavior differs from the measurements. This could happen due to the normal aging of the component, due to external environmental causes (e.g. the device is under the sun, or some object could prevent a correct ventilation), or to components failures.

5.1.1.2 The Local Energy Manager

The Energy Mapper (EM_i) associated to the i-th core computes the optimal core frequency ($f_{EM,i}(t)$) which minimizes the energy consumption keeping the performance loss under a fixed percentage decided by the user. Roughly speaking, this is possible by reducing the core speed proportionally to the characteristics of the executing task. If an executing task is CPU and cache intensive (or CPU bound, i.e. the CPU already has the data, without having to read them from RAM memory), the performance improvement scales linearly with increasing frequency. Otherwise, if a task is memory intensive (or memory bound), the performance improvement is relatively insensitive to increase in frequency. In this scenario it is convenient to reduce frequency for memory bound tasks in order to save energy while keeping performance almost unchanged. The Fig. 5.2 shows some experiments conducted in (2) that verify the assumption previously stated.

Fig. 5.2a shows that performance linearly improves with frequency for CPU bound tasks, whereas it only marginally improves for memory bound tasks due to the limitation on executed speed imposed by memory access latencies. Fig. 5.2b, instead, shows the energy consumption of the tasks normalized against the lowest energy consumption. The energy saving increases exponentially reducing frequency for memory bound tasks, whereas it remains almost constant

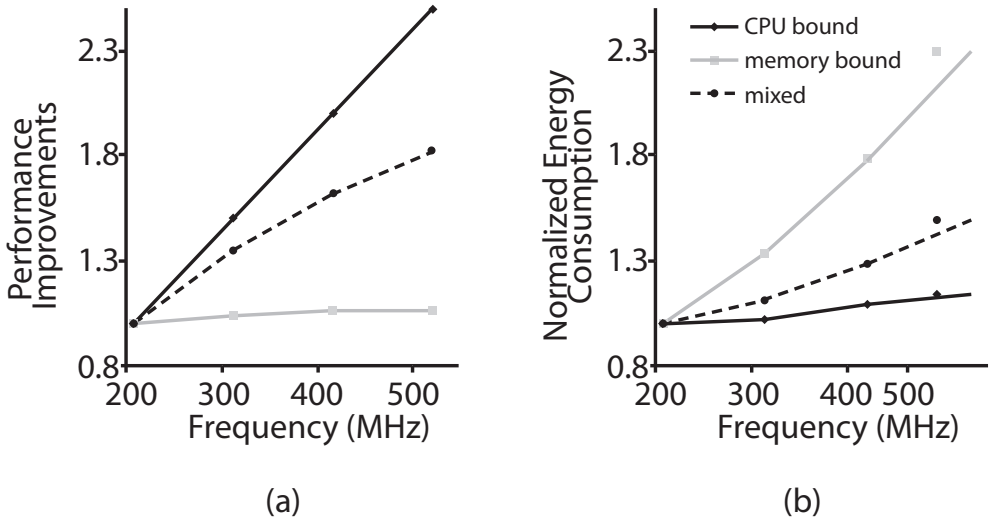


Figure 5.2: Performance Improvement and Normalized Energy Consumption (2)

for CPU bound tasks. Indeed, reducing frequency in a CPU bound task means increasing its execution time, that is we spend less power instantaneously, but for more time. This does not happen in memory bound tasks where the performance delay remain low even decreasing the frequency.

However, tasks typically comprise memory bound and CPU bound phases, hence the best frequency to reduce energy must be find according to the workload of the CPU. The Cycles per instruction (CPI) is the metric we have chosen to measure the workload. Our goal is to select the frequency that minimizes the power consumption while preserving the system performance. Our solution does it by taking advantage of the parallel architecture and letting each core (i) compute autonomously the future frequency in line with the incoming workload requirements.

Indeed, considering an in-order architecture[1] the average time needed to retire an instruction – i.e. to execute and complete it – can be seen as composed of two terms: (i) the $Time_{ALU}$, the portion of time spent in active cycles and (ii) the $Time_{MEM}$, the portion of time spent in waiting for memory cycles. Whereas the first term is proportional to the input frequency, the second one is constant to it and depends on the memory access latency.

Let assume f_{MAX} is the maximum frequency allowed by the system and $CPI_i(t)$ is the workload requirement for the i-th core at the time instant t. Then, the minimum execution time

[1] Multi-core trend is toward integrating a high number of simpler processors (1).

of the task ($Time_i(t)$) is given by,

$$Time_{M_i}(t) = \#_{INST} \cdot \left[\underbrace{1}_{Time_{ALU}} + \underbrace{(CPI_i(t)-1)}_{Time_{MEM}} \right] \cdot \frac{1}{f_{MAX}} \tag{5.2}$$

Assuming $f_{CK,i}(t) = \frac{f_{MAX}}{\alpha}$ the generic frequency of a task, its time execution increase as,

$$Time_{CK,i}(t) = \#_{INST} \cdot [\alpha + (CPI_i(t)-1)] \cdot \frac{1}{f_{MAX}} \tag{5.3}$$

where $0 \leq \alpha \leq 1$ is the $\frac{f_{CK,i}}{f_{MAX}}$ (if 1 the task is CPU bound).

By combining them, the execution time overhead $Time_{\%,i}$ can be represented as function of the new frequency $f_{CK,i}(t)$ and $CPI_i(t)$ as reported in (5.4).

$$Time_{\%,i}(t) = \frac{Time_{CK,i}(t)}{Time_{M,i}(t)} - 1 = \frac{\alpha + (CPI_i(t)-1)}{1 + (CPI_i(t)-1)} - 1 \tag{5.4}$$

Inverting the last equation (5.4) we can find α as,

$$\alpha = Time_{\%,i}(t) \cdot CPI + 1 \tag{5.5}$$

then, knowing that $\alpha = \frac{f_{CK,i}}{f_{MAX}}$ we can write

$$f_{CK,i}(t) = \frac{f_{MAX}}{1 + CPI_i(t) \cdot Time_{\%,i}(t)} \tag{5.6}$$

Therefore, if we suppose to know the predicted CPI for the next sampling interval ($CPI_i([t,t+1]|t)$), we can define the best frequency that preserves the performance within a tolerable penalty $Time_{\%}$ as,

$$f_{EM,i}(t) = \frac{f_{MAX}}{1 + CPI_i([t,t+1]|t) \cdot Time_{\%,i}(t)} \tag{5.7}$$

$f_{EM,i}$ is the best frequency returned as output by the i-th Energy Mapper after it takes as input the predicted CPI_i for the next sampling interval.

5.1.1.3 The Local MPC-based Thermal Controller

The MPC control layer relies on the distributed solution described in Chapter 4.

At the t-th time instant, each Local MPC controller receives as inputs the Energy Mapper output frequency ($f_{EM,i}(t)$), its own core temperature ($T_i(t)$), the temperatures of the physical

neighbours[1] ($T_{NEIGH_i}(t)$), the estimated $CPI_i(t)$, and the ambient temperature ($T_{AMB}(t)$). Then, according to the safe reference temperature (T_{CRIT}) at which the core temperatures ($T_i(t)$) must be constrained, the MPC algorithm adjusts the actual frequency command ($f_{TC,i}(t)$), minimizing the displacement from the Energy Mapper requirement[2] More in detail, the local controller uses the Power Model to convert $f_{EM,i}(t)$ and $CPI_i(t)$ into a target power requirement, $P_{EM,i}$. Then, it exploits the prediction model, feeded with the measurements T_{AMB}, T_{NEIGH_i}, x_i, to compute the estimated core temperature, starting from the current system temperatures. These predictions define the optimization problem:

$$\min_{P_{TC,i}} \sum_{k=0}^{h-1} \|P_{TC,i}(t+k|t) - P_{EM,i}(t+k)\|_Q^2 \tag{5.8a}$$

$$s.t.$$

$$T_i(t+k+1|t) \leq T_{CRIT} \quad \forall k = 0,\ldots,h \tag{5.8b}$$

As output, the block returns the controlled power $P_{TC,i}$ which is equal to $P_{EM,i}$ if the predicted temperature meets the temperature constraint (T_{CRIT}), otherwise it is reduced. Clearly the reduction must be as small as possible to maximize the performance.

The optimization problem can be solved implicitly on-line with a QP solver or explicitly off-line with a multi-parametric QP solver (see Chapter 4 for more details).

5.1.2 The Implementation

This section describes the pseudo-code of the implementation of our solution. First, during system initialization, the self-calibration routine described in Section 5.1.1.1 is executed for each core to gather the local thermal model. Secondly, with the single-core models obtained, we update the weight matrices of each local optimization problem, as shown in Section 5.1.1.3. Third, at each sample we execute the Local Energy Mapper and the Local MPC-based Thermal Controller routines. Subsequently at each controller step we apply the optimal frequency to the controlled core.

[1] The sampling time is assumed small enough as discussed in the previous chapter in order to include only the adjacent cores in the neighborhood.

[2] The computation and actuation times for EM and TC are assumed negligible with respect to sampling time interval. Hence, for mathematical modeling, control outputs are considered generated at the same instant of sampled inputs.

The code below shows the list of operations executed in parallel by each core to gather the local thermal model, adapt the controller parameters and control the system performance.

Pseudo Code

```
1  SYSTEM INITIALIZZATION PHASE:
2    Apply a PRBS task sequence;
3       get P_i, T_i, T_NEIGH_i;
4       obtain α, and β with the distributedARX approach;
5       find the state space representation A_i, B_i, C_i with C_i = [10];
6
7  CONTROLLER ROUTINE
8    Initialize the weight matrices H_i, M_i, and the state x̂_i(0);
9    FOR EACH SAMPLE
10           get previous f_TC,i(t−1) and CPI_i(t−1);
11           compute the optimal f_EM,i(t);
12           convert to P_EM,i(t) using the Power Model;
13           update g_i and b_i;
14           solve the QP problem and find P_TC,i(t) with hot-start;
15           compute f_TC,i(t);
16           estimate x̂_i(t);
17   END_FOR
```

More in detail:

Line 1 during the system initialization phase we execute in parallel in each core the Self-Calibration Routine;

Line 2-3 we apply a pseudo-random task sequence to each core and we collect the core power and local neighborhood temperature traces;

Line 4 we execute the *ARX* optimization in each core to obtain the α and β parameters;

Line 5 we convert the model in the state-space form using the observability matrix to give physical meaning to the first component of the state vector. It correspond to the measured core temperature;

Line 8 we define the constant matrices used in the QP problem (4.22), H_i, M_i), and assign the initial state, $\hat{x}_i(0)$, to the model ;

Line 9 at each time step the loop from line 9 to line 17 is repeated;

Line 10 the Energy Mapper read the previous step core frequency $f_{TC,i}(t-1)$ and $CPI_i(t-1)$, and it estimates the future workload requirement $CPI_i(t)$ and compute optimal target frequency $f_{EM,i}(t)$;

Line 12 the Thermal Controller receives the target frequency and the workload from the EM_i and then it converts them into the target power using the nonlinear function (4.1);

Line 13 the vector g_i, dependent on $P_{EM,i}(t)$, and the vector b_i, dependent on $\hat{x}_i(t)$, $T_{AMB}(t)$ and $T_{NEIGH_i}(t)$, are updated.

Line 14 starting from the previous optimal solution $P_{TC,i}(t-1)$ the solver finds the optimal $P_{TC,i}(t)$, solution of the QP problem.

Line 15 $P_{TC,i}$ and workload are used to find $f_{TC,i}$ inverting the function (4.1) as an example with the Brent's algorithm.

Line 16 the observer estimates the state $\hat{x}(t)$ knowing the $f_{TC,i}(t)$ and $T_i(t)$. This state will be used as initial state by the MPC controller to estimate the future temperature of the core.

The explicit MPC implementation is similar to the one described above with the only difference in lines 13-14. Indeed, the optimal $P_{TC,i}$ is obtained with (4.25). Both implementations, thank to our distributed strategy and the hot-start QP solver (in the implicit implementation), have low overhead and are suitable to be executed with a sample rate of 1-10ms.

5.1.3 Experimental Results

The solution has been tested on the accurate virtual platform described in Appendix B that combines Simics a x86 ISA functional simulator, and GEMS (7), a complex memory hierarchy timing-accurate model. This virtual platform emulates a general-purpose multi-core running in a full system. It provides a flexible and effective tool to support the design space exploration of power, thermal and reliability control-theory-based close-loop resource management solution.

We executed each Local Energy Mapper and Local MPC-based Thermal Controller with a sampling time of $1ms$. The Local MPC-based Thermal Controller routine embeds the explicit MPC implementation and estimates full state with the state observer. Even if the controller routines are not executed directly on the target multi-core, the complexity analysis in Section 4.3.1 demonstrates that the distributed solution has negligible run-time. Thus, the perturbation due to its computations to the program execution flow can be neglected.

The floorplan used to test the proposed solution on the virtual platform is the Xeon® X7350 already showed in Fig. 4.7 in Chapter 4. On each core we run different PARSEC 2.1 (9) benchmarks workloads each one with a number of tasks equal to the number of cores. The temperature constraint for each core is $T_{CRIT} = 330\,^{\circ}K$)[1]. The test has been performed on four control configurations:

[1]Used thermal model is calibrated on a device with high performance thermal dissipation dynamics, indeed to stress our thermal controller we are forced to use a lower temperature constraint

Original : without any type of control;

Energy Mapper : with the energy control, but without the thermal control;

Centralized MPC : with the energy control and with the centralized MPC thermal control;

Distributed MPC : with the energy control and with the distributed MPC thermal control.

The results has been compared using as metrics the maximum temperature overshoot,namely the maximum temperature reached minus T_{CRIT}, and the percentage of time the critical temperature is violated. A quality of service (QoS) degradation metric is then used to quantify the performance loss respect to the centralized solution and it has been computed as the mean squared error between the energy mapper frequency target (f_{EM}) and the one applied to the system by the controller (f_{TC}). We relativized it against the centralized controller one.

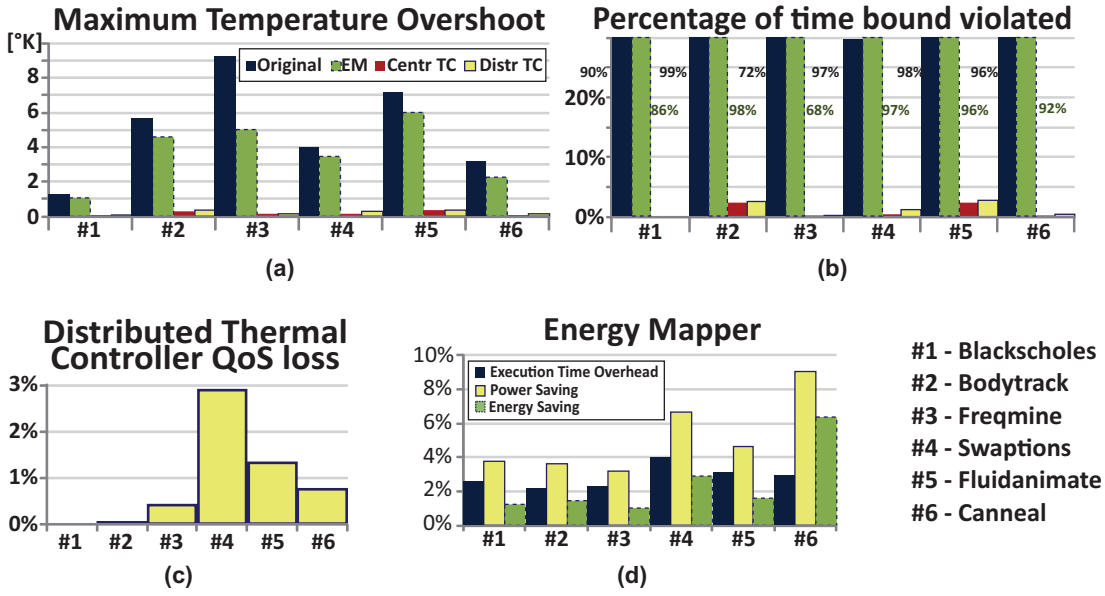


Figure 5.3: Virtual platform test results

Fig. 5.3 shows the results collected. First, from Fig. 5.3a,b and c, we can notice that the proposed distributed solution performs as well as the centralized one. In particular Fig. 5.3a, the maximum overshoot in Kelvin degrees, and Fig. 5.3b, the percentage of time the constraint is violated, show that both solutions are capable of drastically reducing the portion of time in which each core runs out of the thermal bound. Looking at the performance loss (Fig. 5.3c), we notice that our proposed solution performs at the same level of the centralized one, with

a degradation less than 3%. Finally in more symmetrical workloads[1], such as `swaptions`, `fluidanimate`, `canneal`, we noticed that the average frequency applied to the external cores (#1, #4) is kept lower (up to -14%) than the internal cores. This is a sign that the MPC controller is able to optimize the core frequency locally, taking advantage of the difference between the local thermal models. Indeed the external thermal models have less thermal dissipation headroom since thermal model considers the chip lateral boundary adiabatic (8). Finally, Fig. 5.3d shows the performance of the EM alone, while allowing a performance penalty of $T_{\%,i}(t) = 5\%$. We can notice that it is able to maintain the performance overhead under the selected threshold while achieving a significant power and energy saving.

5.2 A feasible two-layer distributed MPC approach to thermal control of Multiprocessor Systems on Chip

The central aim of the control solution presented in this section is to address the MPSoCs thermal issue by using a fully distributed control solution able to ensure feasibility, reliability and efficiency.

The main idea to develop a solution with these properties is to exploit the results obtained in Section 4.4.4. Indeed, we have proved that a thermal system modeled with a PDE is always feasible in solving temperature capping problem. However, temporal and spatial discretization affect this property. The uncertainties introduced by discretizing the model as well as unpredictable measures (e.g. the workload cannot be accurately forecasted and usually it is considered equal to the past one assuming low variability between two time steps) prevent the use of an ideal distributed MPC solution as the one shown in the Chapter 4, where each local controller supervises one core, maintaining the temperature under a fixed threshold, $\bar{T}_{CRIT}$, and maximizing at the same time the workload request from the High Level SoC Manager. Before proceeding, it is worth to make some considerations on the plant model we used for simulations. It has been built assuming that the main contribution for heat dissipation occurs through the heat spreader located on the top of the chip, whereas the dissipation along lateral boundaries has been considered negligible. For this reason the steady-state temperature when the minimum power is given to the cores is uniformly distributed and the critical space varying threshold $\bar{T}_{CRIT}$ considered in Section 4.4.4 can be considered equal to T_{CRIT}. A solution with $\bar{T}_{CRIT} \neq T_{CRIT}$ is actually under development.

[1] The parallel benchmark executes the same code on all the processors.

A first attempt to take into account model uncertainties is using a conservative identification approach as the one shown in Section 4.1.2. Overestimating the future temperature of the cores, the controller applies a more moderate control action. However, the identified model uses a set of data collected from the real system that may not capture all possible behaviors, leaving the problem of model uncertainties not completely solved.

Another simple solution may be to put a margin between the critical temperature T_{CRIT} and the maximum temperature allowed by the MPC, that we called τ_{MPC}. Thus, the temperature could violate τ_{MPC} due to underestimations on predictions, but it remains below T_{CRIT} because of the margin.

Nevertheless, guarantees for conservative thermal capping requires a significantly large margin – increasing with sampling time period – that strongly impacts on performance. Indeed, as temperature threshold decreases, also power consumption and core speed reduce as well. A even bigger problem is related to our decision of using a distributed solution, which, as already shown in Section 4.4.5 and due to the sampling time and model uncertainties, may take to an infeasible solution.

Our solution considers a two-layer hierarchical architecture. The higher layer exploits the distributed MPC scheme previously mentioned to address the thermal capping issue, maximizing performance at the same time. The lower one, namely the Safety layer, guarantees the control feasibility, the respect of the T_{CRIT} bound and the reduction of the MPC margin, favoring better performance. Its functioning is totally independent from the system at hand since the knowledge of the model of the system is unnecessary.

5.2.1 The Architecture

The Fig. 5.4 depicts the block diagram of the proposed solution. Each local controller comprises three components, one of them is computed off-line, the other two works at run time and differs for the implementation. One it is suppose to be software-based, whereas the other one is hardware-based.

Local Iterative Identification Procedure : it derives, off-line, the local, but interacting, thermal prediction model adopted in Local MPC Controller.

Local Safety Controller : it switch the *i*-th core frequency to the minimum value if its temperature cross the critical threshold (i.e. τ_{SWITCH}). It guarantees feasibility.

Local MPC Controller : it trims the frequency to ensure a safe working temperature (i.e. $\leq \tau_{MPC}$). Local controllers jointly optimize global system operation by exchanging a limited amount of information at run-time on a neighborhood basis. This is the solution presented in Chapter 4.

In the next paragraphs we provide a detailed description of these three components.

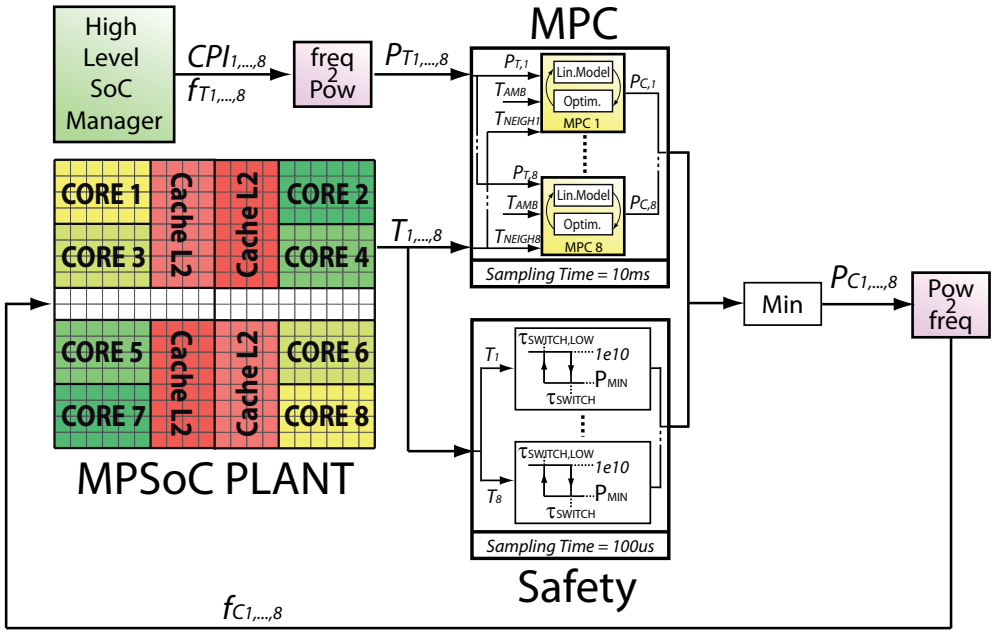


Figure 5.4: General Architecture

5.2.1.1 Local Iterative Identification Procedure

The basic concept of MPC is to use a dynamic model to forecast system behavior, and optimize the predictions to produce the best control decision. Thus, the efficiency of the controller is strongly related to the accuracy and complexity of the model used for predictions. In order to identify a model with such characteristics each core runs an off-line Iterative Identification Procedure. This procedure recalls repeatedly the H_∞ problem shown in Section 4.1.2. At each iteration, the approach finds the model that minimizes the maximum error between the predicted temperature and the measured one (obtained applying a set of training stimuli to the real processor), keeping the error always positive (for conservative reasons). The technique start considering all possible measurements as model input, then at each iteration it discards the negligible one.

The model considered for each core is a second order model,

$$\begin{aligned} x_i(t+1) &= A_i \cdot x_i(t) + B_i \cdot u(t) \\ T_i(t) &= C_i \cdot x_i(t) \end{aligned} \tag{5.9}$$

where $C_i = [1\ 0]$, and $u(t)$ are the inputs selected after the iterative approach has been applied. We found that these inputs are $[P_i(t)\ T_{AMB}\ T_{NEIGH_i}]$.

It is also worth to note that the model, having a order greater than one, needs an estimator to evaluate the unknown state. The particular structure of the model allow us to obtain the real value of the states by simply storing the past control inputs and temperatures. Indeed, considering the second order model obtained as result of the H_∞ problem,

$$T(t|t-1) = \alpha_2 \cdot T(t-1) + \alpha_1 \cdot T(t-2) + \beta_2 \cdot u(t-1) + \beta_1 \cdot u(t-2),$$

the unknown state can be estimated as,

$$x_2(t) = a_2 \cdot T_{t-1} + b_2 \cdot u_{t-1}$$

5.2.1.2 Local Safety Controller

The Safety layer is composed by a set of hardware-based switch controllers, one for each core, completely independent from the MPC layer, namely the Local Safety Controllers. When the current temperature of the i-th core, T_i, reaches the critical temperature threshold, T_{CRIT}, the correspondent switch controller bypasses the MPC layer providing to the core a minimum power, P_{MIN}, until the temperature reaches a fixed lower value, $\tau_{SWITCH,LOW}$.

The three central goals of this layer are: to ensure feasibility, to respect the temperature constraint T_{CRIT} and to improve performance respect to the MPC solution with completely conservative margin (i.e. the MPC controller is designed to avoid the Safety layer activation).

We prove the first two properties directly using the results established in Section 4.4.4, indeed,

Proposition 5.2.1. *Consider the system (4.31) and assume that temperatures on sources are measured, then there exists a set of* local *time-continuous switch regulators that solve the thermal capping problem and whose form is:*

$$P_{C,i}(t) = \begin{cases} P_{MPC,i}(t) & T_i(t) \leq T_{CRIT} \\ P_{MIN} & Otherwise \end{cases} \tag{5.10}$$

where $P_{C,i}$ and $P_{MPC,i}$ are respectively the final controlled power and the power chosen by the MPC controller supervising that core.

Proof. From the proposition 4.4.2, the maximum temperatures are known because of the sensors on sources. The property 4.4.7 ensures the feasibility by applying instantaneously a limited control action, P_{MIN}, on sources which are exceeding the thermal limit T_{CRIT}. On the other hand, according to the proposition 4.4.3, the sources which are not controlled to P_{MIN} cannot cause the threshold violation of the nullified ones (or elsewhere), since their temperatures are lower than T_{CRIT}. □

However, this result has theoretical validity under continuous-time hypothesis, but the clock driven nature of MPSoCs imposes the use of a discrete-time controller. Consequently, as temperature violation can happen during sampling interval, it is necessary to provide a margin, that is to decrease the critical temperature T_{CRIT} to a value τ_{SWITCH} according to the sampling time chosen. Notice that, because the Safety layer is hardware-based, the sampling time can be very small. A procedure to find τ_{SWITCH} relies on the inversion of the discrete models (5.9) of each core. It consists in finding the initial state that reaches the critical temperature T_{CRIT} after a sampling interval, assuming the worst possible case (i.e. maximum power, maximum ambient temperature and maximum temperature of neighbors). Therefore, inverting the second order model (4.11) we obtain,

$$\begin{bmatrix} \tau_{SWITCH} \\ x_{2,init} \end{bmatrix} = A^{-1} \cdot \left(\begin{bmatrix} T_{CRIT} \\ x_{2,fut} \end{bmatrix} - B \cdot \begin{bmatrix} P_{MAX} \\ T_{AMB,MAX} \\ T_{neighs,MAX} \end{bmatrix} \right)$$

Notice that the previous equation includes three unknown variables, τ_{SWITCH}, $x_{2,init}$ and $x_{2,fut}$, but only two equations. To solve the problem we overestimated the value of $x_{2,fut}$ setting an appropriate ΔT and using the previous equation which can be written as,

$$x_{2,fut} = a_{12}^{-1}(T_{CRIT} + \Delta T - a_{11} \cdot T_{CRIT} - B \cdot \left(\begin{bmatrix} P_{MAX} \\ T_{AMB,MAX} \\ T_{neighs,MAX} \end{bmatrix} \right) \tag{5.11}$$

where a_{11} and a_{12} are the first line elements of A.

In order to completely define the switch controller we need to set the value $\tau_{SWITCH,LOW}$ that determines its deactivation. As the nominal behavior resumes, the state of the system must be feasible, hence lower than the temperature limit imposed by the MPC, τ_{MPC}. We set $\tau_{SWITCH,LOW} = \tau_{MPC} - \Delta$ where Δ is a small arbitrary value.

Thus, summarizing, the resulting controller is a hardware-based, discrete-time, hysteresis controller, capable of guaranteeing feasibility.

The last property to be discussed is how the use of the Safety layer can improve global performance of the system. The central idea is to design the two layers knowing the existence of the other. Indeed, designing a stand-alone MPC layer would correspond to set the threshold τ_{MPC} to the maximum value that prevent the use of the Safety layer, causing performance degradation. Instead, we can set τ_{MPC} to a higher value, allowing the cores to be faster and the Safety layer to intervene more frequently. How to choose the τ_{MPC} value is described in the next section.

5.2.1.3 Local MPC Controller

The MPC layer, as well as the Safety layer, has been designed with a distributed structure. Such configuration allows the controller to be computationally more efficient, as shown in Section 4.3 and more reliable, since the break down of a core cannot compromise the whole system performance. Each controller solves an optimization problem which exploits the predictions, computed with the identified single-core model, to find the best control decision that maintains the temperature under τ_{MPC}. When the temperature of the i-th core reaches the Safety threshold, τ_{SWITCH}, the i-th switch controller takes the control of the core imposing the minimum power. Immediately, when the switch controller deactivates, the MPC controller, which has remained active, takes place and continue with the best control action computed at the beginning of the previous sampling time.

The optimization problem solved inside the Local MPC Controller is the one presented in Chapter 4, that is,

$$\min_{P_{C,i}} \sum_{k=0}^{h-1} \|P_{C,i}(t+k|t) - P_{T,i}(t+k)\|_Q^2 \tag{5.12a}$$

$$s.t.$$

$$T_i(t+k+1|t) \leq \tau_{MPC} \quad \forall k = 0,\ldots,h \tag{5.12b}$$

An important step in the development of the global control solution is choosing the MPC threshold. Ideally $\tau_{MPC} = \tau_{SWITCH}$, but, as already mentioned, the existence of uncertainties would involve a frequent intervention of the Safety controller. We need to provide a margin between τ_{MPC} and τ_{SWITCH}. Clearly, the greater is the margin, the more conservative is the controller, and the lower are the performance (the MPC would maintain the core speed to a lower level). However, also setting a low margin would correspond to low performance

due to the high activation frequency of the Safety layer. Thus, the central idea of the two-layer solution is to choose a MPC threshold greater than the completely conservative one (i.e. the one that prevents the activation of the Safety layer), that lets the Safety layer managing critical situations, in order to maximize the performance. Note that this latter is a key point for manufacturers whose profit is strictly related to performance.

Therefore, the τ_{MPC} results from the solution of a trade-off problem between conservativeness of the MPC controller and frequency of activation of the Safety controller. Nevertheless, the great number of factors affecting the controller, as external inputs and the already cited model uncertainties, make the use of a rigorous analytical estimation of τ_{MPC} difficult. Thus, we developed empirically-based methods to impose this margin.

A first simple method consists in running typical benchmarks, e.g. PARSEC 2.1 (9), and calibrating τ_{MPC} as the value that reduces the violations of τ_{SWITCH} under an arbitrarily chosen percent of time. This solution let the user the freedom of choosing the degree of exploitation of the Safety layer. However, if our goal is maximizing the computing performance, we need to solve an optimization problem. We search for the τ_{MPC} that maximizes an objective function – the integral of the cores frequency – constraining τ_{MPC} in the interval $[\tau_{MPC,CONS},\ \tau_{CRIT}]$, that is

$$\min_{\tau_{MPC}} \sum_{i=1}^{N} \int_{0}^{Time} f_{C,i}(\tau_{MPC}, t)\, \mathrm{d}t \tag{5.13a}$$

$$s.t.$$

$$\tau_{MPC,CONS} \leq \tau_{MPC} \leq T_{CRIT} \tag{5.13b}$$

where N is the core number, f_C is the controlled cores frequency, $[0, Time]$ is the time interval of the benchmarks, and $\tau_{MPC,CONS}$ is the τ_{MPC} in the completely conservative case (i.e. the Safety layer is never used). We solved the problem for each benchmark and we selected the optimal value of τ_{MPC} as the average of the result of each problem. However, the solution of the problem (5.13) requires to run a great amount of simulations to collect the frequency data for computing the integral in the objective function (one for each value of τ_{MPC} and for each benchmark). In order to reduce the number of simulations we quantized the set of the τ_{MPC} values.

Finally, it is worth to note that the upper bound of τ_{MPC} is T_{CRIT}, not τ_{SWITCH}. The conservativeness of the identified model explains this choice, indeed the temperatures limited

by the MPC could be lower than τ_{SWITCH} even if $\tau_{MPC} > \tau_{SWITCH}$. In this particular case $\tau_{SWITCH,LOW} = \tau_{SWITCH} - \Delta$.

5.2.2 The Implementation

The code below shows the list of operations executed in parallel by each core to gather the local thermal model, adapt the controller parameters and control the system performance.

Pseudo Code

```
1  OFF-LINE MODEL IDENTIFICATION & CONTROLLER SETTINGS:
2    Apply a PRBS task sequence;
3       get P_i, T_i, T_NEIGH_i;
4       obtain α, and β with the H∞ Iterative Procedure;
5       find the state space representation A_i, B_i, C_i with C_i = [10];
6       setting of τ_SWITCH, τ_MPC;
7
8  CONTROLLER ROUTINE
9    Initialize the weight matrices H_i, M_i, and the state x̂_i(0);
10   FOR EACH MPC CONTROLLER SAMPLE
11           get f_T,i(t) and CPI_i(t) = CPI_i(t − 1);
12           convert to P_T,i(t) using the Power Model;
13           update g_i and b_i;
14           solve the QP problem and find P_C,i(t) with hot-start;
15           FOR EACH SAFETY CONTROLLER SAMPLE
16                 compute the min between P_C,i(t) and the output of the Safety Layer;
17           END_FOR
18           compute f_C_i(t);
19           estimate x̂_i(t);
20   END_FOR
```

More in detail:

Line 1 during the system initialization phase we execute in parallel in each core the Self-Calibration Routine;

Line 2-3 we apply a pseudo-random task sequence to each core and we collect the core power and local neighborhood temperature traces;

Line 4-6 according to Fig. 5.5:

1. we **identified the prediction models** of each core performing the iterative procedure of Section 4.1.2. As already mentioned the distributed models may be infeasible, therefore we avoided the feasibility test of Section 4.4.5 (the Safety layer is enough for ensuring feasibility).
2. We found the threshold τ_{SWITCH} of the Safety layer by inverting the models.

3. We detected the **conservative MPC threshold** ($\tau_{MPC,CONS}$) by running a set of benchmarks on the system supervised by the two-layer controller. We set as initial threshold $\tau_{MPC} = \tau_{SWITCH}$ and we used as prediction model the identified one. Then, we decreased the MPC threshold until we detected that cores temperatures never exceed τ_{SWITCH} for each benchmark.
4. we found the final τ_{MPC} that maximizes the performance. Starting from $\tau_{MPC} = \tau_{MPC,CONS}$, we increased τ_{MPC} looking for the value that maximize the integral of the cores frequencies and then we performed a average of the results of all benchmarks.

Line 9 we define the constant matrices used in the QP problem (4.22), H_i, M_i), and assign the initial state, $\hat{x}_i(0)$, to the model ;

Line 10 at each sampling time of the MPC layer the loop from line 10 to line 20 is repeated;

Line 11 the High Level SoC Manager provides the frequency $f_{T,i}(t)$ and $CPI_i(t)$ requirements;

Line 12 the Thermal Controller converts them into the target power using the nonlinear function (4.1);

Line 13 vector g_i, dependent on $P_{T,i}(t)$, and vector b_i, dependent on $\hat{x}_i(t)$, $T_{AMB}(t)$ and $T_{NEIGH_i}(t)$, are updated.

Line 14 starting from the previous optimal solution $P_{C,i}(t-1)$ the solver finds the optimal $P_{C,i}(t)$, solution of the QP problem.

Line 15 at each sampling time of the Safety layer (faster than the MPC layer one) the loop from line 15 to line 17 is repeated;

Line 16 if the current core temperature is greater than τ_{SWITCH}, the Safety Controller imposes $P_{C,i}(t) = P_{MIN}$;

Line 18 $P_{C,i}$ and workload are used to find $f_{C,i}$ inverting the function (4.1) as an example with the Brent's algorithm.

Line 19 the observer estimates the state $\hat{x}(t)$ knowing the $f_{C,i}(t)$ and $T_i(t)$. This state will be used as initial state by the MPC controller to estimate the future temperature of the core.

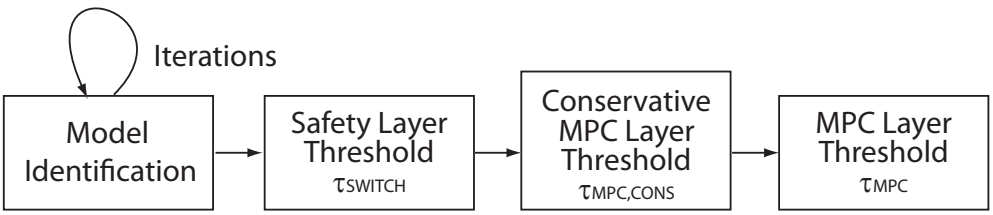


Figure 5.5: Off-line steps summary

Notice that the Safety layer is hardware based and can be executed with a sample rate of 100us, whereas the MPC layer is software based and its sample rate is 10ms.

5.2.3 Experimental Results

We tested our solution on a Matlab/Simulink simulator developed exploiting the finite element approach described in Appendix B. Fig. 5.6 shows the plant we used for our simulations. This is a Xeon-like platform as the one shown in Fig. 4.7, but we decided to double it to show that our control solution is reliable also for a bigger number of cores. The final layout is similar to the Enterprise Xeon® Processor presented in (4).

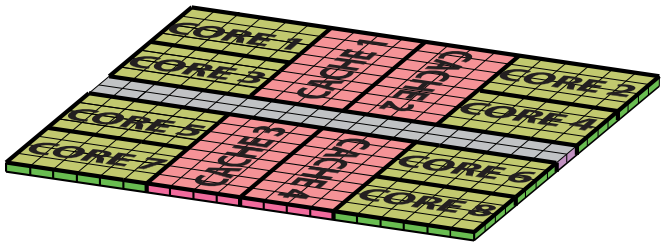


Figure 5.6: Simulator layout

The admissible power consumptions of each core ranges from $P_{MIN} = 4.38W$ to $P_{MAX} = 25W$ corresponding to a frequency of 1600MHZ and 2970MHz and the idle power – power consumed when the core is shut down – is $P_{IDLE} = 0.25W$. The power dissipated by the caches is the 30% of the power consumed by the cores directly connect to them.

For each local MPC controller we identified, according to the H_∞ iterative procedure, a second order model to forecast the temperature of the supervised core. We collected the inputs and output data applying a PRBS power trace to the plant and then we solved the optimization problem (4.10) to find the unknown parameters. At the first iteration we considered all possi-

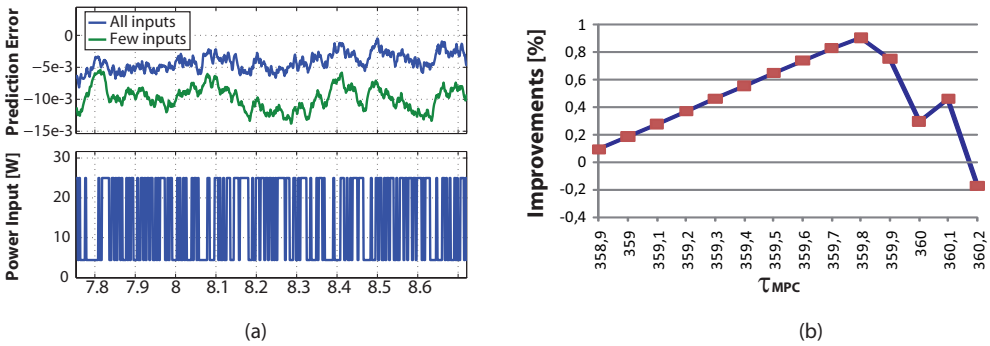


Figure 5.7: (a) Temperature prediction error comparison; (b) Performance comparison with different τ_{MPC}

ble inputs: all core powers, the ambient temperature and the temperature of all cores (except the *i*-th one). The results showed that the power inputs, with the exception of the *i*-th one, had a negligible contribution to the final temperature of the cores. As an example the power consumption contributions for the core 1 are:

$$B = \begin{bmatrix} 4.7e-2 & 6e-6 & -2.5e-6 & 5.6e-6 & -6.4e-6 & -1.5e-6 & 2e-6 & 5.7e-6 & \dots \\ \underbrace{-4.5e-2}_{P_{C,1}} & \underbrace{8e-5}_{P_{C,2}} & \underbrace{-2.6e-4}_{P_{C,3}} & \underbrace{3.8e-5}_{P_{C,4}} & \underbrace{-8.54e-5}_{P_{C,5}} & \underbrace{-7e-4}_{P_{C,6}} & \underbrace{5e-5}_{P_{C,7}} & \underbrace{8e-7}_{P_{C,8}} & \dots \end{bmatrix}$$

In the second iteration we deleted the temperature contributions of the cores situated far from the *i*-th core. We considered only the north, south, east and west sides cores.

In Fig. 5.7a we show a comparison between the models identified respectively in the first (*All inputs*) and last step (*Few inputs*) of the iterative procedure. The temperature prediction error results similar for both the cases. As an example, we reported below the model obtained for the core 3,

$$\begin{bmatrix} T_3 \\ x_{2,3} \end{bmatrix}_{t+1} = \begin{bmatrix} 1.5 & 1 \\ -0.5 & 0 \end{bmatrix} \cdot \begin{bmatrix} T_3 \\ x_{2,3} \end{bmatrix}_t + \\ + \begin{bmatrix} 4.0e-2 & 3.5e-4 & 4e-4 & 7e-4 & 3.8e-4 \\ -4.5e-2 & 0 & 0.5e-4 & -2.7e-4 & 6.7e-4 \end{bmatrix} \cdot \begin{bmatrix} P_{C,3} \\ T_E \\ T_1 \\ T_4 \\ T_5 \end{bmatrix}_t$$

We set $\tau_{SWITCH} = 359.7°K$, assuming $\Delta T = 0.25°K$ in (5.11), and the conservative MPC threshold $\tau_{MPC,CONS} = 358.8°K$ by iteratively decreasing the τ_{MPC} until we detected no Safety layer interventions[1].

The procedure to find τ_{MPC} is similar to the one used to find $\tau_{MPC,CONS}$. We set $\tau_{MPC} = \tau_{MPC,CONS}$ and we increased iteratively its value by $0.1°K$. For each simulation we stored the integral of the frequency as performance metric.

In Fig. 5.7b we compared the performance with respect to the τ_{MPC} chosen. Notice that because we run the simulation with different benchmarks the values plotted in the figure correspond to average values. The final value for τ_{MPC} is $359.8°K$ that is greater than τ_{SWITCH}. As already mentioned this is due to the conservativeness of the models obtained with the identification procedure. The improvements of performance are about 1% in average. The lower hysteresis threshold of the switch controllers, $\tau_{SWITCH,LOW}$, is equal to $\tau_{SWITCH} - 0.1°K$, that is $359.6°K$.

[1] At each iteration we decrease the previous $\tau_{MPC,CONS}$ by $0.1°K$.

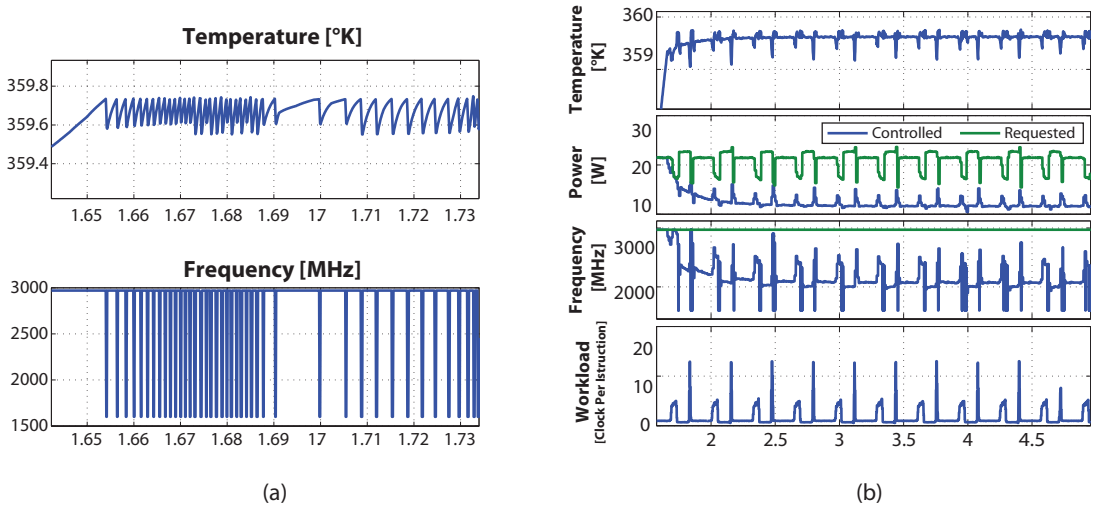


Figure 5.8: Simulation results of the core 3

Fig. 5.8 shows the temperature response of core 3. Fig. 5.8a shows the case when only the Safety layer is active. The temperature results bounded under the $T_{CRIT} = 360°K$ and the controller is feasible, but the performance degrades due to the continuous activation and deactivation of the Safety layer.

Fig. 5.8b, instead, shows the response of the core 3 when a PARSEC 2.1 benchmark (Fluidanimate) is assigned to the controlled system. The temperature is perfectly bounded under T_{CRIT} and the Safety layer intervenes only when the temperature crosses τ_{SWITCH} setting the frequency to 1600MHz.

Finally Fig. 5.9 shows the simulation results for the core 3 when different MPC thresholds were applied: the τ_{MPC} that maximizes the performance ($\tau_{MPC,MAX}$), the τ_{MPC} that reduces the violation under the 0.1% ($\tau_{MPC,0.1\%}$) and the τ_{MPC} completely conservative ($\tau_{MPC,CONS}$). As illustrated the frequency for ($\tau_{MPC,MAX}$) is the greatest on average even though Safety layer intervenes more frequently than in the other case.

5.3 Communication-aware solution

Increasing the number of cores on a single chip substrate is the actual trend for improving processors performance. Dozens or hundreds of core are expected in the next future. As a result, many-core architectures will substitute the actual multi-core one. However, behind the improvement of the throughput per Watt, many-core systems introduce new challenges tied

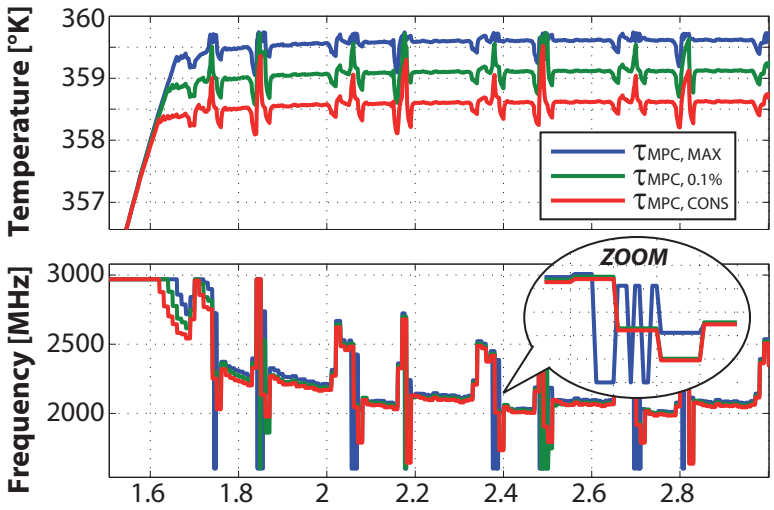


Figure 5.9: Performance comparison with three different τ_{MPC}

to their large scale structure. We can define a processor as many-core if it has a number of cores large enough (several tens of cores) to prevent the use of the traditional multi-processor techniques due to issues of congestion. Indeed, the data and instruction traffic generated by so many cores preclude the use of shared buses and shared memories between the cores (used in the so called cache-coherent approach). Many-core architectures may be non-cache-coherent and the communication between cores may take place via message passing through networks-on-chip (NoCs). Message passing system is usually implemented using a Message Passing Interface (MPI), a library specification that has become the standard for message-passing-based parallel programming. Additionally, many-core chips can benefit from computation migration and mechanisms for actively balancing load which can enhance system throughput and power management. This latter, in particular, represents a critical constraints in today computing platform. A common approach to reduce power consumption is DVFS (Dynamic Voltage and Frequency Scaling) that exploits the dependence on frequency and the square of the supply voltage of the power. Roughly speaking we save energy by reducing frequency and voltage of one core when it experiences thermal issues or performance would experience low degradation. Nevertheless, whereas this technique achieves good results for single/multi-core processors, they do not capture the unique performance-power tradeoffs in many-core systems with MPI. Indeed, voltage and frequency changes affect MPI since its functions or calls are executed on the cores subjected to DVFS regulation (5) (6). According to the results achieved in (5) testing some typical benchmarks on a SCC processor (see Appendix B), the performance and

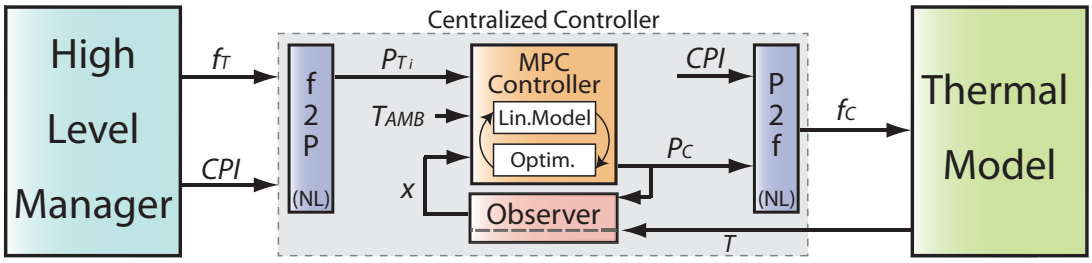


Figure 5.10: Proposed solution architecture

the power efficiency of the many-core system show some benefit if the frequencies of the core that are communicating are balanced.

The aim of this solution is to manage the core frequencies in order to equalize the frequency of the cores that communicate with each other. This management must be performed in a thermally safe environment, therefore the proposed solution consists in updating the thermal control solution for respecting this new requirement. In the next sections we presented our solution, however it is important to remark that a real analysis of the performance improvements has not been conducted yet. This Section has only the aim of presenting a *linear* MPC-based thermal control solution able to account dynamically constraints on frequency. Finally this solution can be exploited also in the case of two or more cores may need to have the same DVFS level due to application requirements or hardware limitations.

5.3.1 Architecture

Differently from the other solution presented, this solution consists of one layer that uses the centralized MPC-based control paradigm,

$$min \sum_{k=0}^{h-1} \|P_T(t+k|t) - P_C(t+k|t)\|_Q^2 \tag{5.14a}$$

$$s.t.$$

$$T_j(t+k+1|t) \leq T_{CRIT} \quad \forall j=1,\ldots,n_T \quad \forall k=0,\ldots,N \tag{5.14b}$$

where $P_C = [P_{C,1}, \ldots, P_{C,N}]$ and $P_T = [P_{T,1}, \ldots, P_{T,N}]$ are respectively the vector of the power consumption allowed to be dissipated by the cores and the one requested by the High Level SoC Manager, $Q_{N\times N}$ is the weight matrix, T_{CRIT} is the critical temperature, and $T = [T_1, \ldots, T_p]$ is the vector of the temperatures measured on the chip (see the control architecture in Fig. 5.10).

We assumed to have a sensor placed in the center of each core, i.e. $p = N$ where N is the number of cores. The notation $T_j(t+k+1|t)$ means the temperature estimated for the future time $(t+k+1)$ based on the information available at time t which implies the existence of a discrete-time thermal model relating the power consumption (and the ambient temperature) with the future temperature of the cores,

$$\begin{aligned} x(t+1) &= A \cdot x(t) + B \cdot \begin{bmatrix} P(t) \\ T_{AMB} \end{bmatrix} \\ T(t) &= C \cdot x(t) \end{aligned} \tag{5.15}$$

The model has been identified using the distributed ARX approach shown in Section 4.1.1. This approach returns N single-core models that can be composed to obtain a global model. Each single-core model has two states (we chose a second order model for each core), the first o which corresponds to the core temperature (T_i), the second, instead, is unknown ($x_{2,i}$). Composing all these states together we obtain the state of the global model, $x(t)$, that has dimension $2N \times 1$. We decided to grouped all the temperature measurements in the first half of the vector, therefore $x(t) = [T_1, \ldots, T_N, x_{2,1}, \ldots, x_{2,N}]$. A is a $2N \times 2N$ matrix, B is $2N \times (N+1)$ matrix and C is a $N \times 2N$ matrix of the form $[I_N \; 0_N]$.

It is worth to note that in the problem formulation (5.14) we exploited, as in the rest of the thesis, the possibility of separating the nonlinear frequency-to-power relation in order to have a linear MPC problem. The Power Model we considered is the same of (4.1) but we substituted to the supply voltage V_{dd} the nonlinear function of the frequency $h(f) = a_1 \cdot f^{a_2} + a_3$, that is,

$$\begin{aligned} &P = P_{dynamic} + Pstatic = \\ &\left[k_{A1} \cdot f^{k_B} + k_{A2} + (k_C + k_D \cdot freq) \cdot CPI^{k_E}\right] + \left[Z \cdot V_{dd} \cdot T^2 \cdot e^{\frac{-q \cdot V_t}{K \cdot T}}\right] \end{aligned} \tag{5.16}$$

where $k_{A1} = 3.8696e-008$, $k_B = 2.4090$, $k_{A2} = 1.1025$, $k_E = -0.3016$, $k_C = -4.1376$, $k_D = 0.0051$, $Z = 2.59e+2$, $q = 1.60e-19$, $K = 1.38e-23$.

The three main points of this solution are:

- maximizing the performance by reducing the tracking error between the target power and the controlled one;
- constraining the temperature below a maximum value T_{CRIT}
- reduce the frequency of the desired core to the same value

The first two points are automatically satisfied by the MPC-based thermal solution. The aim of this solution is to account the last point.

5.3.1.1 Problem update

The simplest approach for modifying the problem structure and allows the controller to manage the communication between cores is to add a constraint as,

$$f_{C,}i(t) = f_{C,j}(t)$$

where $f_{C,i}(t)$ is the controlled frequency of the core i at the time instant t.

However, the manipulated variable of the control problem (5.14) is the controlled power consumption P_C that depends nonlinearly on frequency. Introducing such a constraint would mean to make the MPC problem nonlinear. Moreover, in order to have the possibility of constraining the frequency of all the N core we should have up to $N-1$ nonlinear constraints that greatly affect the computational complexity.

For all these reasons we decide to following another approach that keeps the linear characteristics of the problem.

Suppose, for simplicity, that we want to constraint only the frequencies of the core 1 and 2, $f_{C,1} = f_{C,2}$ (at the end of the section we will generalized the problem for all cores). The main idea is to impose that the difference between the power consumption $P_{C,1}$ and $P_{C,2}$ is equal to a Δ that depends on the workload (CPI) and the frequency. Indeed,

$$\begin{aligned} P_{C,1} - P_{C,2} =& k_{A1} \cdot \left(f_{C,1}^{k_B} - f_{C,2}^{k_B} \right) + k_C \cdot \left(CPI_1^{k_E} - CPI_2^{k_E} \right) + \\ &+ k_D \cdot \left(f_{C,1} \cdot CPI_1^{k_E} - f_{C,2} \cdot CPI_2^{k_E} \right) = \Delta \end{aligned} \tag{5.17}$$

where the static powers can be discarded due to their small and comparable values. Assuming at steady-state $f_{C,1} = f_{C,2} (= f_X)$ we can rewrite (5.17) as,

$$\begin{aligned} \Delta &= k_C \cdot \left(CPI_1^{k_E} - CPI_2^{k_E} \right) + k_D \cdot f_X \cdot \left(CPI_1^{k_E} - CPI_2^{k_E} \right) = \\ &= (k_C + k_D \cdot f_X) \cdot \left(CPI_1^{k_E} - CPI_2^{k_E} \right) \end{aligned} \tag{5.18}$$

Finally we can approximate the f_X as the rate,

$$\begin{gathered} (P_{MAX} - P_{MIN}) : (f_{MAX} - f_{MIN}) = (P_{C,2} - P_{MIN}) : (f_X - f_{MIN}) \\ f_X = (P_{C,2} - P_{MIN}) \cdot \tfrac{f_{MAX} - f_{MIN}}{P_{MAX} - P_{MIN}} + f_{MIN} \end{gathered} \tag{5.19}$$

where we called f_{MAX} and f_{MIN} the maximum and the minimum frequency respectively (e.g. 3000MHz and 1600MHz), and P_{MAX} and P_{MIN} the maximum and minimum power obtained from the Power Model (5.16), using for both the CPI_2.

Substituting the f_X in (5.18), we obtain,

$$\Delta = (k_C + k_D \cdot (P_{C,2} \cdot l - P_{MIN} \cdot l + f_{MIN}) \cdot \left(CPI_1^{k_E} - CPI_2^{k_E}\right) \tag{5.20}$$

where $l = \frac{f_{MAX} - f_{MIN}}{P_{MAX} - P_{MIN}}$. From (5.20) we can write,

$$P_{C,1} - P_{C,2} = \alpha(t) \cdot P_{C,2} + \beta(t) \tag{5.21}$$

where α and β depend on the CPI, and hence, on the time.

$$\begin{aligned} \alpha &= k_D \cdot l \cdot \left(CPI_1^{k_E} - CPI_2^{k_E}\right) \\ \beta &= (k_C - k_D \cdot P_{MIN} \cdot l + k_D \cdot f_{MIN}) \cdot \left(CPI_1^{k_E} - CPI_2^{k_E}\right) \end{aligned} \tag{5.22}$$

The equation (5.21) represents a constraint on power consumption in order to impose the same frequency to two cores. This constraint can be included in the objective function of the problem (5.14b) as,

$$min \sum_{k=0}^{h-1} \|P_T(t+k|t) - P_C(t+k|t)\|_{Q1}^2 + \|P_{C,1}(t+k|t) - P_{C,2}(t+k|t) - \alpha(t+k|t) \cdot P_{C,2}(t+k|t) - \beta(t+k|t)\|_{Q2}^2 \tag{5.23a}$$

$s.t.$

$$T_j(t+k+1|t) \leq T_{CRIT} \quad \forall j = 1, \ldots, n_T \quad \forall k = 0, \ldots, N \tag{5.23b}$$

where Q_1 is the diagonal weight matrix $Q_1 = diag(w_{1,1}, \ldots, w_{N,N})$ which relates the error between the power consumption assigned to the cores and the target one. $w_{i,j}$ is the scalar value representing the weight between the power error of the core i and the power error of the core j. For the sake of simplicity, we assumed $w_{1,1} = w_{2,2} = \cdots = w_{N,N}$. Similarly $Q_2 = r_{1,2}$ where $r_{1,2}$ is the weight value between the power of the core 1 and 2 in order to impose to them the same frequency

The problem (5.23) can be translated into a QP problem,

$$min_{\bar{P_C}} \frac{1}{2} \bar{P_C^T} \cdot H \cdot \bar{P_C} + g^T \cdot \bar{P_C} \tag{5.24a}$$

$s.t.$

$$M \cdot \bar{P_C} \leq b \tag{5.24b}$$

where $\bar{P_C} = [P_C(t|t)\, P_C(t+1|t)\, \ldots\, P_C(t+h-1|t)]^T$ and $P_C = [P_{C,1}\, \ldots\, P_{C,N}]^T$. Assuming $h_p = 1$ as in the rest of the thesis, then $\bar{P_C} = P_C(t|t)$ that is the vector of power consumption constantly assigned to the cores for the interval $[t, t+1]$.

Computing the products in the objective function (5.23a) we obtain,

$$\begin{aligned} & P_C^T\, Q_1\, P_C - P_T^T\, Q_1\, P_C - P_C^T\, Q_1\, P_T + P_T^T\, Q_1\, P_T + \\ & + P_{C,1}\, Q_2\, P_{C,1} - P_{C,1}\, Q_2\, (1+\alpha) P_{C,2} - P_{C,1}\, Q_2\, \beta + \\ & + (1+\alpha) P_{C,2}\, Q_2\, (1+\alpha) P_{C,2} - (1+\alpha) P_{C,2}\, Q_2\, P_{C,1} + (1+\alpha) P_{C,2}\, Q_2\, \beta + \\ & - \beta\, Q_2\, P_{C,1} + \beta\, Q_2\, (1+\alpha) P_{C,2}! + \beta\, Q_2\, \beta \end{aligned} \tag{5.25}$$

The terms $P_T^T\, Q_1\, P_T$ and $\beta\, Q_2\, \beta$ can be discarded since they are independent on the control variable P_C.

Comparing (5.24a) and (5.25) we obtain,

$$H = 2 \cdot \begin{bmatrix} w_{1,1} + r_{1,2} & -r_{1,2}(1+\alpha) & 0 & \ldots \\ -r_{1,2}(1+\alpha) & w_{2,2} + r_{1,2}(1+\alpha)^2 & 0 & \ldots \\ 0 & 0 & w_{3,3} & \ldots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \quad g = -2 \cdot \begin{bmatrix} P_{T,1} \cdot w_{1,1} + \beta \cdot r_{1,2} \\ P_{T,2} \cdot w_{2,2} + \beta \cdot r_{1,2} \cdot (1+\alpha) \\ P_{T,3} \cdot w_{3,3} \\ \vdots \end{bmatrix} \tag{5.26}$$

whereas the constraints matrices remains unchanged

$$M = \begin{bmatrix} C \cdot B_1 \\ -I_N \end{bmatrix} \quad b = \begin{bmatrix} T_{CRIT} - C \cdot A \cdot x(t) + C \cdot B_2 \cdot T_{AMB}(t) \\ [-P_{MIN}]_{N \times 1} \end{bmatrix} \tag{5.27}$$

where A, $B = [B1_{2N \times N}\ B2_{2N \times 1}]$, and C are the matrices of the model (5.15). Moreover we added a constraint to impose that cores power is not lower than P_{MIN}. It is worth to note that the matrices H, g, b depend on time (due to the CPI, x and P_T), therefore every time sample they must be updated.

We conclude this section generalizing the approach. It is enough to see that enabling a constraint between the frequency of two cores i and j means to modify the matrices $H = 2 \cdot Q_1$ and $g = -2 \cdot [P_{T,1} \cdot w_{1,1}, \ldots, P_{T,N} \cdot w_{N,N}]^T$. More in details we must add the value:

- $r_{i,j}$ in $H_{i,i}$,
- $-r_{i,j}(1+\alpha)$ in $H_{i,j}$ and $H_{j,i}$,
- $-r_{i,j}(1+\alpha)^2$ in $H_{j,j}$
- $\beta \cdot r_{i,j}$ in $g(i)$,
- $\beta \cdot r_{i,j} \cdot (1+\alpha)$ in $g(j)$,

5.3.2 The Implementation

The pseudo-code below shows the sequence of operations necessary to make the proposed solution work.

Pseudo Code

```
1  OFF-LINE MODEL IDENTIFICATION:
2    Apply a PRBS task sequence;
3       get P_i, T_i, T_NEIGH_i;
4       obtain α, and β with the distributedARX approach;
5       find the global state space representation A, B, C with C = [I_N 0];
6
7  CONTROLLER ROUTINE
8    Initialize the weight matrices M_i, and the state x̂_i(0);
9    FOR EACH SAMPLE
10           get f_T(t) and CPI(t);
11           convert to P_T(t) using the Power Model;
12           update H, g, and b;
13           solve the QP problem and find P_C(t);
14           compute f_C(t);
15           estimate x̂(t);
16   END_FOR
```

More in detail:

Line 1 during the system initialization phase we execute the model identification;

Line 2-3 we apply a pseudo-random task sequence to each core and we collect the core power and local neighborhood temperature traces;

Line 4 we execute the *ARX* optimization in each core to obtain the α and β parameters;

Line 5 we convert the single-core models in the state-space form using the observability matrix to give physical meaning to the first component of the state vector (it correspond to the measured core temperature) and then we compose all of them together in order to obtain a global model;

Line 8 we define the constant matrix used in the QP problem (5.24), M, and assign the initial state, $\hat{x}(0)$, to the model ;

Line 9 at each time step the loop from line 9 to line 16 is repeated;

Line 10 the High Level SoC Manager provides the desired core frequencies $f_T(t)$ and the predicted workload $CPI)(t)$;

Line 11 the Centralized Thermal Controller converts them into the target power $P_T(t)$ using the nonlinear function (5.16);

Line 12 the matrix H, dependent on $CPI(t)$, the vector g, dependent on $CPI(t)$ and $P_T(t)$, and the vector b, dependent on $\hat{x}(t)$, that is estimated with a Luenberger observer, $T_{AMB}(t)$ and $T_{NEIGH_i}(t)$, are updated.

Line 13 the solver finds the optimal $P_C(t)$ solution of the QP problem.

Line 15 P_C and workload are used to find f_C inverting the function (5.16) as an example with the Brent's algorithm.

Line 16 the observer estimates the state $\hat{x}(t)$ knowing the $f_C(t)$ and $T(t)$. This state will be used as initial state by the MPC controller to estimate the future temperature of the core.

5.3.3 Experimental Results

We have tested the performance of the centralized communication-aware solutions using the 8 cores plant shown in the previous solution (refer to Fig. 5.6). The target frequencies range from 1600MHz to 3000MHz, whereas the CPIs range from 0.5 to 100. The caches consumes the 30% of the adjacent core powers.

As a first test we ran a PRBS target frequency on all the cores and we assigned to CPI a constant value of 1.5. Only the CPI of the core 2 varies as a PRBS. For the first 10s core 1 and 2 communicates, later communication is between core 1 and 5. Fig. 5.11 shows the results obtained from the test. The temperature is perfectly bounded even if the prediction model has a low order compared to the plant and the power consumption of the caches are not measured. However, we have to remark that we considered the target CPI and power perfectly known. The frequency effectively assigned to the core 1 and 2, $f_{C,1}$ and $f_{C,2}$, result the same with an error of about the 1%. The Fig. 5.11 also shows that the solution is able to dynamically change the communication constraints at run-time. Indeed, the frequencies of cores 1 and 2 are the same until 10s, then the frequencies of cores 1 and 5 equalized, while the frequency of core 2 is the one that minimizes the power error between P_C and P_T.

A second test measure the complexity of the solution. We have translated the code of the control algorithm from Matlab language to C++ language and then we have run some PARSEC 2.1 benchmarks measuring the time necessary for solving the QP problem[1]. The results for the Fluidanimate benchmark are shown in Fig. 5.12. The mean time to solve the problem is 37us.

[1] We ran this test on a simple four cores processor.

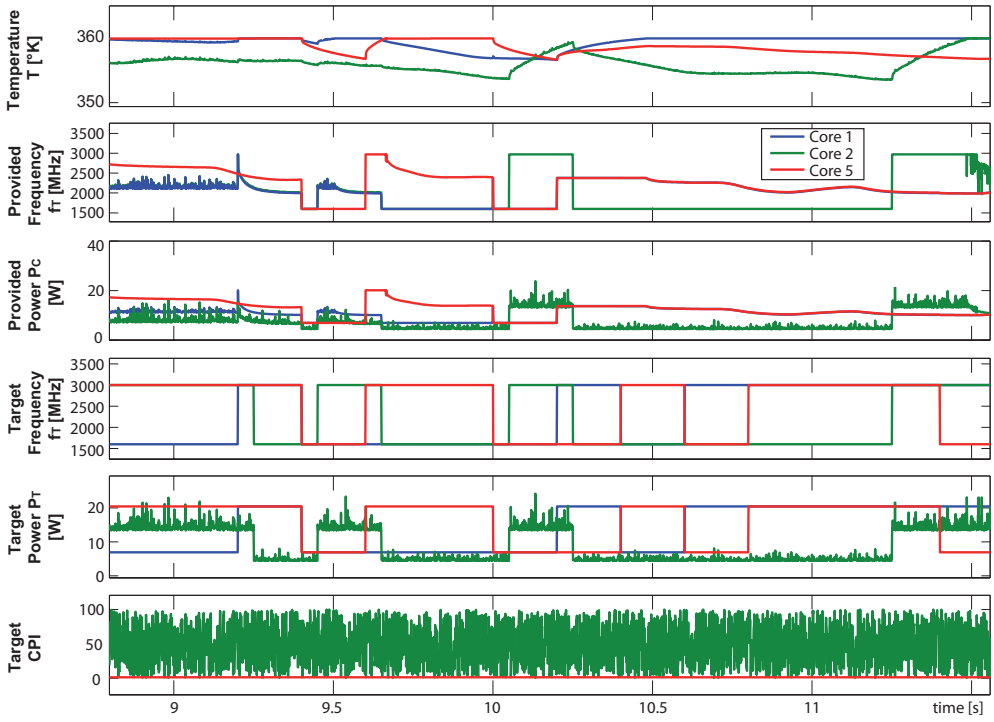


Figure 5.11: Temperature, Frequency and Power results of cores 1, 2, and 5. Before 10s the $f_{C,1} = f_{C,2}$, then $f_{C,1} = f_{C,5}$.

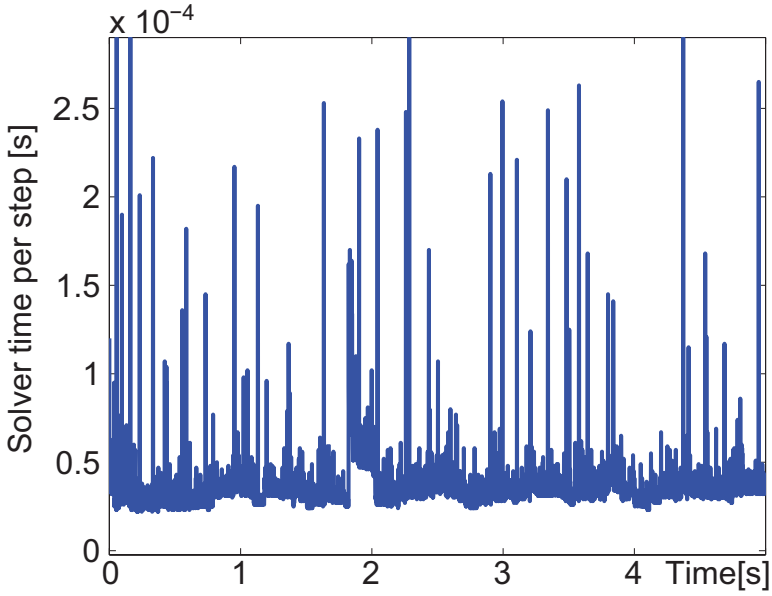


Figure 5.12: Time spent for solving the QP problem at each time step.

Bibliography

[1] J. Howard et al., *A 48-core ia-32 processor in 45 nm cmos using ondie message-passing and dvfs for performance and power scaling*, IEEE Journal of Solid-State Circuits, Vol. 46(1):173183, Jan. 2011. 136

[2] G. Dhiman, T. S. Rosing, *Dynamic voltage frequency scaling for multi-tasking systems using online learning*, ISLPED, Portland, OR, USA, Aug, 27-29, 2007. ix, 135, 136

[3] G. Keramidas, V. Spiliopoulos, and S. Kaxiras, *Interval-based models for run-time dvfs orchestration in superscalar processors*, ACM CF '10, pp. 287-296, New York, NY, USA, 2010.

[4] S. Rusu, S. Tam, H. Muljono, J. Stinson, D. Ayers, J. Chang, R. Varada, M. Ratta and S. Vora, *A 45 nm 8-Core Enterprise Xeon®Processor*, IEEE Journal of solid-state circuits, Vol. 45, pp. 7-14, Jan. 2010. 151

[5] A. Bartolini, M. Sadri, J. Furst, A.K. Coskun, L. Benini, *Quantifying the impact of frequency scaling on the energy efficiency of the single-chip cloud computer*, Design, Automation Test in Europe Conference Exhibition (DATE), pp.181-186, Mar. 2012. 154

[6] N,Ioannou, M. Kauschke, M. Gries, M. Cintra, *Phase-Based Application-Driven Hierarchical Power Management on the Single-chip Cloud Computer*, International Conference on Parallel Architectures and Compilation Techniques (PACT),pp. 131-142, Oct. 2011 154

[7] Martin Milo M. K. et al. *Multifacets general execution-driven multiprocessor simulator (GEMS)*, toolset. SIGARCH Comput. Archit. News, 33(4):92-99, 2005. 140

[8] G. Paci, M. Morari, V. Dua and E.N. Pistikopoulos, *Exploring temperature-aware design in low-power MPSoCs*, DATE, Vol.1, 2006, pp.1–6. 142

[9] C. Bienia, S. Kumar, J.P. Singh, K. Li, *The PARSEC Benchmark Suite: Characterization and Architectural Implications*, PACT, 2008. 140, 148

BIBLIOGRAPHY

Chapter 6

Guaranteed Re-sprinting in MPSoCs exploiting MPC

In this chapter a novel control solution is presented in order to address the "Utilization Wall" issue in mobile devices. The solution mitigates the problem with a computational sprinting approach. A Phase Change Material package enables higher performance, and a two-layer predictive control enables thermally-safe sprinting while guaranteeing a predictable re-sprinting rate.

6.1 Overview

The control solution that we present in this chapter addresses the "Utilization Wall" issue introduced in the Chapter 2. Although conceptually we could describe this solution in the previous chapter (it presents the distributed MPC-based thermal controller as basic element), we prefer to reserved an entire chapter to this solution due to the extension of the dissertation and in order to highlight the novelty of the contribution.

We have already discussed about the issues introduced by the "race" for improving CPU performance. The growing transistor counts, the limited power budgets, the breakdown of voltage scaling and the difficulties in heat dissipation prevent the possibility of run all cores without getting into a thermal crisis, limiting de facto the number of usable cores in a device. In this power-limited computing era the parallelism has shown its benefits on performance, but it is now accepted that in upcoming and future technology generations (22nm and beyond) all the units on a die cannot be continuously switched on at the same time, as their total power

consumption would exceed the maximum Thermal Design Power (TDP), leading to a thermal run-out. Cooling solutions, needed to remove the heat from the silicon die, are limited by cost or by physical constraints: in data centers approximately 50% of the energy consumed is used for powering the cooling infrastructure (1), whereas in embedded/mobile platforms the cooling system is constrained by form factor and packaging cost (2). Dark Silicon has been estimated to be the 21% at 22nm and 50% at 8nm (3): practically limiting the maximum parallelism achievable by future many-cores. In future mobile platforms this problem is even more serious since active cooling cannot be easily implemented without compromising battery lifetime, form factor, user experience (4).

Nevertheless, mobile devices have characteristics that bind well with computational sprinting approaches – i.e. approaches where all cores are powered on at the maximum frequency for short time windows, ideally triggered on-demand. Indeed, the applications that run on power-bound systems are typically composed of alternating sequences of parallel and sequential sections (5) and thus they do not exploit constantly the underling hardware parallelism (parallel tasks are also characterized by different computational phases (6)). This is exacerbated in mobile platform where applications are often triggered by the users and differently from batch jobs, their Quality of Service (QoS) does not depend on average throughput, but on users experience and response time. Studies demonstrate that an average user perceives a response time below 150ms as crisp, noticeable within 1s, annoying in 2s and not acceptable in the 2-5s range (7). Moreover, accordingly to the usage scenario, the user needs to share its cognitive resources with the perception of the external environment. The minimum continuous span of attention to the mobile device is 4s in a busy street whereas the maximum switch-back duration (the time spent on attending the environment before switching back to the mobile device) is 7-8s (8). As a consequence, mobile platform are often response-time constrained and they need to provide fast bursts of computation on demand. Therefore, the seldom use of parallelism and the importance of responsiveness for the Quality of Service perceived by the user, rather than average throughput, enable the possibility of turning on at the same time all cores and exceed temporarily the thermal power budget to provide instantaneous throughput, after which the chip must return to nominal operation to cool down.

The duration of the time windows in which all cores can run at the maximum speed depends on the thermal capacitance of the chip. Traditional Thermal Design Power is defined statically on the worst-case power consumption, considering only the package thermal resistance (9). As

consequence, packages are optimized for minimizing their resistance with the ambient temperature neglecting the heat storage capabilities of the different package materials, associated with their thermal capacitance. In a regime where classical TDP cannot ensure the power-on of all the integrated cores, but maximum performance is needed mainly in short bursts, the heat buffer associated with the thermal capacitance can be sufficient to run all the cores at the maximum performance for short time windows, ideally triggered on-demand whenever peak parallel workload phases and user interaction occur.

These Computational Sprints (10) can be lengthened by increasing the thermal capacitance of the package. Phase Change Material that are solid at ambient temperature, can store extra heat during the melting process, releasing it to the ambient later on, during solidification. As consequence of that, PCM allows packing in a small volume and within a small temperature gap a large thermal capacitance placed close to the silicon die (10). Embedding the correct PCM quantity and material can enable longer sprints (e.g. 1s), suitable for periodically speed-ups and can improve the QoS of interactive tasks.

From the above considerations, clear advantages can be achieved by exploiting silicon/-package thermal capacitance, PCM or other materials as "heat tanks" to be filled along sprinting phases. However, these tanks are finite, then only limited sprinting intervals are sustainable. Suitable "rest intervals" are, then needed to let the tanks to release heat in the external ambient. This is necessary to keep cores temperatures below their critical values and to restore sprinting capabilities. Such intrinsically-dynamic thermal behavior requires a suitable run-time temperature management to guarantee safe working, even under variable and possibly uncertain conditions.

Once a sprinting architecture has been defined with a reliable and effective thermal control, one has to cope with another crucial challenge, not yet deeply considered in literature. That is *how to exploit limited sprinting capabilities, when different tasks are running together with different QoS requests or criticality features*. As a matter of facts, rest intervals are a sort of blanking periods w.r.t. possible sprinting requests, then their durations and placement along time will affect the actual QoS of different applications. In the mobile domain, different APPs/tasks can be executed at the same time with different QoS requirements (e.g. video encoding/decoding (11), driving augmented reality(12) and health monitoring (13), phone call and text message). Moreover, in other domains as automotive embedded control, mixed criticality scenarios will take place with hard real-time and even safety-critical tasks running along with soft real-time applications. A suitable run-time control policy is clearly needed to tackle

such heterogeneous workload scenarios. Sprinting capabilities have to be exploited or reserved to favor, or even guarantee, the execution and the QoS of most critical tasks. Summing up, sprinting time is a shared, limited and dynamical resource which needs dynamical restore time. Then, whenever a sprint is launched, the system should preserve a suitable "room" for subsequent sprints, according to a policy defined for managing different task criticalities. We refer such a feature as *Re-sprinting* management.

Our control solution guarantees thermally-safe and run-time optimal sprinting and re-sprinting using a two-layer MPC-based solution.

6.2 Sprinting Architecture

Contrary to the other solutions presented in this thesis, before introducing the control architecture, we need to:

- describe the characteristics of the chip on which we have implemented our control algorithm;
- show how to model the thermal behavior of this platform for studying the effectiveness of our solution;
- define concepts useful for better understand the rest of the work.

6.2.1 Platform Characteristics

We considered a multi-core processor with 16 cores, the die area is $6.8mm \times 6.8mm$ with a thickness of $350\mu m$. Per-core DVFS is assumed and the maximum power dissipated by each core under maximum frequency and worst load conditions is set to $P_{c,max}$=$1W$ (i.e. the chip maximum dissipated power is P_{max}=$16W$). Maximum frequency, corresponding to maximum power is set to 1.5Ghz and minimum one is set at 500Mhz, with a per core power equal to $P_{c,min}$=$150mW$ under worst load conditions.
In sprinting conditions, all the cores are requested to work with maximum frequency and full utilization. In rest conditions, only one core is assumed to run at full frequency and utilization (i.e. at full power), while all of the others are assumed to be in idle status with an equivalent power of $P_{c,idle}$=$50mW$ each (i.e. a chip power of P_{rest}=$1.75W$ is assumed). This characterization of computational/power traces is simplified, but significant to represent relevant thermal

issues. Indeed, actual sprinting traces are expressed in terms of high frequency (and voltage) pulses, while the actual power consumption also depends on current workload characteristics. Nevertheless, in this work, to simplify the discussion, sprinting traces will be represented in terms of power (P_{max} along sprints and P_{rest} in rest conditions), assuming worst workload conditions. This does not give a relevant generality loss, when focusing on thermal issues.
Thermal stability is obviously guaranteed in rest condition, while the system cannot sustain permanent sprinting; the maximum admissible Silicon temperature is T_{max}=360°K, while the considered maximum ambient temperature is T_{AMBmax}=318°K. As reported in Fig. 6.1a a layer

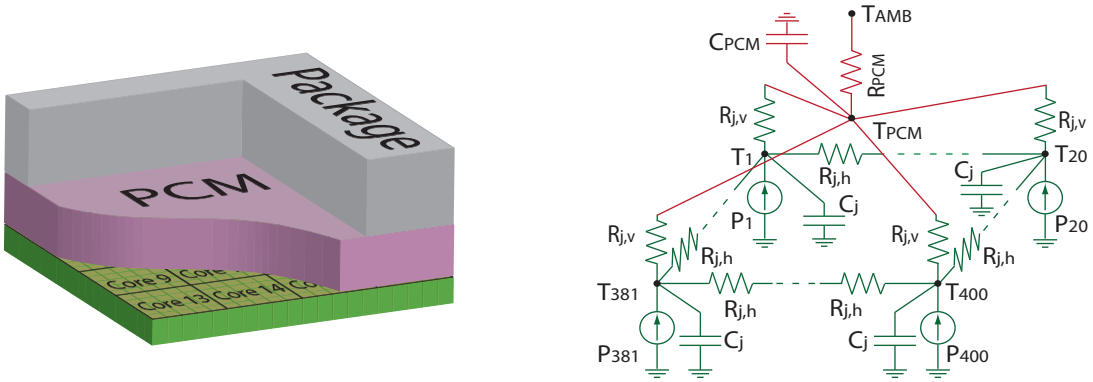


Figure 6.1: The considered sprinting architecture and the adopted thermal modelling

of PCM with physical properties similar to commercial Climsel-C70 (14) is interposed in between the Silicon die and the device case. The thickness of such layer is 410μm. According to common mobile and embedded applications, no heat-sink is considered (2). A Copper based thermal conductivity enhancer is assumed inside the PCM layer to improve conductivity and speed up heat charging and discharging during the melting phase. This mixture can be seen as an homogeneous material with both high thermal conductivity and high heat capacity (10).

The only measurements available for control purpose come from the temperature sensors of the cores, the PCM layer and the ambient. Thei readings are $T_{c,i}$, with i=1..16, T_{PCM} and T_{AMB}, respectively.

6.2.2 Thermal Modeling (Simulator)

The chip described in the previous section has been modeled using the same finite elements technique used for the thermal simulators shown in the Appendix B. A lumped thermal model is obtained according to the equivalent electric network reported on the Fig. 6.1b. Also in this

case we considered two layers one of silicon and the other of a mixture of PCM and copper (the enhancer). The silicon layer has been split in cells, each one modeled by a current generator representing the power dissipated by the cell, a resistor and a capacity (each core has 25 cells). As in previous work (10), the PCM layer and conductivity enhancer have been modeled by a single large cell assuming a uniform distribution of the heat. The resulting model is,

$$\begin{aligned} \dot{T}_k &= \frac{P_k}{C_j} + \frac{T_{PCM} - T_k}{C_j \cdot R_{j,v}} + \sum_{i=1}^{\#neighbors} \frac{T_{neigh,i} - T_k}{2 \cdot C_j \cdot R_{j,h}} \\ \dot{U} &= \sum_{k=1}^{16} \frac{T_k - T_{PCM}}{R_{j,v}} - \frac{T_{PCM} - T_{AMB}}{R_{PCM}} \end{aligned} \tag{6.1}$$

where T_k, P_k are respectively the temperature and the power dissipated by the k-th Silicon cell, $T_{neigh,i}$ are the neighbor cells temperatures, T_{PCM} is the temperature of the PCM, T_{AMB} is the ambient temperature and U is the internal energy of the PCM layer.

It is worth noting that, in (6.1), the first equation is obtained by standard space-discretization of the well-known Heat Equation. In contrast, for the second equation, according to (15), an energy-based model has been exploited for PCM to handle its nonlinear phase-changing behavior. The PCM temperature can be easily derived from U by a nonlinear map, $T_{PCM} = \mathcal{F}_{PCM}(U)$, to represent monotonic increasing behavior in solid and liquid condition, while constant temperature will be given in the melting phase, see Fig. 6.2a. The first term in the top equation represents the contribution of the power consumption in each cell, whereas the last two, according to Fourier Law, are the effects of heat flow entering in the cell k from PCM and neighbor cells, respectively. Similarly, in the bottom equation, the first term is the heat flow from all of the silicon cells to PCM, whereas the second gives the heat flow from PCM to the external environment; no direct thermal path is assumed between the silicon cells and the ambient.

The following values for cell capacitance and thermal resistances are assumed, $C_j = 6.6e-5 J/^\circ K$, $R_{j,h} = 22.9^\circ K/W$, $R_{j,v} = 215^\circ K/W$ and $R_{PCM} = 7.9^\circ K/W$. The latter two resistances, linking the PCM layer to the others elements, benefit from the Copper conductivity enhancer, in particular in $R_{j,v}$ contact resistance between Silicon and PCM layer has been considered. Parameters of the adopted PCM are melting temperature, $T_{melt} = 70\,^\circ C$, density, $1700 Kg/m^3$, specific latent heat, $396 KJ/Kg$, latent heat of the whole volume $12.8 J$ and differential thermal capacitance in solid and liquid condition, $3.52 J/(^\circ K\, \mu m^3)$. These parameters will characterize the whole "PCM + conductivity enhancer layer", since Copper thermal capacitance does not affect relevantly the PCM thermal inertia. All these leads to a maximum internal energy for

solid phase $U_{meltmin} = 23J$ and a minimum internal energy for liquid phase $U_{meltmax} = 35.8J$, in between melting phase will take place.

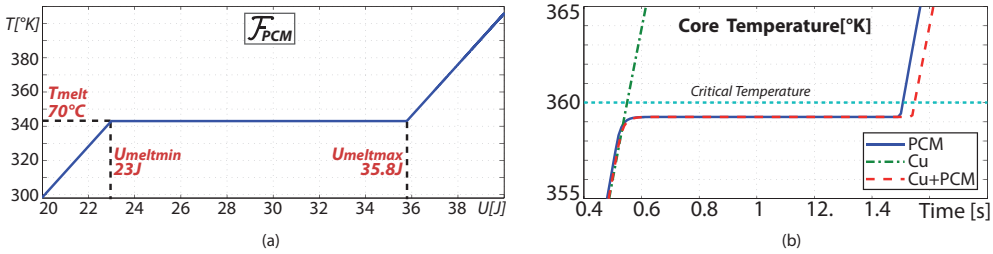


Figure 6.2: (a) Internal Energy to Temperature nonlinear function; (b)Comparison among sprinting architectures

In Fig. 6.2b the effectiveness of the PCM with conductivity enhancer is highlighted. A constant full power working condition is considered and the adopted device is compared with two variants where the PCM + enhancer is replaced by a 2*mm* thick heat spreader or by a pure PCM layer, respectively. By observing the time when core temperatures reach T_{max}, it can be noted that the considered architecture enables larger sprinting room w.r.t. the others.

6.2.3 Guaranteed re-sprinting definition

In this Subsection, we provide and motivate a "formal" definition of Guaranteed Re-sprinting capability, then we translate it in a clear requirement for PCM energy management by exploiting a simplified, but effective, thermal model of the given system.

We referred to the term Re-sprinting management as the policy to handle heterogeneous tasks with different QoS requests or criticality levels. In particular, limited sprinting capability have to be spent or preserved according to a policy which favors, or even guarantees, most critical or QoS-demanding task space. Toward this goal, we consider the following scenario. We suppose to have two main task groups, both requiring full-power sprinting, critical hard real-time periodic and predictable tasks and non-critical aperiodic tasks. For the first group, the total fulfillment of the sprinting requests has to be guaranteed, while, for the second one, best effort is admissible; we refer those groups as *Guaranteed Group* and *Best Effort Group*, respectively. For the Guaranteed Group a known periodicity of M is assumed, while the total time needed to execute all these tasks together at full power is assumed equal or lower than a given bound equal to N. This kind of information is usually available when dealing with real-time critical tasks. Differently, for the Best Effort Group no information is assumed.

Under the considered condition, it is clear that the sprinting and rest intervals need to be dynamically arranged not only to maximize service of generic sprinting providing thermal capping, but N-long sprinting windows must be guaranteed every M seconds to serve the Guaranteed Group. We formalize such requirement in the following definition of N-M Guaranteed Re-sprinting.

N-M Guaranteed Re-sprinting for a computing system is the capability of guaranteeing a sprint at full power of duration N, as soon as a time M ($M > N$) has elapsed from the starting of the previous guaranteed sprinting.

From the above definition, it is clear that whenever a sprint has been launched, the run-time control system has to limit the sprint period not only taking into account maximum temperature bounds, but also considering that the system needs some rest time to cool down properly and get ready for a possible new sprinting request of N seconds, after M seconds from the beginning of the previous one. Such behavior requires a suitable sizing and management of the PCM heat tank which provides room for sprinting according to its "charge level". In the following an explicit and treatable relation is determined to link PCM energy condition with re-sprinting requirements. Toward this goal, we considered a simplified, but effective, thermal model by collapsing all the Silicon cells into a single cell with temperature T_{Si}. Summing up all the cells capacitors and power sources (neglecting horizontal resistances $R_{j,h}$), and by parallel compositions of all the $R_{j,v}$, we can define the total approximated Silicon capacitance, C_{Si}=$26mJ/^{\circ}K$, and the total approximated Silicon-to-PCM thermal resistance, $R_{Si-PCM} = 0.6^{\circ}K/W$. Then the following approximated model can be drawn.

$$\begin{aligned} \dot{T}_{Si} &= \frac{P_{tot}}{C_{Si}} - \frac{T_{Si} - T_{PCM}}{C_{Si} \cdot R_{Si-PCM}} \\ \dot{U} &= \frac{T_{Si} - T_{PCM}}{R_{Si-PCM}} - \frac{T_{PCM} - T_{AMB}}{R_{PCM}} \end{aligned} \tag{6.2}$$

where $P_{tot} = \sum P_k$. According to the commonly used sprinting time-scale, usually in the range of 1-10s, the effect of the Silicon inertia can be neglected with respect to PCM and enhancer dynamics (i.e. $\dot{T}_{Si} = 0$). Therefore, a static relation between T_{Si} and P_{tot} can be exploited to revise (6.2) leading to

$$\begin{aligned} T_{Si} &= T_{PCM} + R_{Si-PCM} \cdot P_{tot} \\ \dot{U} &= P_{tot} - \frac{T_{PCM} - T_{AMB}}{R_{PCM}} \end{aligned} \tag{6.3}$$

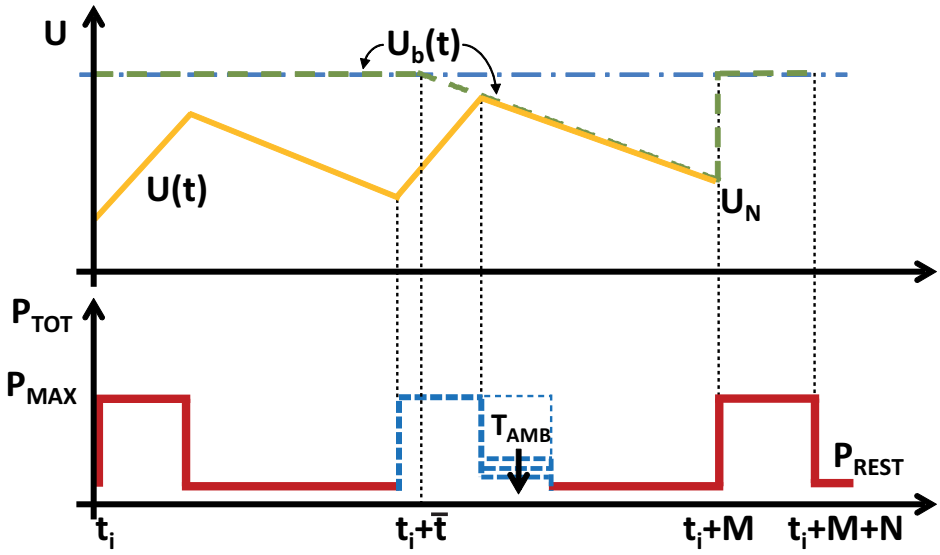


Figure 6.3: Translation of the *N-M* Guaranteed Re-sprinting objective in a time-varying constraint on PCM internal energy, U

With this model at hand, the re-sprinting requirement can be easily translated in a time-varying constraints on the PCM internal energy U according to the time diagram of Fig. 6.3. Let assume a guaranteed sprinting has been issued at time t_i, then according to the prescribed *N-M* Guaranteed Re-sprinting between time t_i+M and time t_i+M+N the PCM is expected to be able to store $P_{max}\cdot N$ energy without violating Silicon temperature bounds. That means $U(t_i+M+N)\leq U_{MAX}$, where U_{MAX} can be easily defined, using (6.3), as

$$U_{MAX} = max\{U \,|\, \mathcal{F}_{PCM}(U) = T_{max} - R_{Si-PCM}\cdot P_{max}\} \tag{6.4}$$

In the considered 16-cores case, we have $T_{melt} < T_{max} - P_{max}\cdot R_{Si-PCM}$. That means we can run a sprinting up to a $T_{PCM} > T_{melt}$ and a corresponding $U > U_{meltmax}$, where PCM is totally liquid. Nevertheless, we simply assume $U_{MAX} = U_{meltmax}$, this saves some margin and mitigates thermal cycles caused by the on-off computational paradigm. Having defined U_{MAX}, we can compute U_N reported in Fig. 6.3 as follows,

$$U_N = U_{MAX} - \left(P_{max} - \frac{T_{PCM} - T_{AMBmax}}{R_{PCM}}\right)\cdot N \tag{6.5}$$

U_N in (6.5) is the maximum admissible energy at time t_i+M which guarantees that a sprint at full power can be sustained in the following N seconds. Therefore, whatever the previous sprint requests are, N-M Guaranteed Re-sprinting asks for an $U(t_i+M)\leq U_N$. It is worth noting

that, in (6.5), U_N derivation is carried out assuming that PCM melting condition is preserved, i.e. $U_N > U_{meltmin}$ but can be easily extended to more general condition. Now, moving back along time from (t_i+M, U_N), we can derive the time-varying $U_b(t)$ taking into account the maximum cooling capabilities of chip under $T_{AMB}=T_{AMBmax}$. These are achieved when all the cores but one are in idle conditions (i.e. the system is in a rest phase). Hence, in a generic time instant $t < t_i+M$, the maximum admissible internal energy $U_b(t)$ is the maximum one which can be steered to (t_i+M, U_N) exploiting the maximum cooling capability of the system. At the same time, the bound $U<U_{MAX}$ has to be considered, then $U_b(t)$ can be computed as follows $\forall t \in [t_i, t_i+M[$, leading to the line reported in Fig. 6.3.

$$U_b(t) = min\left\{U_{MAX}, U_N+\left(\frac{T_{PCM}-T_{AMBmax}}{R_{PCM}}-P_{rest}\right)\cdot((t_i+M)-t)\right\} \tag{6.6}$$

where obviously $\frac{T_{PCM}-T_{AMBmax}}{R_{PCM}}-P_{rest}>0$, otherwise the system is not thermally balanced (i.e. is not sized correctly). In addition, the instant $\bar{t}$ where $U_N+\left(\frac{T_{PCM}-T_{AMBmax}}{R_{PCM}}-P_{rest}\right)\cdot((t_i+M)-\bar{t})=U_{MAX}$ has to be greater than t_i+N, otherwise the N-M Guaranteed Re-sprinting is not sustainable by the system owing to its physical properties (i.e. a suitable resizing is needed).
Finally, summing up all of the previous considerations, the N-M Guaranteed Re-sprinting request can be effectively translated into a time-varying bound on the internal energy U of the PCM by using, in each M interval, the $U_b(\cdot)$ profile derived in (6.6).

In Fig. 6.3 a typical P_{tot} profile is also reported to highlight sprinting from both Guaranteed (continuous line) and Best Effort Groups (dashed line). It is worth noting as the sprinting for the Best Effort Group will be affected by the time varying bound $U_b(\cdot)$. In addition, it is possible to figure out that, whenever the bound is reached, the control system will enforce a power consumption that makes the PCM internal energy U to slide along $U_b(\cdot)$. The power giving such a behavior is the rest power with $T_{AMB} = T_{AMBmax}$, according to (6.6), but it can be larger whenever $T_{AMB} < T_{AMBmax}$. This degree of freedom can be effectively used to maximize the integral of the sprinting power.

For the considered sprinting architecture, a 0.2-4s *Guaranteed Re-sprinting* is requested. By applying (6.5) and (6.6) it can be noted that this kind of Re-sprinting is effectively sustainable with $U_{MAX} = U_{meltmax}$, $U_N = 33.2$ and $\bar{t} = t_i + 2.23s$. In addition, it can be verified that, according to the system sizing, this re-sprinting capability requirement for Guaranteed Group leaves a significant room for Best Effort Group. Detailed computations for such issue, and sizing procedure in general, are not reported here since this is not the main focus of this

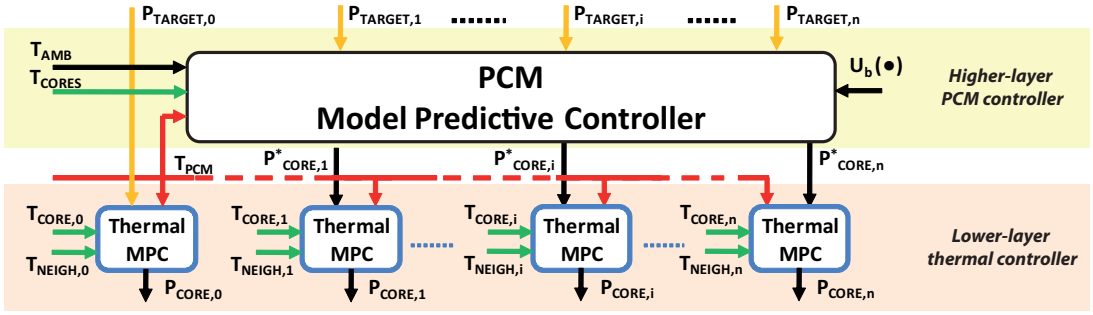


Figure 6.4: Structure of the proposed controller

work (nevertheless, some hints on such topic have been already depicted while discussing the feasibility issues in (6.5) and (6.6)).

6.3 Architecture

The proposed control solution is a novel closed-loop controller based on MPC approach that provides effective and reliable thermal capping and achieves optimal management of various and dynamics sprinting scenarios. Workload with mixed-criticality heterogeneous tasks are considered, then Guaranteed Re-sprinting requirements, are directly tackled. In addition, the proposed solution is designed to handle as well situations where just best effort approach is needed, with no re-sprinting guarantees.

The Fig. 6.4 depicts the block diagram of the proposed solution. We used a two-layer hierarchical approach:

Lower-layer thermal controller : it manages optimization of the current core computation powers and effective capping of core temperatures, taking advantage of the distributed and scalable MPC solution presented in Chapter 4. Notice that this layer is enough for manage a sprinting approach that maximizes performance.

Higher-layer PCM controller : with its novel centralized MPC solution, it manages the heat buffer (PCM, in this paper) maximizing sprinting time, while guaranteeing re-sprinting performance for the considered system.

The higher level interacts with the lower one by tuning the computational power requested for sprinting. When no guaranteed re-sprinting is needed, the proposed system can be easily "downgraded" to such simpler condition by just turning off the higher layer of the pro-

posed controller. This can be useful when a generic task-set have to be served according to a best effort approach. It is worth noting that this transition between "guaranteed" and "non-guaranteed" re-sprinting, and even the opposite, can be handled dynamically at run-time.

6.3.1 The Lower-layer thermal controller

In the lower layer we exploited the distributed MPC-based Thermal scalable solution presented in Chapter 4. For each i-th core the following discrete-time local MPC problem is set,

$$\begin{aligned} &\min_{P_{c,i}} \left(P_{c,i}(t|t) - P^*_{c,i}(t)\right)^2 \\ &\text{subject to } T_{c,i}(t+1|t) \leq T_{max} \end{aligned} \tag{6.7}$$

$P^*_{c,i}$ are the core power references. Their values for $i = 2,\ldots,16$ are determined by the higher level controller, while for the core 1, $P^*_{c,1} = P_{c,max}$ at any time, according to sprinting architecture previously described in Section 6.2. $P_{c,i}$, are the actual core powers (i.e. the sum of the powers dissipated in cores areas).

We assume to use them as control knobs, although actually only frequency and voltage are directly controllable. We imagine as in previous solution the presence of a Power Model to convert frequency and workload (obtained by performance counters readings) to power. T_{max} is the maximum admissible temperature for the cores as defined in Section 6.2. The general idea driving such control approach is to keep the power of each core as close as possible to the requested one, complying with hard thermal bounds. As long as no thermal issue occurs, the power will be equal to the requested one, otherwise temperature limitation will be enforced with minimum performance penalty. This will provide optimal power performance combined with reliable temperature capping. The distributed setting of such MPC problem allows one to obtain linear complexity w.r.t. the number of cores and to split the implementation on all of the cores (see Chapter 4).

Differently from the other solutions treated in the previous chapter, the discrete-time single-core model adopted for predicting the future temperature of the i-th core is represented by a first order equation,

$$T_{c,i}(t+1|t) = T_{c,i}(t) + \tau_{sT}\left(\frac{P_{c,i}(t)}{C_c} + \frac{T_{PCM}(t) - T_{c,i}(t)}{C_c \cdot R_{c,v}} + \sum_{i=1}^{\#neigh} \frac{T_{c\,neigh,i}(t) - T_{c,i}(t)}{2 \cdot C_c \cdot R_{c,o}}\right) \tag{6.8}$$

where τ_{sT} is the sampling time, $T_{c\,neigh,i}$ are the temperatures of the i-th core neighbors, C_c= $1.65 mJ/^\circ K$, $R_{c,o}$=$4.6^\circ K/W$, $R_{c,v}$=$9.6^\circ K/W$ are given by straightforward parallel composition

of capacitances and resistances of the cells belonging to a core. Note that in this single-core model the interaction with the reminder of the chip is taken into account by means of the neighbor cores and PCM temperatures, acting as uncontrollable but measurable inputs. An additional positive effect of such feature is that the model (6.8) is Linear Time-Invariant, despite of the large nonlinearity characterizing PCM. Thanks to the use of the simplified, but effective, first-order model (6.8), the MPC problems (6.7) can be solved explicitly, leading to the following algorithm for the thermal controller of each core,

$$P_{c,i}(t) = \begin{cases} P^*_{c,i}(t) & if\ P^*_{c,i}(t) \leq \bar{P}_{c,i}(t) \\ \bar{P}_{c,i}(t) & if\ P^*_{c,i}(t) > \bar{P}_{c,i}(t) \end{cases} \tag{6.9}$$

where

$$\bar{P}_{c,i}(t) = \frac{C_c}{\tau_{sT}} \left(\bar{T}_{max} - T_{c,i}(t) - \tau_{sT} \left(\frac{T_{PCM}(t) - T_{c,i}(t)}{C_c \cdot R_{c,v}} + \sum_{i=1}^{\#neigh} \frac{T_{c\,neigh,i}(t) - T_{c,i}(t)}{2 \cdot C_c \cdot R_{c,o}} \right) \right) \tag{6.10}$$

$\bar{P}_{c,i}(t)$ in (6.10) represents the maximum sustainable power at the instant t preventing violation of the thermal bound T_{max} at $t+1$. This expression is derived from the simplified model (6.8), taking into account the current temperature sensor readings. $\bar{T}_{max}$, lower than T_{max}, is adopted in (6.10) to save some margin and take into account model approximations and parameters uncertainty. A reliable $\bar{T}_{max}$ can be derived empirically with simulation tests or formally with more complex computations. As an example we could find it with the same techniques used to find τ_{MPC} margin in Section 5.2.1.3.

In the considered benchmark, we have $\bar{T}_{max} = 359.9°K$, while $T_{max} = 360°K$. This testifies the good approximation given by simplified models (6.8), when no parameter uncertainties are accounted. It is worth underlining that, in order to improve accuracy, we could replace (6.8) with a more complex linear model (e.g. a second order model). In this case, the problem (6.7) could be translated into an equivalent *Quadratic Programming (QP)* problem and solved with a standard *Active Set* algorithm triggered at each sampling instant by the controller. A run-time observers could be exploited to recover states of the considered models when they are not directly available from sensor readings and/or relevant measurement noises are present.

Finally, in the actual implementation of the algorithm (6.9)-(6.10) we added a dead-zone that collapses to $P_{c,idle} = 50mW$ all the values of $P_{c,i}(t)$ lower than $P_{c,min} = 150mW$. This item represents the discontinuity in DVFS between minimum frequency and idle condition, as stated at the beginning of Section 6.2.

We used a sampling time of $\tau_{sT} = 2.5ms$, thus computational time of the MPC routine, $\sim 5\mu s$ (see Section 4.3.1), can be neglected. In general, sampling time is related to the physical properties of the considered system. A rule-of-thumb for its selection in such kind of applications is $1/3$-$1/5$ of the smallest time constant characterizing the uncontrolled system (16).

6.3.2 The Higher-layer PCM controller

Once thermal capping with minimum performance degradation is guaranteed by the lower-layer MPC, the higher level one can be design to manage the PCM in order to ensure guaranteed re-sprinting performance, whenever heterogeneous tasks with different QoS requirements or criticalities are tackled. According to Subsection 6.2.3, we considered two tasks groups; the Best Effort Group and the Guaranteed Group, with a *N*-*M* Guaranteed Re-sprinting requirement (0.2s-4s for the considered benchmark). Taking the cue from Fig. 6.3 and (6.6), the idea is to enable every sprinting request until the PCM internal energy, U, approaches the bound U_b, then a suitable sprinting reduction has to be taken to preserve $U(t) = U_b(t)$, i.e. to preserve the requested re-sprinting room for the Guaranteed Group. Moving from these considerations, we propose the following centralized MPC problem to tackle PCM management.

$$\begin{aligned} &\min_{P^*_{c,i}} \sum_{i=2}^{16} (P^*_{c,i}(t|t) - P_{t,i}(t))^2 \\ &\text{subject to } U(t+1|t) \leq U_b(t+1) \end{aligned} \tag{6.11}$$

where $P^*_{c,i}$, $i = 2,\ldots,16$ are the control knobs of such controller, representing the power references delivered to distributed thermal controllers as reported in Fig. 6.4 and in Section 6.3.1. Differently, $P_{t,i}(t)$, $i = 1,\ldots,16$ are the original computational power targets for the cores; they are equal to $P_{max}/16 = P_{c,max} = 1W$, when a sprinting request is running; whereas, in rest conditions, $P_{t,1} = P_{c,max} = 1W$ and $P_{t,i} = P_{c,idle} = 50mW$ for the other cores. The rationale of this approach is to let the power references $P^*_{c,i}$, for lower level MPC, to be close to $P_{t,i}(t)$, whenever the PCM energy is clearly far from the time-varying repetitive bound U_b (see Fig. 6.3). Differently, when the prediction of the internal energy $U(t+1|t)$ approaches $U_b(t+1)$, power references are decreased. A one-step-ahead preview is assumed on the bound $U_b(\cdot)$, this is admissible according to strategy adopted to derive it in (6.6).

The approximated discrete-time model adopted for prediction purposes, with sampling time τ_{sP}, is

$$U(t+1|t) = U(t) + \tau_{sP}\left(\frac{T_{PCM}(t) - T_{AMB}(t)}{R_{PCM}} + \sum_{i=1}^{16} \frac{T_{c,i}(t) - T_{PCM}(t)}{R_{c,v}}\right) \tag{6.12}$$

It is worth noting that, since T_{PCM} is available from measurements, it can be used as an input and the energy-based model (6.12) will be Linear Time-Invariant, despite the nonlinearity of the PCM behavior.

The PCM internal energy, $U(t)$ adopted in (6.12) is not directly available, an observer is actually used for such variable, exploiting temperature measurements. Some attention has to be paid along the melting condition since the internal energy becomes unobservable from the PCM temperature.

In the prediction model proposed in (6.12), the control knobs $P^*_{c,i}$ are not directly available. In order to apply standard MPC solutions a more complex model would be necessary to highlight the relationships among the control knobs and the internal energy. In addition, this would involve lower-layer thermal controllers, too. To prevent such heavy modeling and to save the separation among the control layers, the simplified model proposed in (6.3) is exploited and the following non-conventional approximated model predictive controller is derived

$$P^*_{c,i}(t) = \begin{cases} P_{t,i}(t) & if\ \bar{U}_b(t+1) > U(t+1|t)\ from\ (6.12) \\ \bar{P}_{t,i}(t) & if\ \bar{U}_b(t+1) \leq U(t+1|t)\ from\ (6.12) \end{cases} \tag{6.13}$$

where $i = 2,\ldots,16$ and $\bar{P}_{t,i}(t) = \bar{P}_{tot}(t)/15$ with,

$$\bar{P}_{tot}(t) = \frac{\bar{U}_b(t+1) - \bar{U}_b(k)}{\tau_{sP}} + \frac{T_{PCM}(t) - T_{AMB}(t)}{R_{PCM}} - P_{t,1}(t) \tag{6.14}$$

With such a solution, (6.12) is used to predict $U(t+1|t)$, while (6.14), directly derived discretizing (6.3), is adopted to compute the power references when $U(t+1|t)$ hits the bound $\bar{U}_b(t+1)$. Similarly to the lower layer thermal controller case, the bound $U_b(t)$ is replaced with $\bar{U}_b(t) < U_b(t)$ to save some margin and to compensate for the approximations of the adopted models. Computations and simulations tests can be carried out to derive a reliable bound $\bar{U}_b(t)$. For the considered benchmark, under 0.2-4s Guaranteed Re-sprinting requirement, $\bar{U}_b(k) = U_b(k) - 0.25J$.

Focusing on $\bar{P}_{t,i}(k)$, it can be shown with straightforward computations that, according to the definition of $U_b(t)$ in (6.6), $\bar{P}_{t,i}(t) = P_{c,idle}$ when $T_{AMB}(t) = T_{AMBmax}$, but it can be larger whenever $T_{AMB} < T_{AMBmax}$. This behavior has been already highlighted at the bottom of Fig. 6.3 and

can be effectively exploited to improve sprinting performance.
It is worth noting that the proposed control approach is valid for every $U_b(\cdot)$, computed according to (6.6), and, then, for every *N-M* Guaranteed Re-sprinting request or even for more involved scenarios. Moreover, as already anticipated, this control layer can be easily and dynamically enabled/disabled according to the current re-sprinting requirement. Whenever no guaranteed re-sprinting is needed, the following power references will be simply set $P_{c,i}(k) = P_{t,i}(k)$ for $i = 2, \ldots, 16$ and only the lower layer will be active. Finally, the adopted sampling time for the considered benchmark is $\tau_{sP} = 10ms$.

6.4 The Implementation

The code below lists the operations executed by the proposed control solution.

Pseudo Code

```
1  CONTROLLER ROUTINE
2    Initialize the parameters of the models used by the Lower/Higher-layer controller
3    FOR EACH HIGHER-LAYER CONTROLLER SAMPLE (10ms)
4      get P_{t,i}(t);
5      read the current T_{c,i}(t), T_{AMB}(t), T_{PCM}(t);
6      compute the forecast of U(t), U(t+1|t);
7      solve equation (6.13) to find P*_{c,i}(t);
8      FOR EACH LOWER-LAYER CONTROLLER SAMPLE (2.5ms) & FOR ALL THE LOCAL CONTROLLER
9        read the current T_{c,i}(t), T_{PCM}(t);
10       compute the maximum sustainable power, P̄_{c,i}(t), for the next sampling interval;
11       solve equation (6.9) to find P_{c,i}(t);
12     END_FOR
13   END_FOR
```

More in detail:

Line 2 Off-line we set the parameter of the models used by both the control layers (e.g. $R_{c,o}$, $R_{c,v}$, C_c and R_{PCM});

Line 3 at each sampling time of the Higher-layer controller the loop from line 3 to line 13 is repeated;

Line 4 the Higher-layer controller receives from the High Level SoC Manager the target power, $P_{t,i}$;

Line 5-6 using the equation (6.12) the Higher-layer controller forecasts the future value of the internal energy stored in the PCM, $U(t+1|t)$;

Line 7 $U(t+1|t)$ is compared with the maximum allowed energy U_b: if it is lower then $P^*_{c,i}(t) = P_{t,i}(t)$, otherwise $P^*_{c,i}(t)$ results from the equation (6.14);

Line 8 at each sampling time of the Lower-layer controller the loop from line 8 to line 12 is repeated;;

Line 9-10 using the equation (6.10) each local Lower-layer controller forecasts the maximum power that can be spent in the next sampling interval, that is the power necessary to reach the temperature T_{max} in the next sample instant, $\bar{P}_{c,i}(t)$;

Line 11 $\bar{P}^*_{c,i}(t)$ is compared with the maximum allowed power $\bar{P}_{c,i}(t)$: if it is lower then $P_{c,i}(t) = P^*_{c,i}(t)$, otherwise $P_{c,i}(t) = \bar{P}_{c,i}(t)$;

6.5 Experimental Results

The proposed solution has been tested on the Matlab/Simulink environment where the simulatore described in Section 6.2.2 has been implemented. Then, the results has been compared with a Threshold-based solution as defined in (10), where each sprinting request is executed at maximum speed, until each core, but the #1, is forced to shutdown once the temperature reaches T_{max}.

Before showing the results it is worth to stress that, to simplify the discussion, the sprinting traces used in simulations are represented in terms of power. Constant workload instruction characteristics are assumed, then power consumption can be seen as proportional to the adopted frequency. In addition, those traces are assumed fixed, even though, in real reactive applications, the sprinting request trace is dynamically affected by the actual responsiveness of the device. This effect has been disregarded here to make the results easier to be interpreted.

Tests can be split into three macro groups. The first shows the behavior of our solution respect to the Threshold-based one when generic workload is applied and no guaranteed re-sprints are needed. In the second, critical periodic tasks are introduced to evaluate the performance of our solution when N-M Guaranteed Re-sprinting are required. Finally, to show the reliability of our solution, we applied non-nominal working conditions to the controlled system.

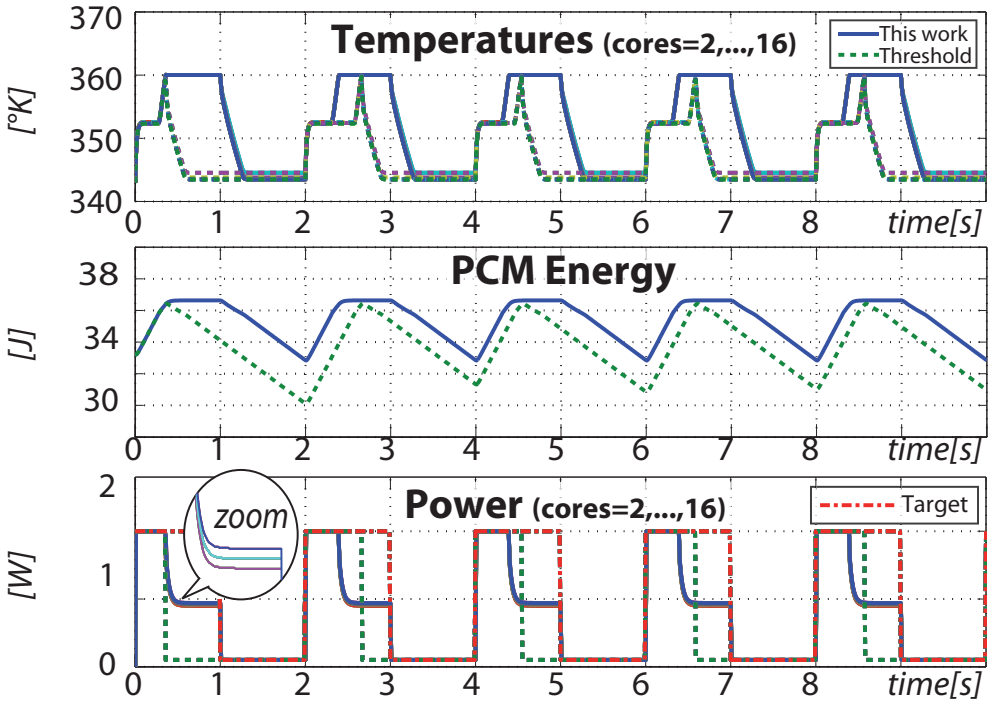


Figure 6.5: Typical non guaranteed trace response

6.5.1 Generic workload

When no guaranteed re-sprints are needed the higher-layer controller (namely PCM Model predictive controller in Fig. 6.4) can be disabled. As consequence of that, only the lower-layer (namely thermal MPC in Fig. 6.4) is active and $P_{c,i} = P_{t,i}$. In Fig. 6.5 we compare the performance of our solution against the Threshold-based one when a sprint of 1s is requested every 2s.

Differently from the threshold one, the proposed solution is not purely on/off and is capable of finding an optimal intermediate operating mode (i.e. operating frequency and voltage, implying a particular core power consumption level) for each core, that maintains the cores temperatures close to the limit. This optimal value is inversely proportional to the ambient temperature. Indeed in a colder environment, the heat is dissipated to the ambient faster, allowing higher cores frequencies (power). Moreover, compared to (10) our distributed controller solution does not assume isothermal silicon temperature and thus can exploit the different core thermal dissipation efficiency[1] as shown in the zoomed area. Threshold-based solution puts

[1]boundary cores are colder than the center one, since they have lower thermal resistance with the PCM layer

all cores to idle (except #1) when any core first crosses T_{max}, our solution instead decreases frequency (power) independently for each core. From the same figure we can notice that our solution leads to a more regular sprinting duration profile, whereas for the threshold one it depends strongly on the previous sprint duration and in the internal energy level.

tasks Gaussian mean value

	μ=1	**μ=1.5**	**μ=2**	**μ=2.5**
This work	97.4	93.4	91.8	81.0
Threshold	95.5	88.5	85.8	68.0

Figure 6.6: Non guaranteed performance comparison

As consequence of that to quantify the improvement obtained with our solution we built 4 sprinting traces of 200*s* constituted by tasks with duration distributed with Gaussian probability (standard deviation 0.5 and mean value respectively 1*s*, 1.5*s*, 2*s*, 2.5*s*) and separated from each other with a rest time generated using a Poisson cumulative distribution function ($\lambda = 10$). The table in Fig. 6.6, collects the percentage of the total operating frequencies (computed integrating on time the operating frequencies of all cores) normalized with respect to the requested one: $\frac{\int \sum_i f_{c,i}(\tau)d\tau}{\int \sum_i f_{t,i}(\tau)d\tau} \cdot 100$; where $f_{c,i}$ and $f_{t,i}$ are the i-th controlled and target core frequencies respectively. Our solution outperforms the Threshold-based one reaching a 19% of improvements when tasks have mean value equal to 2.

6.5.2 Guaranteed re-sprints

As previously discussed when the system executes both critical and sporadic non-critical tasks a guaranteed re-sprint needs to be ensured. In the second set of tests we evaluated the performance of our solution comparing it to the state-of-the-art threshold one. Fig. 6.7 refers to a 0.2-4*s* Guaranteed Re-sprinting scenario and shows in order the temperature, the PCM internal energy and the P_t and P_c of the core #7. In the top subplot we can first notice that Threshold-based solution runs the tasks until the temperatures reach the T_{max}, while in our solution, for a design decision, the PCM cannot cross T_{melt}, and then cores are actually bounded to a temperature a little bit lower than T_{max}. This gives an extra heat storage room to the Threshold-based controller but on the contrary leads to crisper thermal cycles. The internal energy of the system is shown in the central subplot. Whereas our solution keeps the energy below U_b, limiting non

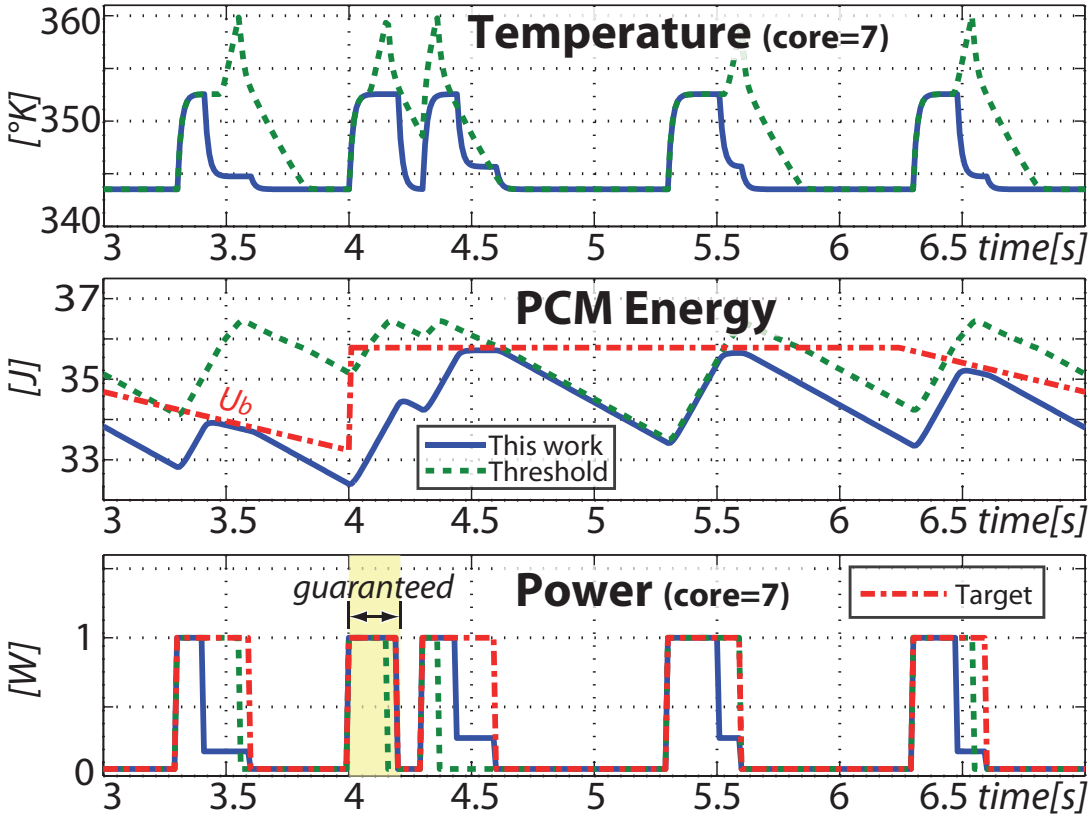


Figure 6.7: Typical guaranteed trace response

guaranteed tasks to have enough energy to run the hard real-time task at time 4*s*, the Threshold-based solution is not so far-seeing. In fact, it cannot ensure a predefined internal energy level at the beginning of a guaranteed task. This leads to an unpredictable and variable duration of the guaranteed sprint phase, that strongly depends on the previous sporadic computation.

Non guaranteed number of tasks

Guaranteed tasks mean value

	λ=3	λ=6	λ=9	λ=12	λ=3	λ=6	λ=9	λ=12	
μ=0.175	100	100	100	100	74.2	76.8	75.6	76.1	*This work*
	94.4	82.4	76.2	70.6	76.6	80.6	80.2	80.7	*Threshold*
μ=0.125	100	100	100	100	76.4	77.0	78.3	77.9	
	98.3	91.4	82.9	79.9	79.0	81.2	83.1	83.0	
μ=0.075	100	100	100	100	78.0	80.5	79.4	81.7	
	99.9	98.9	95.1	92.8	80.4	84.4	84.4	86.7	
μ=0.025	100	100	100	100	83.0	80.8	83.4	82.6	
	100	100	99.6	99.2	84.9	84.3	87.6	87.3	
	% Guaranteed				% Total				

Figure 6.8: Guaranteed performance comparison

To quantify the benefit of our approach in a realistic scenario we performed a set of tests under different stochastic workloads. Fig. 6.8 shows the results. In these tests, the guaranteed tasks duration is determined using a uniform probability distribution function, while the non guaranteed total tasks duration (1.4*s*) of each period (4*s*) has been split into a number of equal parts dependent on a Poisson probability distribution function. Each part constitutes the mean value of a Gaussian distribution with standard deviation $0.3/\#tasks$. Each trace differs from the other for the characteristic number λ of the Poisson function and the mean value of the uniform distribution. The ambient temperature is set to $25\,^{\circ}C$ and the metrics used to evaluate the performance are $\frac{\int \sum_i f_{c,i}(\tau)d\tau}{\int \sum_i f_{t,i}(\tau)d\tau} \cdot 100$ and $\frac{\int \sum_i f^G_{c,i}(\tau)d\tau}{\int \sum_i f^G_{t,i}(\tau)d\tau} \cdot 100$, called respectively $\%Total$ and $\%Guaranteed$ in Fig. 6.8, where $f^G_{c,i}$ and $f^G_{t,i}$ differently from the previously defined total controlled and target frequencies of the i-th core ($f_{c,i}$ and $f_{t,i}$ respectively) are assumed not zero only in guaranteed windows. This two metrics are proportional to the effective throughput. As it is clear from the table our solution guarantees the execution of all hard real-time tasks, whereas the Threshold-based does not. From the same figure we can notice that the operating frequency percentage is slightly higher for the Threshold-based approach. This was expected since our solution does not exploit the core temperature up to T_{max} and because of the hard constraint U_b, adopted in our solution to preserve guaranteed re-sprinting room.

In Fig. 6.9 we repeat the previous tests by considering a Threshold-based solution which switches off the cores when the PCM shows $T_{PCM} > T_{melt}$. In this way we remove the advantage

Non guaranteed number of tasks

Guaranteed tasks mean value

	λ=3	λ=6	λ=9	λ=12	λ=3	λ=6	λ=9	λ=12	
μ=0.175	100	100	100	100	74.2	76.8	75.6	76.1	*This work*
	92.9	79.4	73.7	68.8	74.4	77.0	75.8	76.4	*Threshold*
μ=0.125	100	100	100	100	76.4	77.0	78.3	77.9	
	97.2	89.0	80.4	77.4	76.5	77.2	78.6	78.2	
μ=0.075	100	100	100	100	78.0	80.5	79.4	81.7	
	99.7	97.8	93.1	90.3	78.2	80.8	79.7	82.0	
μ=0.025	100	100	100	100	83.0	80.8	83.4	82.6	
	100	100	99.2	98.7	83.3	81.0	83.6	82.9	
	% Guaranteed				% Total				

Figure 6.9: Guaranteed performance comparison ($T_{max} = T_{melt}$)

for such solution in exploiting the cores up to T_{max}. In this case, the performance loss of our solution for total operating frequency is below the 0.3% (see Fig. 6.9). This is the actual cost of guaranteed re-sprinting rooms.

6.5.3 Non-nominal conditions

Finally, we have tested our solution with non-nominal working conditions with mixed-criticality workload. Since a well suited chip thermal design should prevent critical temperatures when nominal operating frequencies are provided, we set the ambient temperature to $40\,°C$ (i.e. "phone on the beach") and we increased the power consumption of each core (i.e. $2W$ per core instead of $1W$). This emulates a possible leakage power increase due to higher ambient temperatures, aging and process variation. As shown in Fig. 6.10 cores thermal controllers must intervene to reduce power. Power reduction is different from each core and optimized to maximize performance maintaining the temperature close to T_{max}.

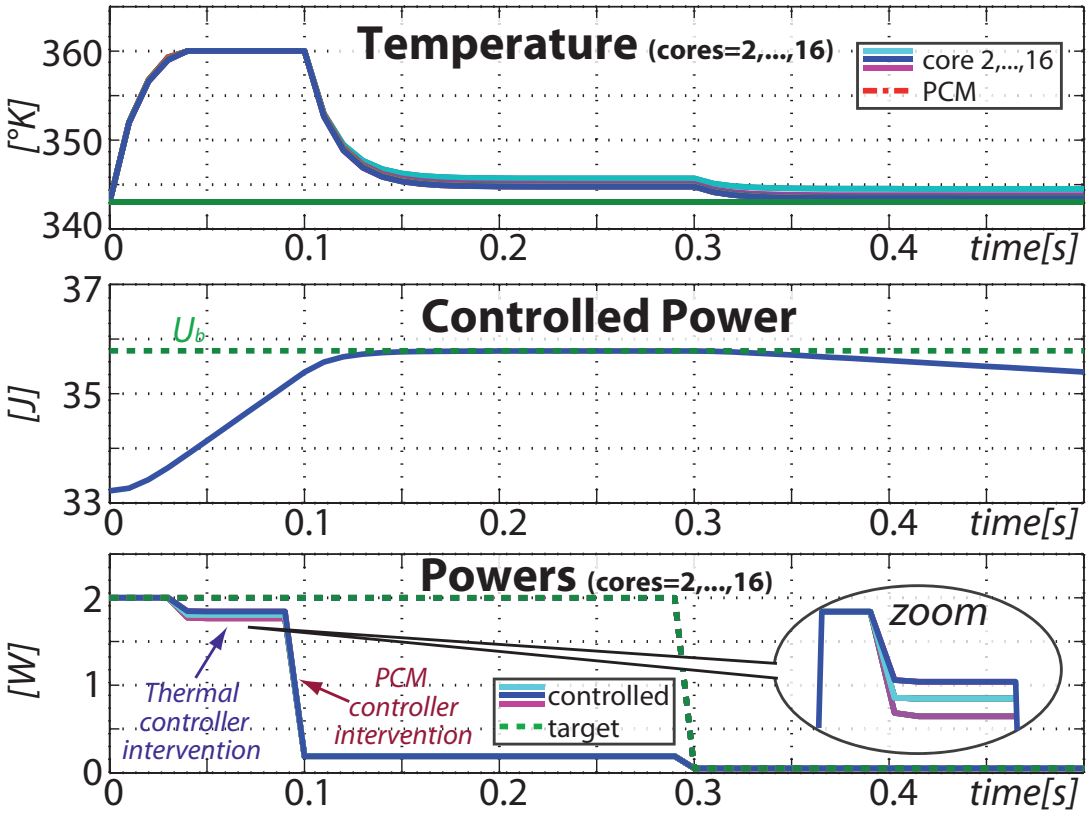


Figure 6.10: Non-nominal workload system response

Bibliography

[1] www.activepower.com, *Data center thermal runaway. A review of cooling challenges in high density mission critical environments*, White Paper 105, 2007. 166

[2] Z. Luo, H. Cho, X. Luo, K. il Cho, *System thermal analysis for mobile phone*, Applied Thermal Engineering, Vol. 28(1415):1889 – 1895, 2008. 166, 169

[3] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, D. Burger, *Dark silicon and the end of multicore scaling*, SIGARCH Comput. Archit. News, Vol. 39(3):365–376, June 2011. 166

[4] R. Yavatkar, M. Tirumala, *Platform wide innovations to overcome thermal challenges*, Microelectronics Journal, Vol. 39(7):930 – 941, 2008. 166

[5] A. Marongiu, L. Benini, *An openmp compiler for efficient use of distributed scratchpad memory in MPSoCs*, Computers, IEEE Transactions on, Vol. 61(2):222–236, Feb. 2012. 166

[6] K. Wonyoung, G. M.S., W. Gu-Yeon, and D. Brooks. *System level analysis of fast, per-core dvfs using on-chip switching regulators*, Proc. of HPCA, pp. 123–134, Feb. 2008. 166

[7] N. Tolia, D. Andersen, and M. Satyanarayanan. *Quantifying interactive user experience on thin clients*, Computer, Vol. 39(3):46–52, Mar. 2006. 166

[8] A. Oulasvirta, S. Tamminen, V. Roto, and J. Kuorelahti. *Interaction in 4-second bursts: the fragmented nature of attentional resources in mobile hci*, Proc. of CHI '05, pp. 919–928, 2005. 166

[9] J. Benson. *Thermal characterization of packaged semiconductor devices*, 2002 166

[10] A. Raghavan, Y. Luo, A. Chandawalla, M. Papaefthymiou, K.P. Pipe, T.F. Wenisch, M.M.K. Martin, *Computational Sprinting*, In HPCA, Feb. 2012. 167, 169, 170, 181, 182

[11] Apple iMovie http://itunes.apple.com/en/app/imovie/id377298193?mt=8 167

[12] iOnRoad http://www.ionroad.com/ 167

[13] B. Gyselinckx, C. Van Hoof, J. Ryckaert, R.F. Yazicioglu, P. Fiorini and V. Leonov. *Human++: autonomous wireless sensors for body area networks*, Proc. of CICC, pages 13- 19, Sept. 2005 167

[14] Climator. *www.climator.com*. 169

[15] A. Tilli, R. Diversi, *A modular approach to dynamic modelling of heat exchangers in vapor compression cycles*, in Proc. of IFAC, Sept. 2011. 170

[16] G. F. Franklin, J. D. Powell, M. L. Workman, *Digital Control of Dynamic Systems*, Addison-Wesley,1997. 178

BIBLIOGRAPHY

Chapter 7

Conclusion and future developments

7.1 Conclusion

The demands for increasingly efficient systems caused the diffusion of MPSoCs architectures in every sector of the worldwide economy, ranging from industries to everyday life products (e.g. laptops, tablets, smartphones,...). Guided by the Moore's law, the rush for high-performance processors has seen single CPUs gradually disappear, due to the unsustainable power consumes, in favor of multi-core processors, which are able to exploit parallelism greatly reducing the power consumption. Nevertheless, several constraints imposed by technological scaling determined an increase of the power density which translated in powers unevenly dissipated on the chip and localized in "hot-spots" where the temperature reaches harmful values that strongly undermines the reliability of the MPSoC. In this respect, studies have shown the correlation between high temperatures and chip failure mechanisms acceleration (e.g. electromigration and stress migration) and the reduction of transistors speed and chip components lifetime. Cooling and heat management are rapidly becoming the key limiters for high performance processors. Another interesting issue, mainly present in mobile devices, is related to the limited power budget and the economical and practical limitations of cooling infrastructures. Indeed, it is now accepted, that in upcoming devices, all the units on a die cannot be repeatedly switched on at the same time, as their total power consumption would exceed the Thermal Design Power (TDP) – the maximum amount of power the cooling system is required to dissipate – leading to thermal run-out.

In this context we developed reliable MPC-based control solutions that maximize performance, limiting temperatures and power consumptions, at the same time. To do that we ex-

ploited MPC controllers for implementing a DVFS technique more efficient than the one obtained with other control theory based solutions. Our solution for managing the temperature of the MPSoC is based on a distributed technology. Compared to the centralized solution present in literature, we have obtained a greater reliability and a far lower computational complexity with similar performance.

Due to the importance of model accuracy and complexity to perform the predictions of the temperature, we explored three approaches respectively based on distributed ARX, H_∞ problem, and proper orthogonal decomposition techniques.

We proved the control feasibility of the centralized and distributed MPC solutions for the family of thermal systems over any prediction horizon. This proof, usually disregarded in literature, is extremely important for guaranteeing the respect of temperature constraints at each time instant. The study has been conducted on a generic thermal model described by partial differential equations and it has revealed other interesting properties for the simplification of the control design.

The distributed MPC solution has been included in more complex control schemes:

- a two-layer control solution able to ensure feasibility and efficiency at the same time;
- a fully distributed solution able to maximize the energy saving;
- a communication-aware solution to allows the communication between cores in a message passing context;
- a MPC hierarchical solution able to contrast the “Utilization Wall” issue in mobile devices.

In particular this latter solution is based on the computational sprinting approach which consist in running all cores at maximum speed only for short time intervals in order to not exceed the temperature limits. Differently respect to literature solutions, our controller is able to maximize performance, and guarantee a time window where executing critical tasks at maximum speed, placed at the beginning of an initially specified time interval. The dimension of the sprinting window has been increased exploiting an opportunely defined phase change material (PCM).

In order to develop and test these control solutions we used the Matlab/Simulink environment. A processor thermal simulator has been implemented. Additionally a C/C++ solution has

been developed to estimate the time overhead necessary for computing the control decisions. The code will be used for the implementation of the control algorithm on real hardware.

7.2 Future works

In the next future the work it is expected to continue, mainly focusing on three activities:

Implementation on real HW. The proposed control solution has been tested on a cycle-accurate simulator. In the next future we expected to implement the control algorithm on a real hardware. In this scenario it will be necessary to reformulate the problem in order to account for the uncertainties unavoidably present on the identified model parameters, and the unmeasurable disturbances acting on the real system which may compromise the effectiveness of the model predictive controller. Under these nondeterministic conditions, literature suggests the reformulation of the MPC controller as a "Robust MPC" problem, that is bounding uncertainties and adopting worst-case approach. Although no robust MPC solutions exist in literature for MPSoCs thermal issue, this approach is often too conservative and pessimistic. Thus, we expected that a "Stochastic MPC" problem will be preferable due to its ability of using the additional information of the probabilistic distribution of the uncertainty to reduce significantly the conservativeness respect to classic approaches. The control algorithm will have soft constraints met with a desired probability instead of hard constraints, the objective function will be formulated as an expected cost and the prediction models will incorporate information on uncertainties. Due to the presence of unmeasurable states a Kalman filter will be designed. The control solution will be implemented and tested on the SCC processor shown in Appendix B.

Hierarchical solution improvement. Another activity will address the "Utilization Wall" issue by improving the hierarchical control solution shown in Chapter 6. Initially, the effectiveness of the controller respect to different chip layouts will be carefully checked. Then, a novel hierarchical control structure will be designed eliminating some of the current assumptions and improving the overall performances.

Heterogeneous multi-core. In the next future multi-core chips will be composed of many specialty cores working in concert, each one with a particular role inside the device (e.g. mobile phones already use heterogeneous cores). In this context the management of the different resources must be carefully controlled to improve performance and reduce consumes. A "Hybrid MPC controller" able to handle both the temperature and the workload of the processors,

and a consensus based task scheduling manager are interesting solutions for improving heterogeneous processors performance.

Chapter 8

Publications

C. Bonivento, M. Cacciari, A. Paoli, M. Sartini. (2010) *Mathematical modeling for Software-In-the-Loop prototyping of automated manufacturing systems.*
Mathematical Methods in Engineering International Symposium, Coimbra 21-24 Ottobre 2010.

A. Bartolini, M. Cacciari, A. Tilli, L. Benini, M. Gries. (2010) *A virtual platform environment for exploring power, thermal and reliability management control strategies in high-performance multicores.*
GLSVLSI '10 Proceedings of the 20th symposium on Great lakes symposium on VLSI 2010. Providence, Rhode Island (USA). May 16 -18-2010. (pp. 311 - 316). ISBN: 978-1-4503-0012-4. : ACM (UNITED STATES).

A. Bartolini, M. Cacciari, A. Tilli, L. Benini. (2011) *A Distributed and Self-Calibrating Model-Predictive Controller for Energy and Thermal management of High-Performance Multicores.*
Design, Automation & Test in Europe Conference & Exhibition, 2011. DATE '11. Grenoble, France. 14-18 March 2011. NEW YORK: IEEE Press (UNITED STATES).

A. Bartolini, M. Cacciari, A. Cellai, M. Morelli, A. Tilli, L. Benini. (2011) *Fault Tolerant Thermal Management for High-Performance Multicores.*
Design, Automation & Test in Europe Conference & Exhibition, 2011. DATE '11. Grenoble, France. 18 March 2011. (workshop uPM2SoC)

C. Bonivento, M. Cacciari, A. Paoli, M. Sartini. (2011) *Rapid prototyping of automated man-*

ufacturing systems by software-in-the-loop simulation.
Chinese Control and Decision Conference, 2011. CCDC'11. 23-25 May 2011, pp.3968–3973.

A. Bartolini, M.S. Sadri, F. Beneventi, M. Cacciari, A. Tilli, L. Benini. (2011) *A System Level Approach to Multi-core Thermal Sensors Calibration.*
PATMOS 2011.

A. Bartolini, M.S. Sadri, F. Beneventi, M. Cacciari, A. Tilli, L. Benini. (2011) *SCC Thermal Sensor Characterization and Calibration, MARC3 Symposium 2011.*

A. Tilli, E. Garone, M. Cacciari, A. Bartolini. (2012) *Thermal Models Characterization for Reliable Temperature Capping and Performance Optimization in Multiprocessor Systems on Chip.*
ACC, June 2012.

A. Bartolini, M. Cacciari, A. Tilli, L. Benini. (2012) *Thermal and Energy management of High-Performance Multicores: Distributed and Self-Calibrating Model-Predictive Controller.*
IEEE TDPS 2012 vol. 23.

A. Tilli, A. Bartolini, M. Cacciari, L. Benini. (2012) *Don't burn your mobile! Safe Computational Re-Sprinting via Model Predictive Control.*
ESWeek 2012, Tampere Hall, Finland.

Appendices

Appendix A

Mathematical Background

In this Appendix are reported useful contents relative to optimization problem and model predictive control.

A.1 Convex Linear MPC with quadratic cost function implementation

The main control tool we used in this thesis is the linear MPC algorithm. It is characterized by a convex quadratic cost function and affine constraint functions. The main contribution of this Section is to give an idea of how the MPC algorithm is usually implemented.

Assume this is the optimization problem which must be solved to obtain the next control decision:

$$\min_{U_0} J_0(x(0),U_0) \triangleq x(h_p)' P x(h_p) + \sum_{k=0}^{h_p-1} x(k)' Q x(k) + u(k)' R u(k) \tag{A.1a}$$

$s.t.$

$$x(k+1) = Ax(k) + Bu(k) \quad k = 0,\dots,h_p-1 \tag{A.1b}$$

$$E x(k) + M u(k) \leq \psi_k \quad k = 0,\dots,h_p-1 \tag{A.1c}$$

$$x(0) = x(t) \tag{A.1d}$$

where $P = P' \succeq 0$, $Q = Q' \succeq 0$, $R = R' \succ 0$, $U_0 = [u(0),\dots,u(h_p-1)]'$ is the control sequence that we want to optimize, h_p is the prediction horizon, (A.1b) is the model of the plant, and $x(t)$ is its current state. Notice that the problem and the model are time-invariant, therefore the control sequence will depend only on the initial state. This is the reason why we put $x(0)$

instead of $x(t|t)$ and $U_0 = [u(0),\dots,u(h_p-1)]'$ instead of $U_{t\to t+h_p-1} = [u(t|t),\dots,u(t+h_p-1|t)]'$.

Assume that a full measurement of the state $x(t)$ is available at the current time t (otherwise an observer is necessary). Then, the optimization problem must be solved at each sampling time.

Each MPC problem has a different structure that depends on the application and the requirements it has to satisfy. However, the implementation of an ad-hoc solver is time consuming, and extremely inefficient. The usual way of solving linear quadratic MPC problems is to translating the optimization problem into a QP problem for which efficient solvers based on active-set methods and interior point methods are available.

The steps to obtain the QP formulation are presented below. First, we need to rewrite the problem (A.1) in the matrix form,

$$\min_{U_0} J_0(x(0),U_0) \triangleq x(h_p)' P x(h_p) + X_0' \mathcal{Q} X_0 + U_0' \mathcal{R} U_0 \tag{A.2a}$$

$s.t.$

$$X_0 = \mathcal{A} x(0) + \mathcal{B} U_0 \tag{A.2b}$$

$$\mathcal{E} X_0 + \mathcal{M} U_0 \leq \Psi \tag{A.2c}$$

$$x(0) = x(t) \tag{A.2d}$$

where $U_0 = [u(0),\dots,u(h_p-1)]' \in \mathbb{R}^{h_p m}$, $X_0 = [x(0),\dots,x(h_p-1)]' \in \mathbb{R}^{h_p n}$, and
$\mathcal{Q} = diag\{Q,\dots,Q\} \in \mathbb{R}^{h_p n \times h_p n}$,
$\mathcal{R} = diag\{R,\dots,R\} \in \mathbb{R}^{h_p m \times h_p m}$,
$\mathcal{E} = diag\{E,\dots,E\} \in \mathbb{R}^{h_p n_C \times h_p n}$,
$\mathcal{M} = diag\{M,\dots,M\} \in \mathbb{R}^{h_p n_C \times h_p m}$.
Moreover, the matrices $\mathcal{A} \in \mathbb{R}^{h_p n \times n}$ and $\mathcal{B} \in \mathbb{R}^{h_p n \times h_p m}$ can be defined as,

$$\mathcal{A} = \begin{bmatrix} A \\ A^2 \\ \vdots \\ A^k \end{bmatrix} \quad \mathcal{B} = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ B & 0 & 0 & \cdots & 0 \\ AB & B & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ A^{h_p-2}B & A^{h_p-3}B & \cdots & \cdots & 0 \end{bmatrix} \tag{A.3}$$

remembering that the state at each sampling time can be defined respect to the initial state $x(0)$ and eliminating the intermediate states as,

$$x(k) = A^k x(0) + \sum_{j=0}^{k-1} A^j B u(k-1-j) \tag{A.4}$$

Substituting X_0 of equation (A.2b) in the equations (A.2a) and (A.2c) we obtain,

$$\frac{1}{2}x(0)'Y_{QP}x(0)+\min_{U_0}\frac{1}{2}U_0'Q_{QP}U_0+x(0)'F_{QP}U_0 \tag{A.5a}$$

$$s.t.$$

$$M_{QP}U_0 \leq W_{QP}+E_{QP}x(0) \tag{A.5b}$$

(A.5c)

where $Q_{QP}=\mathcal{B}'\bar{\mathcal{Q}}\mathcal{B}+R=Q_{QP}'\succ 0$, $F_{QP}=\mathcal{A}'\bar{\mathcal{Q}}\mathcal{B}$, $Y_{QP}=\mathcal{A}'\bar{\mathcal{Q}}\mathcal{A}$, and $\bar{\mathcal{Q}}=diag\{\mathcal{Q},P\}$. Notice that both Q_{QP}, F_{QP}, M_{QP}, W_{QP}, E_{QP} depend on the matrices Q and R, and that the term with weight matrix Y_{QP} is usually avoided since it represent a fixed cost independent from the manipulable variable U_0. Notice also that in the unconstrained case the solution of this problem can be easily obtained by zeroing the gradient of the cost function or alternatively in a recursive way using the dynamic programming approach.

A.2 Multi-parametric Quadratic Programming

This Section presents the algorithm for solving a multi-parametric quadratic program (mpQP) in order to determine explicitly the state feedback control law which minimizes the optimization problem (A.5). According to the operations research jargon, a *multi-parametric program* is a problem that depends on a vector of variables. In this case we want to find a function $u^{\circ}(0)=f(x(0))$ for all feasible values of x. This approach represents an alternative to the online computation presented in the previous Section which is sometimes impracticable due to computing time and costs.

In this thesis we applied this approach to our linear quadratic MPC in order to find its explicit solution. The starting point is the QP problem (A.5) influenced by the parameter $x\in X\subseteq\mathbb{R}^n$.

Assume there exists a subset of feasible parameters such that,

$$X^{\circ}=\{x(0)\in X:\exists U_0 \quad satisfying \quad M_{QP}U_0\leq W_{QP}+E_{QP}x(0)\} \tag{A.6}$$

If the set is empty the problem could not be solved.

We can solve the problem (A.5) using the Karush-Kuhn-Tucker (KKT) conditions. For any optimization problem with differentiable objective and constraint functions for which strong duality obtains, any optimal solution of the primal and dual problems must satisfy the KKT

conditions. Because our problem is convex, KKT conditions are also sufficient (1). The KKT of our problem are:

$$Q_{QP}\,U_0^\circ + F'_{QP}\,x(0) + M'_{QP}\,\lambda = 0 \tag{A.7a}$$
$$\lambda_i\,(M^i_{QP}\,U_0^\circ - W^i_{QP} - E^i_{QP}\,x(0)) = 0 \quad i = 1,\ldots,m \tag{A.7b}$$
$$\lambda \geq 0 \tag{A.7c}$$
$$M_{QP}\,U_0^\circ \leq W_{QP} + E_{QP}\,x(0) \tag{A.7d}$$

where λ represents the vector of Lagrangian multipliers.

Solving for U_0° the equation (A.7a) becomes,

$$U_0^\circ = -Q_{QP}^{-1}\,(F'_{QP}\,x(0) + M'_{QP}\,\lambda) \tag{A.8}$$

where we remark that $Q_{QP} \succ 0$. Equation (A.8) can be simplified as,

$$U_0^\circ = -Q_{QP}^{-1}\,(F'_{QP}\,x(0) + \widetilde{M}'_{QP}\,\widetilde{\lambda}) \tag{A.9}$$

where we indicated with the accent $\widetilde{\ }$ the constraints part relative to active constraints, i.e. constraints holding with equality at the optimum. For inactive constraints $\lambda_i = 0$. We can substitute U_0° in the complementarity condition (A.7b) obtaining,

$$-\widetilde{M}_{QP}\,Q_{QP}^{-1}\,(F'_{QP}\,x(0) + \widetilde{M}'_{QP}\,\widetilde{\lambda}) = \widetilde{W}_{QP} + \widetilde{E}_{QP}\,x(0) \tag{A.10}$$

Since the rows of $\widetilde{M}_{QP}$ are linearly independents, $\widetilde{M}_{QP}\,Q_{QP}^{-1}\,\widetilde{M}_{QP}$ exists, and we can find $\widetilde{\lambda}$ as,

$$\widetilde{\lambda} = -(\widetilde{M}_{QP}\,Q_{QP}^{-1}\widetilde{M}'_{QP})^{-1}\,(\widetilde{W}_{QP} + (\widetilde{E}_{QP} + \widetilde{M}_{QP}\,Q_{QP}^{-1}\,F_{QP})\,x(0)) \tag{A.11}$$

Thus, $\widetilde{\lambda}$ is an *affine function* of $x(0)$. Substituting (A.11) in (A.9) we obtain, U_0° as,

$$U_0^\circ = Q_{QP}^{-1}\,[\widetilde{M}'_{QP}\,(\widetilde{M}_{QP}\,Q_{QP}^{-1}\,\widetilde{M}'_{QP})^{-1}(\widetilde{W}_{QP} + (\widetilde{E}_{QP} + \widetilde{M}_{QP}\,Q_{QP}^{-1}\,F_{QP})\,x(0)) - F_{QP}\,x(0)] \tag{A.12}$$

Thus, also the optimizer function U_0° is *affine*. Moreover, using the receding horizon strategy $u^\circ(0) = [I\ 0\ \ldots\ 0]\cdot U_0^\circ$ that is affine too. However, this solution is valid only for the states belonging to the region where the set of the active constraints remains unchanged. We define this region as the *critical region* (CR_A), that is the set of parameters x for which the same set A of constraints is active at the optimum. A new state x belongs to the critical region if the optimizer function U_0° meets the primal condition (A.7d) and (A.7c) keeps non-negative values. Thus if

$$\begin{aligned} &M_{QP}\,Q_{QP}^{-1}\,[\widetilde{M}'_{QP}\,(\widetilde{M}_{QP}\,Q_{QP}^{-1}\,\widetilde{M}'_{QP})^{-1}(\widetilde{W}_{QP} + (\widetilde{E}_{QP} + \widetilde{M}_{QP}\,Q_{QP}^{-1}\,F_{QP})\,x(0)) - F_{QP}\,x(0)] \leq W_{QP} + E_{QP}\,x(0) \\ &-(\widetilde{M}_{QP}\,Q_{QP}^{-1}\widetilde{M}'_{QP})^{-1}\,(\widetilde{W}_{QP} + (\widetilde{E}_{QP} + \widetilde{M}_{QP}\,Q_{QP}^{-1}\,F_{QP})\,x(0)) \geq 0 \end{aligned} \tag{A.13}$$

the CR_A set can be defined as,

$$CR_A = \{x \in X : (A.13) \; are \; satisfied\}$$

As it is possible to see from (A.13) CR_A is a polyhedron in X. After having defined the critical region CR_A the rest of the space must be explored in order to generate other critical regions.

Summarizing, let $A_i(\overline{x})$ be the set of active constraints for $\overline{x} \in X^\circ$, then we have proved that,

- CR_A is a polyhedron;
- the optimizer function U_0° is an affine function of the states inside CR_A, i.e. $U_0^\circ = G_{u,i} \cdot x + Off_{Ri} \quad \forall x \in CR_A$
- the value function $J_0^\circ(x)$ is a quadratic function of the states inside CR_A, i.e. $J_0^\circ(x) = x' \cdot M_i \cdot x + c_i \cdot x + d_i \; \forall x \in CR_A$

This is true for all critical region CR_{Ai} associated to a specific set of active constraints. Finally, it is also possible to prove that,

Proposition A.2.1. *Consider the multi-parametric quadratic program (A.5) and let $Q_{AP} \succ 0$. Then, the optimizer U_0° is continuous and piecewise affine in each polyhedral critical region, and the optimal solution $J_0^\circ(x)$ is continuous, convex and piecewise quadratic on polyhedra.*

Additionally, $J_0^\circ(x)$ is a $\mathcal{C}^{(1)}$ function (3) (2).

A.2.1 A mpQP algorithm

The mpQP algorithm goal is to partition the feasible state space X° into a set of critical regions CR_{Ai} and find the expression of the functions U_0° and $J_0^\circ(x)$. It usually comprises an *active set generator* and a *KKT solver*. The former computes the set of active constraints A_i for a particular value of the state, whereas the latter find the CR_{Ai} and the values of U_0° and $J_0^\circ(x)$ associated to the states inside the region. The algorithm presented below is one of the simplest algorithm developed in these years.

Algorithm.

1. Find the starting state vector x_0 inside the polyhedral set X° as the center of the largest ball contained in X°;
2. solve the QP problem for $x(0) = x_0$ to obtain $(U_0^\circ, \lambda^\circ)$;

3. determine the set of active constraint A_0 for U_0° and $x(0) = x_0$;

4. using the functions (A.11) and (A.12), characterize the region CR_{A0};

5. partition the rest of the space X° in region R_i;

6. for each new R_i repeat the code from point (1).

The Fig. A.1 summarizes the procedure.

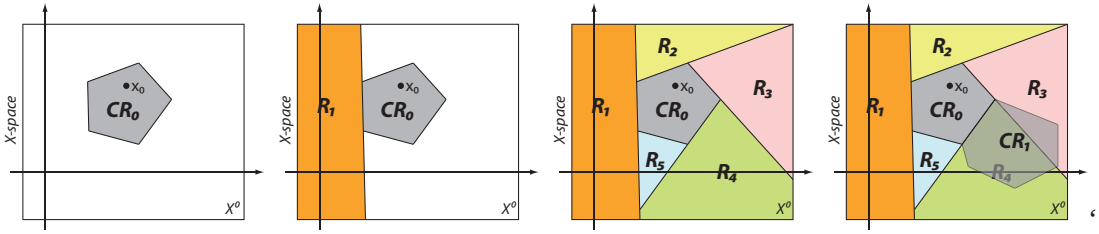


Figure A.1: mpQP algorithm description (from dispenses of Prof. Bemporad).

It is worth to remark that of the optimal control sequence U_0° only the first input is applied. For this reason it is possible to reduce the number of critical regions by condensing the critical regions for which the first element of the control vector is the same.

Bibliography

[1] S. Boyd, L. Vandenberghe, *Convex Optimization*, Cambridge University Press, 2004. 202

[2] F. Borrelli, A. Bemporad, M. Morari, *Model Predictive Control for linear and hybrid systems*, in preparation, last update Nov, 2012. 203

[3] A. Bemporad, M. Morari, V. Dua, and E. Pistikopoulos, *The explicit linear quadratic regulator for constrained systems*, Automatica, Vol. 38(1):3-20, 2002. 203

BIBLIOGRAPHY

Appendix B

MPSoCs and Simulators

In the first part of the appendix a brief review of the terminology used in these thesis regarding the chip system is presented. Then the accurate model used for simulations is described.

B.1 The MPSoC System

When we look at a computing system we can distinguish two major topic: softwares and hardware. *Softwares* are all the programs that direct the actions of the hardware and are classified in *application software* and *system software*. The former are all the application that can be invoked by a user, whereas the latter are interfaces between the system hardware and application softwares. The *operating system* is a part of the system software, that allows the applications to use the hardware resources (thanks to application programming interface or API). An application is usually divided in processes. A *process* (or task) is an instance of a program which has been executed. It is an independent execution unit that contains its own state information, use its own address spaces, and only interact with other processes via interprocess communication mechanisms. Each process can contain one or more threads. A *thread* is the smallest list of *instructions* – the basic commands understood by the processor – that can be scheduled by the operating system on a processor. Focusing on *hardware* it comprises all the physical components (processing units included). Each *processor* logically comprises two main components: the *datapath*, that performs arithmetic operations and the *control unit*, that commands the datapath, I/O devices, and memory according to the instructions of the program. The components inside of a single-core processor comprises: an instruction decoder, which interprets the successive instructions (fetched from memory), an arithmetic unit, which perform operations (add,

compare, move, ...) on quantities contained in registers (and sometimes in memory), a program counter which keeps track of the current location in the process, a bus control circuitry which handles communication with memory and I/O. It also includes *cache memories*, which holds values recently fetched from memory for quicker access (cache memories are small and fast memories realized using the static random access memory (SRAM) technology. They act as buffer for DRAM memory, are faster but more expensive than this latter).

With the advent of Multiprocessors Systems-on-Chip (MPSoCs) and multi-core, in the same substrate of silicon we can find multiple processing units. It is important to notice the difference existing between the previously mentioned processors and Multiprocessors. This latter refer to systems with multiple processors but not on the same chip. From now we refer to MPSoCs and multi-cores as synonyms. The idea under the rising of MPSoCs consists in exploiting parallelism to increase the average throughput. This solution has been necessary to reduce the power consumption, that reached unsustainable quantity due to the failure of Dennard's scaling.

B.2 The Power Consumption

In the past years single-core processors performance increase exponentially, doubling every two years. The primary drivers for this incredible trend were technology and microarchitectural improvements. Microarchitectural techniques exploit the abundance of transistors to increase processor efficiency with instruction level parallelism, extraction techniques, deeper pipeline, ... Technology improvements mainly refers to transistor scaling. In 1974, a work of Robert H. Dennard, a IBM fellow, proved that as transistors get smaller, they can switch faster and use less power. This theory, called Dennard's scaling theory, underlies the most famous Moore's law which states that the number of transistors on a chip double every two years.

A transistor can be seen as an on/off switch controlled by an electric signal. The first transistors used on processors were nMOS, then substituted with CMOS for noise immunity and low static power consumption reasons.

However, as the transistor dimensions reached atomic sizes, Dennard's theory failed. At every technology generation as transistor doubled and performance improved, the power consumption exponentially increased. In the early 2000's processors hit the so called "Power Wall" that refers to the impossibility of improving processors performance by scaling. This because the power consumption of the chip would have generated an amount of heat impossible to

dissipate with conventional cooling systems. The cooling infrastructure costs and the heat extraction limit imposed to the processor industries a change of strategy with the introduction of the multi-core paradigm.

In order to understand better the problem we explain briefly how to compute the power consumption of a processor.

The power consumption of a processor strictly depends on the power consumption of a transistor. Indeed, every integrated circuit (as it is a processor) is made of transistors. In a processor the number of transistors may be of billions. So lets start from the power consumption of a CMOS transistor. The great of the power is spent in moments: during the switching and during the rest. The former component is usually called *dynamic power* while the latter *static power*:

$$P_{CMOS} = P_{dynamic} + P_{static} \tag{B.1}$$

A CMOS transistor comprises an nMOS transistor in series with a pMOS transistor. The circuit of a CMOS inverter is shown in Fig. B.1b where V_{IN} and V_{OUT} are the input and output voltage, C is the load capacitor, and V_{dd} is the supply voltage.

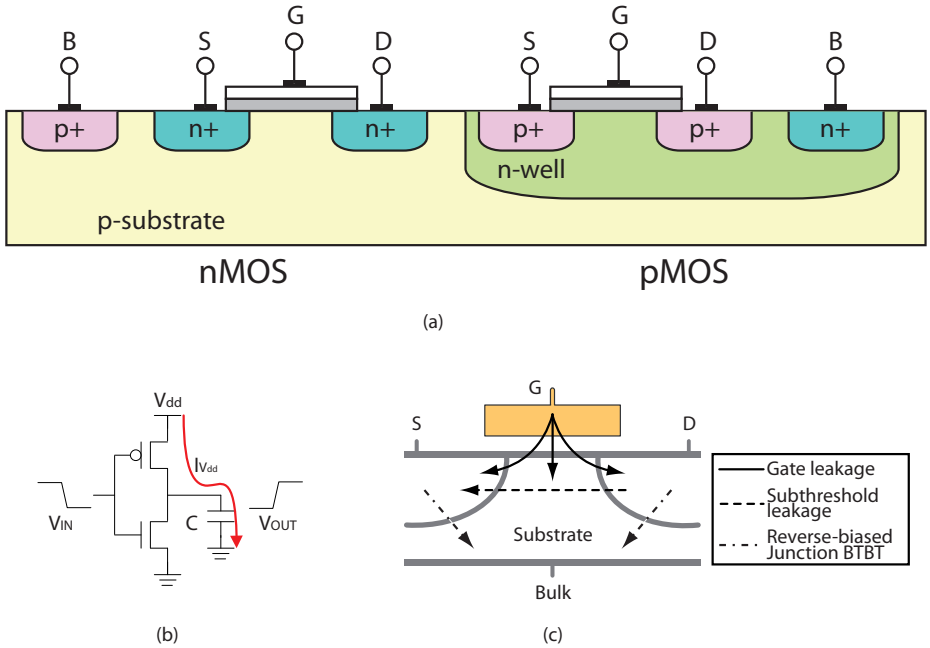


Figure B.1: (a) CMOS transistor; (b) CMOS inverter circuit during switching low-to-high; (c) Leakage current.

The dynamic power is the power spent to charge and discharge the load capacitance, in order to switch the output voltage from low-to-high or from high-to-low. The power can be obtained multiplying the switching frequency ($freq$) by the energy dissipated for one switching. Assuming an ideal V_{IN} with zero rise/fall time, this energy can be computed as the integral of the instantaneous power over the period of interest:

$$E_{V_{dd}} = \int_0^{\infty} I_{V_{dd}} \cdot V_{dd}\, dt = V_{dd} \int_0^{\infty} C \cdot \frac{dV_{OUT}}{dt}\, dt = C \cdot V_{dd} \int_0^{V_{dd}} dV_{OUT} = C \cdot V_{dd}^2 \tag{B.2}$$

Thus, every time a capacitive node switches from ground to $_{Vdd}$, an energy of $C \cdot V_{dd}^2$ is consumed. The power drawn from the supply is given by,

$$P_{dynamic} = C \cdot freq \cdot V_{dd}^2 \tag{B.3}$$

It is important to note that slower circuit (i.e. low $freq$) consume less power not less energy, and that power is a function of the voltage squared, therefore it is convenient to reduce V_{dd}. The failure of Dennard's scaling depends on the impossibility of decreasing V_{dd}.

The static power is related to the currents flowing when the V_{IN} voltage is unchanged. It is usually referred to as leakage power. Ideally, the transistor has a null power consumption when it is in steady-state. However real system has undesired currents flowing from the drain to the source (subthreshold leakage), from the gate to the drain/source (gate leakage) and between the p and n regions (Band-to-Band Tunneling current). These currents increase exponentially as the size of transistors and the threshold voltage reduce (that is another reason to limit the lower value of the V_{dd}). After an estimation of these currents the static power is given by,

$$P_{static} = I \cdot V_{dd} \tag{B.4}$$

The total power consumption of a transistor can be stated as,

$$P_{CMOS} = freq \cdot C \cdot V_{dd}^2 + I \cdot V_{dd} \tag{B.5}$$

Notice that the switching and the leakage power consumption are the most important contributions, but not the only one in a real system. As an example, if V_{IN} transitions are not instantaneous, there could be a period in which both the nMOS and the pMOS transistors simultaneously conduct, generating a current between power supply and ground terminals. The *short-circuit power* is the dissipation caused by this current (1).

We found the power consumption of a CMOS, but a processor counts many millions of transistors. How much power does it consume?

As a first approximation, the total power can be computed as the sum of the power consumption of each transistor. However, the global power consumption at any given time is not always the same. Indeed, the dynamic power is closely tied to the number of transistor that change state. Therefore it is usual to find a constant A which pre-multiply the total dynamic power,

$$P_{TOT} = A \cdot freq \cdot C_{LOAD} \cdot V_{dd}^2 + I_{EFF} \cdot V_{dd} \tag{B.6}$$

A represents the activity factor, that is the fraction of the circuit that is switching and it depends on the workload requested, $freq$ is the clock frequency of the processor, C_{LOAD} is the circuit equivalent capacitance, V_{dd} is the supply voltage (2).

B.3 The Power Model

How can we compute the power consumption practically?

The use of the equation (B.6) for characterizing a real platform often leads to inaccurate Power Models which can compromise the efficiency of a control algorithm. There exist different drawbacks that prevent the use of such a model,

1. the information of the internal architecture, material, geometry of the processor usually are unknown because protected by intellectual property or because too complex to model;
2. each component on the chip is different from the others. This is the so called variability issue which depends on the tolerance used during the production process. For this reason even though we perfectly know the architecture we will encounter low accuracy;
3. the ageing effects can modify the behaviour of the system;
4. the parameter *A* that represents the relation between power dissipation and workload is unknown.

For all these reasons the Power Model used in this thesis has been identified directly from measurements obtained from a Intel® server system S7000FC4UR[1]. It runs four quad-core Xeon® X7350 processors at 2.93GHz and has a total memory capacity of 16GB based on FBDIMMs. The Xeon® X7350 consists of two dual-core Core™2 architecture dies in a single package. Each of the two dual cores share a common 4MB sized L2 cache.

[1] Those experiments were conducted by Andrea Bartolini in (3)

The power profile of the platform has been characterized performing three sets of tests. The first test consists in running a power virus – a task that maximizes the power consumption of the CPU – in order to extrapolate the static power of each core. Indeed the dynamic power can be obtained subtracting the idle power (the power when all cores are in rest condition) from the maximum power measured, and the static power by subtracting the dynamic power from the TDP specification, that is,

$$P_{dynamic} = P_{MAX} - P_{IDLE} \tag{B.7a}$$

$$P_{static} = P_{TDP} - P_{dynamic} \tag{B.7b}$$

The second test investigates the contribution to the whole power of voltage and frequency. The experiment consisted on the one hand in forcing the cores to switch to different frequency values without scaling voltage, and on the other hand in scaling also voltage. Comparing the results obtained by scaling frequency (DFS) and scaling both frequency and voltage (DVFS), DVFS shows that voltage reduction accounts for up to 10% of total system savings. Focusing on the dynamic power reduction, it is super-linear on the frequency scaling factor for DVFS (for a decrease in frequency by 1.83, the dynamic power reduces by 2.95 and 2.86 for power virus and a memory bound benchmark respectively), whereas for DFS it is sub-linear (for a decrease in frequency by 1.83, the dynamic power reduces by 1.68 and 1.43 respectively) (3). The results of the test has been used to fit the parameters of a simple analytical model of the dynamic power at different voltage and frequency levels.

The third test goal is to characterize the relation between the core power consumption and the workload at different performance levels. Indeed the previous test has been performed running the same power virus. In this test a set of synthetic benchmark with different memory utilization has been executed on processors, which has been forced to run at different performance levels. For each benchmark has been extracted the clocks per instruction (CPI) metrics, that quantified the workload on the processor, and it has been correlated with the power consumption. Note that the CPI is correlated to the activity factor A presented in the previous section.

The simple model used in test two fits well for high CPI, but not for low one. Therefore, a new analytical model has been adopted. The dynamic power is given by,

$$P_{dynamic} = k_A \cdot freq \cdot V_{dd}^2 + k_B + (k_C + k_D freq) \cdot CPI^{k_E} \tag{B.8}$$

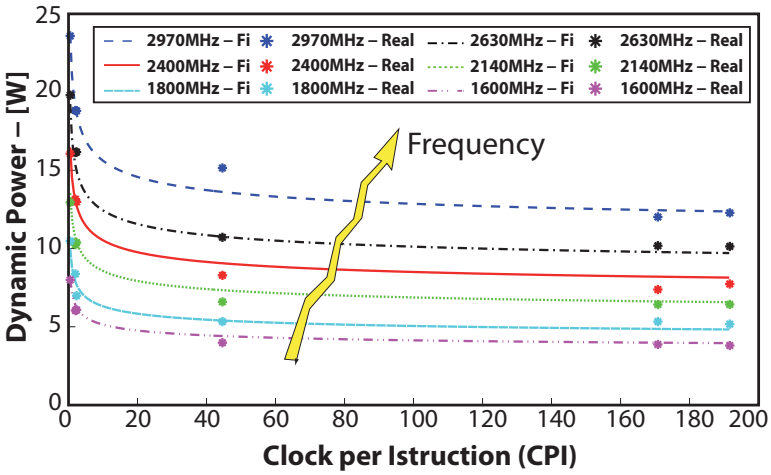


Figure B.2: Per-core Power Based on Activity.

whereas the static power,

$$P_{static} = Z \cdot V_{dd} \cdot T^2 \cdot e^{\frac{-q \cdot V_t}{K \cdot T}} \tag{B.9}$$

where $k_A = 2.13e-3$, $k_B = -1.45$, $k_C = -4.1376$, $k_D = 0.0051$, $k_E = -0.3016$, $Z = 2.59e+02$, $K = 1.38e-23$, $q = 1.60e-19$, V_t is the threshold voltage, and T is the temperature.

These two equations constitute the *Power Model* we used in the thesis (sometimes referred to as *freq2pow* model). It relates the power consumption of the processor to the workload (CPI), the clock frequency, the voltage supply and also the temperature.

It is worth to note that in order to simplify the inversion of the Power Model, the dynamic power can be simplified as,

$$P_{dynamic} = a \cdot freq^2 + b \cdot freq + c \tag{B.10}$$

where $a = 1.549e-6$, $b = 5.1e-3 \cdot CPI^{-0.3016} - 0.002003$, $c = 2.37 - 4.138 \cdot CPI^{-0.3016}$.

Notice that the supply voltage disappear from the function used to compute the dynamic power. However, the contribution of the voltage is incorporated inside the term $a \cdot freq^2$. This because the V_{dd} is a nonlinear function of the frequency ($h(freq)$). It is common to find in literature the $freq$ and the V_{dd} related to the power with a function proportional to $freq^{\gamma}$ where $1 < \gamma < 2$.

B.4 The Thermal plant

The transistor scaling trend dictated by the Moore's Law will persist also in the next future. As a result the power densities on the chip exponentially increase causing high temperatures that affect the reliability of the processors. Heat dissipation has become a key issue for the development of high performance MPSoCs.

From a construction point of view, the architecture of a chip is designed to dissipate the heat generated from the active silicon device layer. The heat is conducted through the silicon die, to the Thermal Interface Material (TIM) which, filling the gap between material asperities, reduces the contact thermal resistance (see Fig. B.3a). Then, the heat flows through the heat spreader and the heat sink and finally is convectively removed to the ambient air.

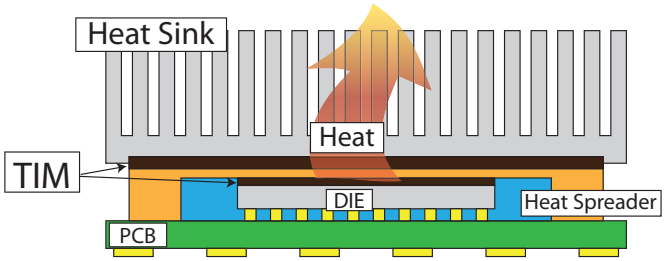


Figure B.3: Chip thermal architecture.

In order to develop a correct thermal management strategy it is necessary to build a model where to test the control algorithm before implementing them on a real hardware. Using a model of the system (that in the follows we will call as plant) guarantees different advantages:

- it measures all the parameters;
- it allows the designers to rapidly modify the control algorithm;
- it allows the designers to rapidly change the chip architecture under exam;
- it saves time and costs;
- it avoids hardware breaking.

The dynamic thermal management techniques implemented in this thesis have been tested on two different type of simulators, one realized in Matlab/Simulink (5) environment, the other, more accurate and complex, is based on Simics (6).

B.4.1 Matlab/Simulink Simulator

The thermal model of a generic processor takes as inputs the frequency, the workload (CPI), the voltage and the ambient information and it returns, as output, the temperature map of the system (see Fig. B.4).

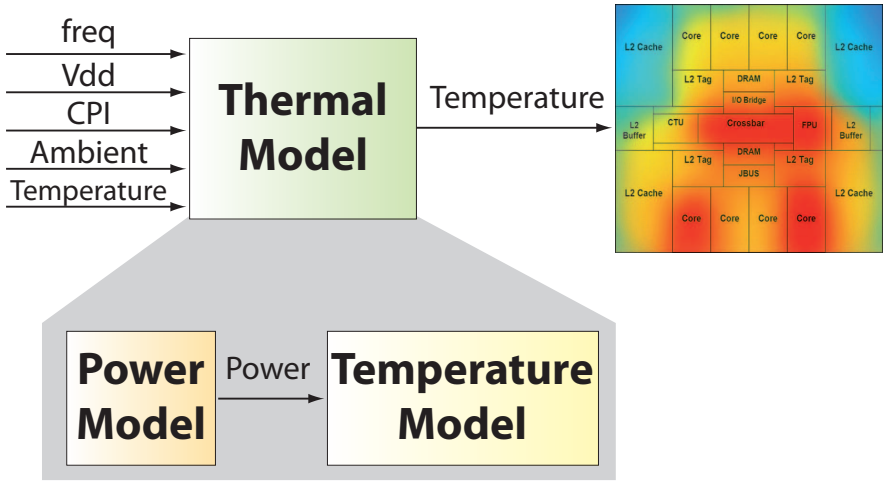


Figure B.4: Thermal Model.

It is widely known that the relation between the aforementioned inputs and the temperature is nonlinear. However, the nonlinearity affects only a part of the relation, the one between the frequency, voltage, CPI and the power consumption. Instead the function that correlates power and temperature is linear. For this reason, the thermal model can be obtained in two steps, first executing a nonlinear function to find the power, and then, a linear function to find the temperature, that is,

$$T = \mathcal{F}(P_{TOT}, T_{ambient}) = \mathcal{F}(\mathcal{P}(freq, CPI, V_{dd}), T_{ambient}) \tag{B.11}$$

where $\mathcal{F}(\cdot)$ and $\mathcal{P}(\cdot)$ are respectively a linear function and a nonlinear function, T is the temperature of the processor and $T_{ambient}$ is the ambient temperature. In detail, $\mathcal{P}(\cdot)$ is the Power Model we discussed in the previous section. Thus, we can assume to know it. Notice that the definition of an accurate Power Model is quite a hard task and represents a crucial issue in thermal control of MPSoCs. In the rest of the thesis, according to (B.10) and (B.9), we will consider $\mathcal{P}(\cdot)$ equal to the Power Model,

$$\begin{aligned} P_{TOT} &= P_{dynamic} + P_{static} = \\ &k_A \cdot freq \cdot V_{dd}^2 + k_B + (k_C + k_D freq) \cdot CPI^{k_E} + Z \cdot V_{dd} \cdot T^2 \cdot e^{\frac{-q \cdot V_t}{K \cdot T}} \end{aligned} \tag{B.12}$$

The $\mathcal{F}$ function, that we call Temperature Model according to the Fig. B.4, could be developed using different mathematical instruments. A partial differential equation may describe the heat flow inside a volume, as well as an analytical function where the parameters are identified from the measurements. We chose a finite element approach which guarantees a good precision and a relative low computational complexity (7) (8). We assumed the chip as a volume composed by two layers: a silicon layer that represents the die and a copper layer that represents the heat spreader. We decompose the layers in elementary cubic cells. Then exploiting the well-known duality between heat transfer and electrical phenomena we associate to each cell a RC circuit, as shown in Fig. B.5.

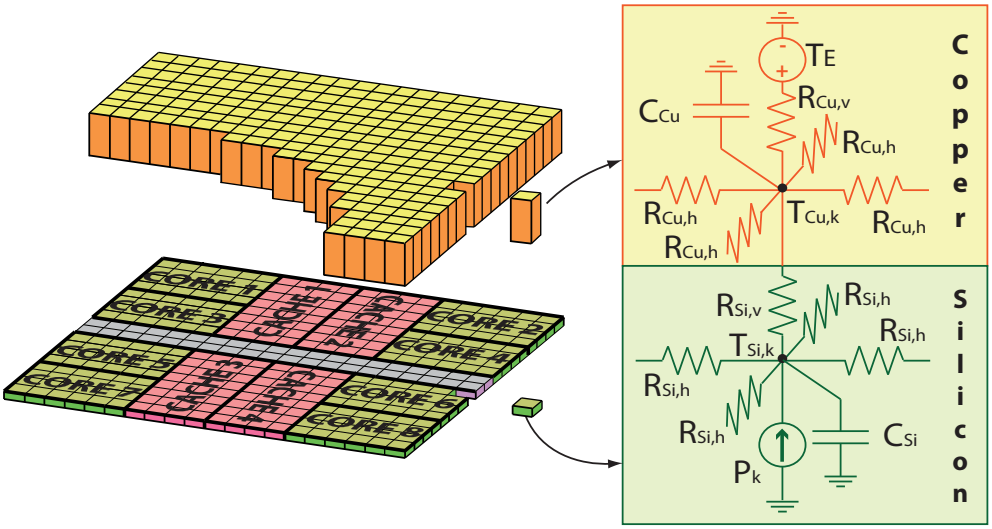


Figure B.5: Finite element approach: equivalent electric circuit.

The current is the heat flow, the voltage represents the temperature difference. R is the thermal resistance and C represents the thermal capacitance that models the transient behavior of the cells, i.e. the time necessary to reach the new temperature after the power input change.

Each cells is composed by a resistor for the vertical thermal dissipation (respectively $R_{Si,v}$, $R_{Cu,v}$ for the silicon and copper cells), four resistors for the horizontal thermal dissipation ($R_{Si,h}$, $R_{Cu,h}$), a capacitance (C_{Si}, C_{Cu}) and a current generator or a voltage generator depending on the belonging layer. The former represents the power dissipated by the active silicon cell, while the latter represents the ambient temperature close to the heat spreader.

Connecting the circuit of each cell to the neighbors we obtain the model:

$$\begin{aligned}\dot{T}_{Si,k} &= \frac{P_k}{C_{Si}} + \frac{T_{Cu,k} - T_{Si,k}}{C_{Si} \cdot R_{Si,v}} + \sum_{i=1}^{\#neighs} \frac{T_{Si,neigh,i} - T_{Si,k}}{2 \cdot C_{Si} \cdot R_{Si,h}} \\ \dot{T}_{Cu,k} &= \frac{T_{Si,k} - T_{Cu,k}}{C_{Cu} \cdot R_{Si,v}} + \frac{T_E - T_{Cu,k}}{C_{Si} \cdot R_{Cu,v}} + \sum_{i=1}^{\#neighs} \frac{T_{Cu,neigh,i} - T_{Cu,k}}{2 \cdot C_{Cu} \cdot R_{Cu,h}}\end{aligned} \tag{B.13}$$

where $T_{Si,k}$ and $T_{Cu,k}$ are respectively the temperatures of the $k-th$ cell of silicon and copper, and $T_{Si,neig,i}$ and $T_{Cu,neig,i}$ are respectively the neighbors of the $k-th$ silicon and copper cell.

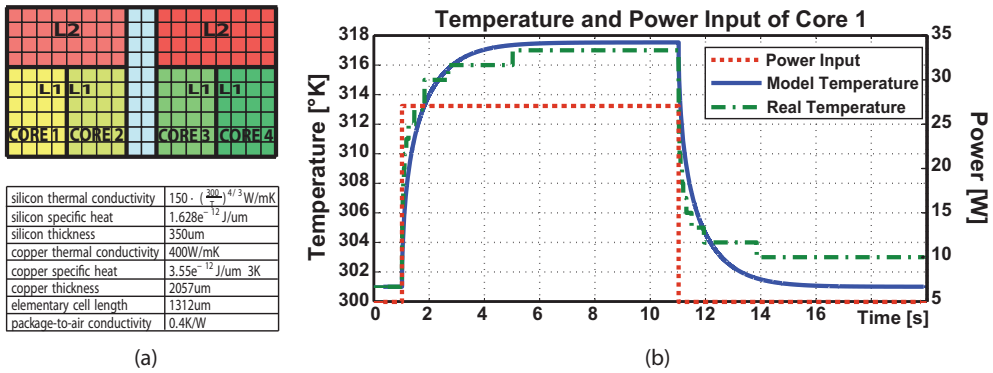
The model is linear and can be re-written as,

$$\begin{aligned}x(t+1) &= A\,x(t) + B\,P_{cell}(t) + B_{AMB}\,T_{AMB}(t) \\ T(t) &= C\,x(t)\end{aligned} \tag{B.14}$$

where $x(t)$ is the state vector containing all the cell temperatures at time t, A is the state matrix, B and B_{AMB} are the input matrix, P_{cell} is the input vector containing the power dissipation of each cell, T_{AMB} is the ambient temperature information, T is the vector containing the measured temperature, and C is the output matrix.

It is worth to note that, although x represents the temperature of all the cells in which the processor has been decomposed, only few of these values can be measured in a real processor. Indeed processors has few thermal sensors for monitoring chip temperatures usually placed in strategic positions. The C matrix selects a subset of temperatures which represent the measured temperatures. In this thesis we assumed to have one sensor per core placed in its center, imaging to find here the highest temperatures (there exist techniques to optimize the location of the sensors (9)). Another assumption of this thesis is that only cores are actuated, that is only the power dissipation of the cores can be directly modified. The other components on the chip, as cache, are indirectly controlled through the cores. Also this limitation is not dramatic because the highest power density are consumed on cores where usually the most dangerous thermal challenges occur.

As an example, the Fig. B.6a shows the approximative layout of the Xeon® X7350 (10) where only cache and cores has been considered. The parameters values, shown on the right, has been set according to the material properties and by comparing the temperature response of the model with the response of the real processor. Fig. B.6b shows the results of these tests for one core. The dash&dot line shows the temperature measurements, whereas the dashed curve shows the input power step. The parameters set allow us to find the C and R values of the equivalent circuit ($R_{Si,v} = 1.6^\circ K/W$, $R_{Cu,v} = 150^\circ K/W$, $R_{Si,h} = 22.9^\circ K/W$, $R_{Cu,h} = 1.2^\circ K/W$, $C_{Si} = 1e-3J/^\circ K$, $C_{Cu} = 1.2e-2J/^\circ K$).



silicon thermal conductivity	$150 \cdot (\frac{300}{T})^{4/3}$ W/mK
silicon specific heat	$1.628e^{-12}$ J/um
silicon thickness	350um
copper thermal conductivity	400W/mK
copper specific heat	$3.55e^{-12}$ J/um 3K
copper thickness	2057um
elementary cell length	1312um
package-to-air conductivity	0.4K/W

Figure B.6: Approx. Intel® Xeon® X7350 Floorplan.

Each core has been decomposed in 24 silicon cells and 24 copper cells. Moreover, on the chip can be present other components as for instance caches, even though these components cannot be directly controlled with frequency and voltage knobs. Frequency, workload, and voltage of the four cores have been converted into power dissipation using the Power Model as an interface function. The cores power feeds the Temperature Model which returns the temperatures of the measured cells. Fig. B.7 shows the temperature of all cells interpolated along space.

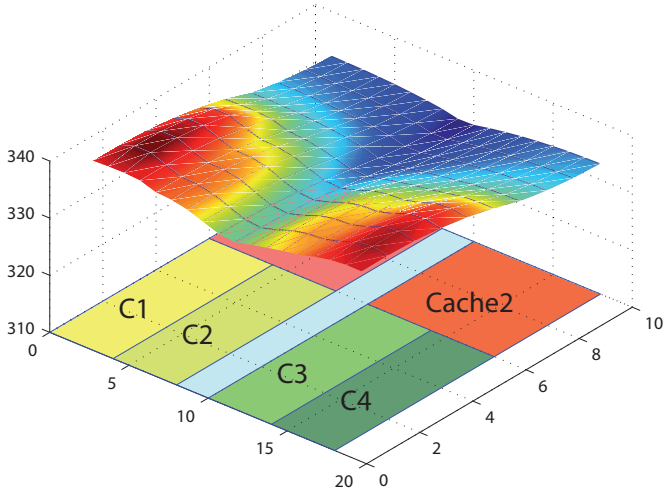


Figure B.7: Temperature map.

Finally, it is worth to note that the Power Model is different for each platform. Nevertheless, in this thesis we assumed the Power Model as a universal function valid for all the processors. Although this assumption is not technically correct, the control algorithms presented in this work are not affected by the nonlinear function that could be substituted with any others. However, in almost all the architectures we considered, we tried to keep the same proportion with the original architecture (e.g. duplicating the floorplan, scaling the dimensions and the parameters)

B.4.2 Simics Simulator

The Matlab/Simulink simulator has been used as first environment where to test our control solutions. However, this simulator presents different limitations: it relies on trace-driven simulations and it disregards the dependencies existing among power, thermal effect, reliability and performance. In order to take into account the interdependencies between control actions and workloads, we test our solutions on a full-system virtual platform. This platform relies on a established system simulator called Simics (6) where models for estimating the power consumption, the temperature distribution and the aging have been integrated (characterized from real hardware). Simics is a commercial instruction set simulator that models a complete multicore platform based on in-order x86 cores with memory, I/O interfaces and operating systems. We configured Simics to emulate the X7350 Intel® core which comprises four Pentium® 4 cores. Simics simulates each x86 instruction in one CPU clock time period. To account for memory latency, and different execution times for different instructions Simics loads a cycle-accurate memory timing-model called RUBY belonging to the GEMS (11) collection. During simulation, RUBY is called from each core before executing each memory access. It determines the latency of memory accesses and stalls the target core until the reading phase finishes. The RUBY and Simics cycle periods are the same. RUBY also contains a module for reading Simics performance counters every arbitrarily chosen N cycles. The extracted data refer to the number of instructions retired, the clock cycles and stall cycles expired, the halt instructions and other core events. Then, these data are used to compute the power consumption and the temperatures of the chip. To compute the CPU power consumption, the Power Model (B.8)-(B.9) has been integrated as a RUBY module. The power dissipated by memories can be obtained by multiplying the memory usage (extracted by counters) by the power estimated using CACTI in different working states. Also the thermal model has been integrated as a RUBY module. It is the finite element model showed in the previous Section and it takes as inputs the

powers computed by the power module. The simulation time required by the virtual platform to execute 1 billion instructions is 1240*s*.

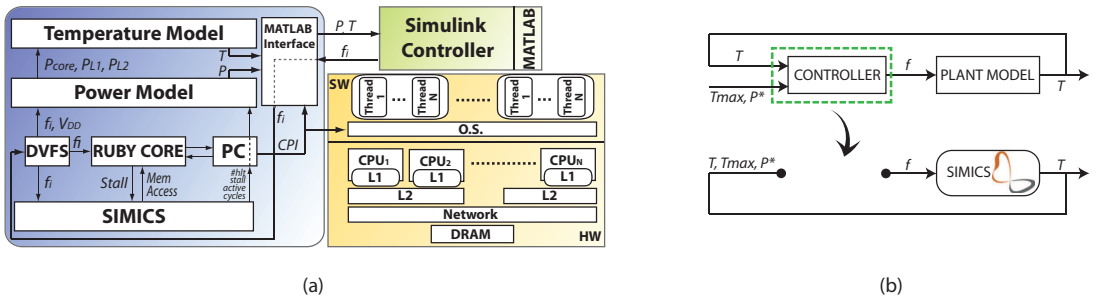


Figure B.8: (a) Virtual platform architecture; (b) Control development strategy.

The next step is to implement the control algorithm on the simulator previously described. In order to simplify the control algorithm integration in the virtual platform a RUBY module has been added to support the MATLAB engine library. This latter allows C programs to use Matlab as a computational engine. At initialization time the virtual platform starts the MATLAB engine process that executes concurrently to the simulator. Then, the Simulink controller model is loaded and initialized, and two communication channels are established between the RUBY module and the Simulink control algorithm. The first channel provides input to the control algorithm. The second one leads controller outputs to the target simulated system. At each sampling instant of the controller:

1. the Simulink controller initializes with the past internal states;
2. the performance counters are read and the data sent to the Matlab environment;
3. one step of Simulink simulation executes;
4. RUBY reads the data from Simulink (core frequencies);
5. Simulink saves the internal state

Fig. B.8a shows the architecture of the virtual platform. Instead, Fig. B.8b shows the steps required for the development of a control algorithm. First, the controller design is carried out in the MATLAB/Simulink framework providing preliminary tests and rapid design adjustments. Then, the tuned controller is directly interfaced with the virtual platform, exploiting the MATLAB/Simulink interfacing features. For more details on the virtual platform architecture refer to (3).

B.5 Performance

In this thesis we often use the term performance for assessing the goodness of a control algorithm. However, the definition of the performance metrics depends on the particular application the processor is used for. As an example in a data center we are interested in the average throughput, whereas in a smartphone it is far most important the responsiveness, that is to respond fast to a request provided by the user. The main performance metric we use in this work is the maximization of the frequency. Indeed, we expect that the higher is the core speed and the lower is the task execution time (it depends on the number of instructions in a program). Thus, considering DFS or DVFS techniques for thermal and power management, it is clear that these mechanisms necessarily affect performance, since impose a frequency decrease.

The usual way to measure performance is to use benchmarks – program specifically chosen to measure performance – that form the workload of cores (i.e. the set of program runs). In order to test our controller we chose a benchmark suite called PARSEC (Princeton Application Repository for Shared-Memory Computers) (12) that collects multithreaded programs. The suite focuses on emerging workloads and was designed to be representative of next-generation shared-memory programs for chip-multiprocessors. We select some corner-case benchmark:

Fluidanimate used for simulating the fluid dynamic for animation purposes with Smoothed Particle Hydrodynamics (SPH) method;

Facesim used for simulating the motions of a human face;

Bodytrack used for the body tracking of a person;

Raytrace used for real-time raytracing;

Dedup used for next-generation compression with data deduplication.

Since our simulator is not designed for taking as input tasks and threads, we profiled the frequency, the CPI, and the power consumption behaviors by running these benchmark on a real platform (the Intel® server system S7000FC4UR which runs four quad-core Xeon® X7350 processors at 2.93GHz and has a total memory capacity of 16GB based on FBDIMMs).

Fig. B.9 shows the traces obtained for the benchmark Fluidanimate for the core 1. In electronic devices the clock determines when events take place in the hardware. The discrete

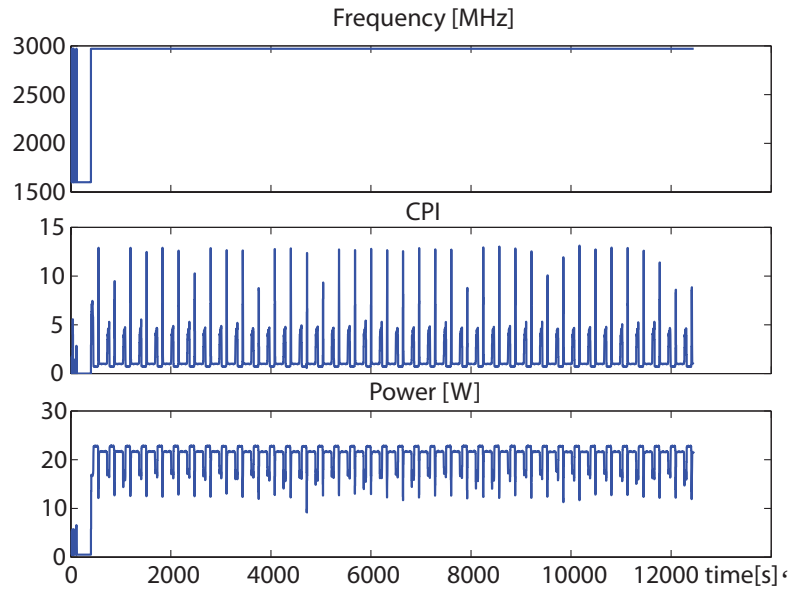


Figure B.9: Fluidanimate traces.

time interval are called *clock cycles* and the frequency of the processor is usually expressed as *clock rate* that is $\frac{1}{clock\,period}$. The execution time of a program can be expressed as,

$$Execution\,time = Clock\,cycles \cdot frequency$$

The number of clocks constituting a program can be achieved by multiplying the total number of instructions by the average clock cycles per instruction (13). The average time each instruction takes to execute is defined as clock cycles per instruction (CPI). Depending on the type of instruction the CPI varies (i.e. the lower is the CPI the lower are the accesses to memory). In this thesis the CPI is computed as the ratio between the clock un-halted cycles at reference frequency and the instruction retired in the observed period. This metric expresses an instantaneous workload measurement.

B.6 The SCC platform

The Single-chip Cluod Computer (SCC) (14) is a 48-core experimental processor created by Intel Labs. It supports on-chip message passing application, Networks-on-Chip (NoCs) communications, and DVFS mechanisms. It is implemented in 45 nm high-K metal-gate CMOS and it contains 1.3 billion transistors in a total die area of $567mm^2$.

Architecture. The cores are P54C CPU with x86 architecture. Each of them belongs to a tile that can accommodate two cores. The tiles are arranged in a 6×4 grid. On each core we can boot an operating system (Linux 2.6.38 kernel) which works independently from the others. Each core has private L1 and L2 caches (16KB and 256KB respectively). Cache coherence is managed through a software protocol as opposed to commonly used hardware protocols. Each tile has a 54-ports, high-speed, and low latency router to connect with 4 neighboring routers and set a 2D-mesh on-die network. According to Fig. B.10 each tile has also a Message Passing Buffer (MPB) used to increase performance of a message passing programming model whereby cores communicate through local shared memory. Tile performance is scalable from 300 MHz at 700 mV to 1.3 GHz at 1.3 V. The on-chip network scales from 60 MHz at 550 mV to 2.6 GHz at 1.3 V. The design target for nominal usage is 1 GHz for tiles and 2 GHz for the 2-D network, when supplied by 1.1 V. Each tile in the SCC contains two ring-oscillator based thermal sensors

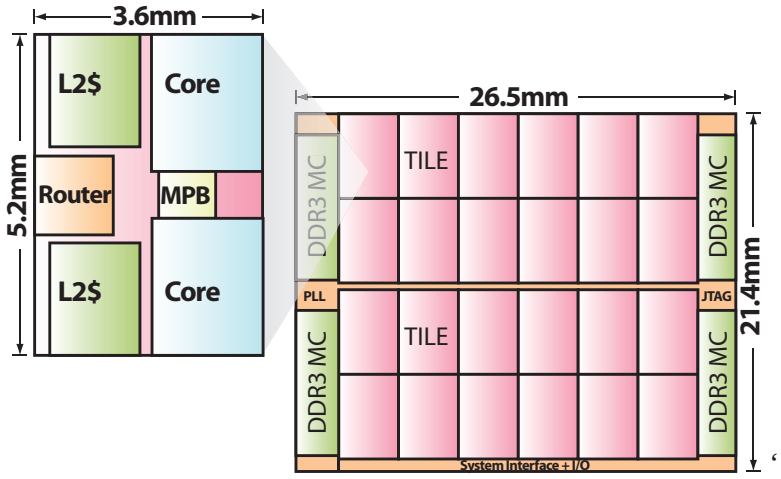


Figure B.10: SCC architecture.

Message Passing. The SCC includes an on-chip message passing application framework, named RCCE, which is a lightweight message passing library developed by Intel and optimized for SCC. It uses the hardware MPB to send and receive messages preventing the use of the network layer abstraction and the TCP/IP protocol overhead for exchanging messages among different physical cores. The library uses the two primitives *put* and *get* to efficiently move data respectively from the L1 cache of one core to the MPB of another one, and from the MPB to the L1 cache of the same core.

DVFS. The processor presents 8 Voltage Islands and 28 Frequency Islands managed with software-based DVFS techniques. The voltage islands are controlled by a VCR (Voltage Regulator Controller) that contains two voltage regulators and it is addressable by every core (Voltage range: $[0,1.3]V$ with $6.25mV$ steps). Two voltage islands supply the 2D-mesh and die periphery, with the remaining 6 voltage islands being divided among the core area. 24 out of the 28 frequency islands are associated to the tiles, one for the 2D-mesh and three for the system interface, VRC, and memory controllers, respectively. Unlike voltage changes, frequency can be changed faster (20ns vs about 1ms for voltage changes).

Bibliography

[1] P.F. Butzen, R.P. Ribas, *Leakage Current in Sub-Micrometer CMOS Gates*, 2010, http://www.inf.ufrgs.br/logics/docman/book_emicro_butzen.pdf 210

[2] L. Torres, P. Benoit, G. Sassatelli, M. Robert, F. Clermidy, D. Puschini, *An Introduction to Multi-Core System on Chip Trends and Challenges*, in M. Hubner, J. Becker, *Multiprocessor System-on-Chip: Hardware design and tool integration*, Springer, 2011. 211

[3] A. Bartolini, *Dynamic Power Management: from portable devices to high performance computing*, Ph.D. dissertation, University Of Bologna, Italy, 2011. 211, 212, 220

[4] H. F. Hamann, A. Weger, J. A. Lacey, Z. Hu, P. Bose, E. Cohen, J. Wakil, *Hotspot-Limited Microprocessors: Direct Temperature and Power Distribution Measurements*, IEEE Journal of Solid-State Circuits,Vol. 4, pp. 5665, Jan. 2007.

[5] The MathWorks. MATLAB & Simulink. http://www.mathworks.com/. 214

[6] Virtutech. Virtutech Simics. http://www.virtutech.com/. 214, 219

[7] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, D. Tarjan, *Temperature Aware Microarchitecture*, in Proc. IEEE/ACM ISCA, San Diego, CA, USA, pp.213, 2003. 216

[8] G. Paci, P. Marchal, F. Poletti, L. Benini. *Exploring "temperature-aware" design in low-power MPSoCs*, In Proc. DATE 06, Vol. 1, pp. 180, 2006. 216

[9] S. Reda, R.J. Cochran, A. Nazma Nowroz, *Improved Thermal Tracking for Processors Using Hard and Soft Sensor Allocation Techniques*, IEEE Trans. Comput., Vol. 60, pp. 841-851, 2011. 217

[10] N. Sakran et al. The implementation of the 65nm dual-core 64b merom processor. In IEEE International Solid-State Circuits Conference, 2007. 217

[11] Martin Milo M. K. et al. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset. SIGARCH Comput. Archit. News, 33(4):92-99, 2005. 219

[12] C. Bienia, S. Kumar, J.P. Singh, K. Li, *The PARSEC Benchmark Suite: Characterization and Architectural Implications*, PACT, 2008. 221

[13] D.A. Patterson, J.L. Hennessy *Computer Organization and Design: the hardware/Software interface*, 4th edition, Morgan Kaufmann Publishers, 2009. 222

[14] J. Howard et al., *A 48-core ia-32 processor in 45 nm cmos using ondie message-passing and dvfs for performance and power scaling*, IEEE Journal of Solid-State Circuits, Vol. 46(1):173183, Jan. 2011. 222

BIBLIOGRAPHY

Appendix C

Accurate Model

In this Appendix part of the code used for simulations is presented.

C.1 The plant

In this Section is reported the Matlab code used to create the accurate thermal model (the simulator) for testing our control algorithms.

The code has been split in part in order to simplify the comprehension to the reader.

C.1.1 Global parameters

```
clear all
clc

%% 0
        %VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
        %||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
        %%%%%%%%%%%%%%%%%%%%%%%%%%                      %%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%   GLOBAL PARAMETER   %%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%                      %%%%%%%%%%%%%%%%%%%%%%%%%%%
        %||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
        %AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

N_CORE = 8;                 % Number Of Cores
N_COMP = 20;                % Number Of Components (Cores,Caches,Empty Areas)

% Set the type of thermal model
%Type  = 'Reduced1L';               % 1 Layer 1 cell per core
%Type  = 'Reduced2L';               % 2 Layer 1 cell per core
Type  = 'Full2L';                   % 2 Layer many cell per core

```

```
% Layout definition
Filename_FLOORPLAN='floorplan8.txt';    % floorplan
Filename_SENSORS='sensors8.txt';        % Sensor location
Filename_HOTSPOT='powers8.txt';         % power distribution

% Each core has 4x6 cells each of them with dimension 1312x1312
Chip_Dimensions.h=5248+5248+2624+5248+5248;                    % Height
Chip_Dimensions.L=7872+5248+5248+7872;                         % Width

% Ambient temperature  (in the case of POD set 0
Tenvironment =310;     % [K]
```

In this first part the global parameters are defined. The user must set the number of cores and the total number of components belonging to the chip. As an example the processor presented in Fig. C.1 has 8 cores and 20 components. Notice that it is not possible to have components with more than a neighbor on the same edge. In this case it is required to split the single real component in more parts. Each cache in figure has two neighbor cores on the same side, hence we need to split the cache in two parts.

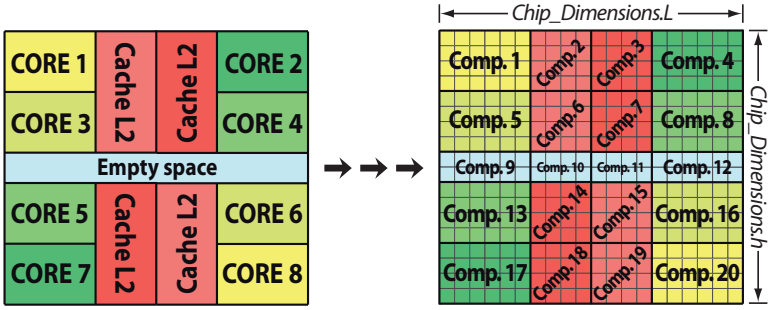


Figure C.1: Layout definition

Then, the user must select the type of model he wants to build. The approach used to build the model lies on the finite element decomposition described in Appendix B (the processor volume is split in cells each of which is associated to an equivalent electric RC circuit). The program allows the user to create:

- models with 1 layer and 1 cell per component;
- models with 2 layers and 1 cell per components (total 2 cells);
- models with 2 layers and many cells per components;

Since we need an extremely accurate model and the accuracy increase with the number of the cells we decided for the third alternative. Notice that the first two solutions are obtained

from the third solution making the parallel of all the vertical resistances of the cells composing a component, the parallel of the horizontal cells linking the component to the neighbor component and neglecting the horizontal resistances between the cell inside the components.

The layout of the processors is described inside 3 textual files the names of which are assigned to the variables: *Filename_FLOORPLAN*, *Filename_SENSORS*, *Filename_HOTSPOT*. Finally, the dimensions of the chip and the initial ambient temperature are defined.

C.1.1.1 *Layout_Files_Generation.m*

```
%% 0
        %VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
        %||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
        %%%%%%%%%%%%%%%%%%%%%%                              %%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%       USER DEFINITIONS       %%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%                              %%%%%%%%%%%%%%%%%%%%%%%
        %||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
        %AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

% Cells per core
core_cells_num_height = 4;      % <---------------- Modifiable by the user
core_cells_num_width = 6;       % <---------------- Modifiable by the user

% Cells per cache
cache_cells_num_height = 4;     % <---------------- Modifiable by the user
cache_cells_num_width = 4;      % <---------------- Modifiable by the user

% Cells subsampling (e.g. if subsplit_height=2 then core=4x6 --> core=8x12)
subsplit_height = 1;            % <---------------- Modifiable by the user
subsplit_width = 1;             % <---------------- Modifiable by the user

% Matrices of ones with the dimensions of the cores and caches components
core = ones(core_cells_num_height*subsplit_height,
    core_cells_num_width*subsplit_width);
cache = ones(cache_cells_num_height*subsplit_height,
    cache_cells_num_width*subsplit_width);

% Sensor position in cores and caches (1 where there is the sensor, 0 otherwise)
core_sens = zeros(core_cells_num_height*subsplit_height,
    core_cells_num_width*subsplit_width);
core_sens(2*subsplit_height, 3*subsplit_width) = 1;
                  % <---------------- Modifiable by the user
cache_sens = zeros(cache_cells_num_height*subsplit_height,
    cache_cells_num_width*subsplit_width);  % <---------------- Modifiable by the
    user

% Power distribution in cores and caches
% The total power of the component is divided by the maximum number
```

```
% assigned to the cells and the results is divided by the number of cells
% with the same number to find the single cell power consumption
% Example: core=[1 1 1 1 1;1 2 3 2 1;1 1 1 1 1] --> 1 cells have a power
% equal to Pow_TOT/3/12, P(2 cells)=Pow_TOT/3/2, and P(2 cells)=Pow_TOT/3/1
core_hot = ones(core_cells_num_height*subsplit_height,
    core_cells_num_width*subsplit_width);
core_hot(1*subsplit_height+1:3*subsplit_height, 1*subsplit_width+1:5*subsplit_width)
    = 2;        % <---------------- Modifiable by the user
cache_hot = ones(cache_cells_num_height*subsplit_height,
    cache_cells_num_width*subsplit_width);  % <---------------- Modifiable by the
    user

% Floorplan definition using previously defined patterns
floorplan=[core 2*cache 3*cache 4*core;
           5*core 6*cache 7*cache 8*core;
           9*ones(2*subsplit_height,core_cells_num_width*subsplit_width)
    10*ones(2*subsplit_height,cache_cells_num_width*subsplit_width)
    11*ones(2*subsplit_height,cache_cells_num_width*subsplit_width)
    12*ones(2*subsplit_height,core_cells_num_width*subsplit_width);
           13*core 14*cache 15*cache 16*core;
           17*core 18*cache 19*cache 20*core];          % <----------------
    Modifiable by the user

% Sensors Layout definition using previously defined sensors patterns
sensors=[core_sens cache_sens cache_sens core_sens;
           core_sens cache_sens cache_sens core_sens;
           zeros(2*subsplit_height,20*subsplit_width);
           core_sens cache_sens cache_sens core_sens;
           core_sens cache_sens cache_sens core_sens];  % <----------------
    Modifiable by the user

% Power Distribution definition using previously defined power patterns
power=[core_hot cache_hot cache_hot core_hot;
           core_hot cache_hot cache_hot core_hot;
           ones(2*subsplit_height,20*subsplit_width);
           core_hot cache_hot cache_hot core_hot;
           core_hot cache_hot cache_hot core_hot];      % <----------------
    Modificabile da utente


%% 1
        %VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
        %|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
        %%%%%%%%%%%%%%%%%%%%%%%                              %%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%       WRITING OF FILES       %%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%                              %%%%%%%%%%%%%%%%%%%%%%%
        %|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
        %AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA


% Floorplan
floorplan_filename='floorplan8.txt';
```

```
75
76  fp1 = fopen(floorplan_filename, 'w');
77  for i=1:size(floorplan,1)
78      for j=1:size(floorplan,2)
79          fprintf(fp1, strcat(num2str(floorplan(i,j)),' \t'));
80      end
81      fprintf(fp1, '\n');
82  end
83  fclose(fp1);
84
85
86
87  % Sensors Layout
88  sensors_filename='sensors8.txt';
89
90  fp2 = fopen(sensors_filename, 'w');
91  for i=1:size(sensors,1)
92      for j=1:size(sensors,2)
93          fprintf(fp2, strcat(num2str(sensors(i,j)),' \t'));
94      end
95      fprintf(fp2, '\n');
96  end
97  fclose(fp2);
98
99
100
101 % Power Distribution
102 power_filename='powers8.txt';
103
104 fp3 = fopen(power_filename, 'w');
105 for i=1:size(power,1)
106     for j=1:size(power,2)
107         fprintf(fp3, strcat(num2str(power(i,j)),' \t'));
108     end
109     fprintf(fp3, '\n');
110 end
111 fclose(fp3);
```

These are the instructions used to generate the three files previously mentioned (shown in Fig. C.2). In the first part of the code the number of cells composing the cores and of the caches are defined. Then the matrices *floorplan*, *sensors*, and *power* are created by the user. Each element of the matrix is associated to a single cell of the silicon layer. The *floorplan* matrix assigns the cells to the components by numbering them. The cells belonging to the same component have the same number. The numeration of the components is from the left to the right and from the top to the bottom. The *sensors* matrix localizes the sensors position. A cell contains a temperature sensor if the correspondent value in the matrix is set to 1 otherwise its

value is 0. The *power* matrix instead defines the power distribution in the components. Wether the power consumption of the whole component is equally dissipated by all the cells then the elements of the matrix associated to these cells have the number 1, otherwise we discern different areas with different power consumption by numbering the cells with increasing value. Then, the total power is equally divided by the number of the defined areas and subsequently by the number of cells belonging to that area in order to find the single cell power consumption. To better understand this mechanism, let us consider the core 1 in Fig. C.2. The number of areas in the power matrix of core 1 is 2. This means that each region consumes:

$$Pow_{region\,1} = \frac{Pow_{TOT}}{2} \qquad Pow_{region\,2} = \frac{Pow_{TOT}}{2}$$

The power dissipated by each cell can be obtained as,

$$Pow_{cells\,1} = \frac{Pow_{region\,1}}{16} \qquad Pow_{region\,2} = \frac{Pow_{region\,2}}{8}$$

In the second part of the code the matrices are saved in a text file.

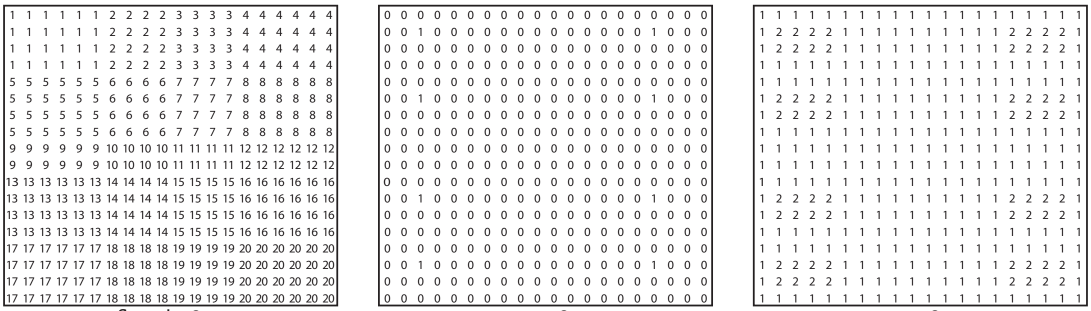


Figure C.2: Layout files generated by the function *Layout_Files_Generation.m*

C.1.2 Input Pattern Generation

```
%% 1
        %VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
        %|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
        %%%%%%%%%%%%%%%%%%%%%%                              %%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%   INPUT PATTERN GENERATION   %%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%                              %%%%%%%%%%%%%%%%%%%%%%%
        %|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
        %AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

% Simulation time information
```

```
Time = struct();
Time.Start = 0;                                         % [s]
Time.End = 30;                                          % [s]
Time.Points = 30000;                                    % Number of points
Time.FrameRate=20;                                      % frame per sec (only for 3D vis.)
Time.Step = (Time.End - Time.Start)/Time.Points;        % Step
Time.array = (Time.Start:Time.Step:Time.End)';          % Time vector

% Input vector with environmental temperature
Tenv = ones(length(Time.array),1)*Tenvironment;

% Power Input initialization
clear PowerCPU Power PowerCache
Power = zeros(length(Time.array),N_COMP);
PowerCPU = zeros(length(Time.array),N_CORE);

% Input cores power: Steps
% PowerCPU(:,1)=[zeros(floor(Time.Points/18),1); 30*ones(floor(Time.Points/18),1); zeros(floor(Time.Points/18*16)+3,1)];
% PowerCPU(:,2)=[zeros(floor(Time.Points/18*3),1); 30*ones(floor(Time.Points/18),1); zeros(floor(Time.Points/18*14)+2,1)];
% PowerCPU(:,3)=[zeros(floor(Time.Points/18*5),1); 30*ones(floor(Time.Points/18),1); zeros(floor(Time.Points/18*12)+2,1)];
% PowerCPU(:,4)=[zeros(floor(Time.Points/18*7),1); 30*ones(floor(Time.Points/18),1); zeros(floor(Time.Points/18*10)+3,1)];
% PowerCPU(:,5)=[zeros(floor(Time.Points/18*9),1); 30*ones(floor(Time.Points/18),1); zeros(floor(Time.Points/18*8)+2,1)];
% PowerCPU(:,6)=[zeros(floor(Time.Points/18*11),1); 30*ones(floor(Time.Points/18),1); zeros(floor(Time.Points/18*6)+2,1)];
% PowerCPU(:,7)=[zeros(floor(Time.Points/18*13),1); 30*ones(floor(Time.Points/18),1); zeros(floor(Time.Points/18*4)+3,1)];
% PowerCPU(:,8)=[zeros(floor(Time.Points/18*15),1); 30*ones(floor(Time.Points/18),1); zeros(floor(Time.Points/18*2)+2,1)];

% Input cores power: PRBS
PowerCPU = [zeros(Time.Points/4,N_CORE);idinput([Time.Points*3/4+1,N_CORE],'prbs',[0 0.5],[0 25])];


% Computation of the power dissipated by caches (30% of adjacent cores)
percentage=0.6;
PowerCache(:,1) = ((PowerCPU(:,1)+PowerCPU(:,3))./2)*percentage;
PowerCache(:,2) = ((PowerCPU(:,2)+PowerCPU(:,4))./2)*percentage;
PowerCache(:,3) = ((PowerCPU(:,5)+PowerCPU(:,7))./2)*percentage;
PowerCache(:,4) = ((PowerCPU(:,6)+PowerCPU(:,8))./2)*percentage;

% Total power inputs (all components)
Power=[PowerCPU(:,1) PowerCache(:,1)./2 PowerCache(:,2)./2 PowerCPU(:,2)
    PowerCPU(:,3) PowerCache(:,1)./2 PowerCache(:,2)./2 PowerCPU(:,4)
    zeros(Time.Points+1,4) PowerCPU(:,5) PowerCache(:,3)./2 PowerCache(:,4)./2
    PowerCPU(:,6) PowerCPU(:,7) PowerCache(:,3)./2 PowerCache(:,4)./2 PowerCPU(:,8)];
```

In this part of the code we described the input pattern applied to the thermal model.
First, the parameters concerning the time of the traces are specified. The user must insert the values in a structure containing the following values:

Start: initial time in seconds;

End: stop time in seconds;

Points: the number of points;

FrameRate: the number of frame per second (used for the 3D visualization).

The time step between two sampling intervals and the array containing all the sampling instants are computed automatically from the previously defined values.
The ambient temperature is defined as an array. Each element correspond to a sampling instants (we assumed the ambient temperature constant in this case).
Then, it is specified the input power trace. Notice that this trace is used to simulate the temperature of the processors. Subsequently, identification approaches are applied on the collected temperature data in order to make the thermal model treatable by the controller. Thus, at this point of the code it is unnecessary to define the inputs as frequency and CPI traces and then convert them into power traces.
The power trace (*Power* in the code) is a matrix with a number of column equal to the number of components and a number of row equal to the number of sampling instants. First, we defined the power consumption of the cores (*PowerCPU*) as a PRBS input ranging from $0W$ to $25W$. Then, for the sake of simplification, we defined the power consumption of the caches (*PowerCache*) as the 30% of the power consumed by the adjacent core. Therefore, as an example, at every sampling interval the power consumption of the cache number 1 is equal to the mean of the power consumption of the cores number 1 and 3 multiplied for the 30%. Finally, the *PowerCPU* and the *PowerCache* matrices are combined to obtain the final *Power vector* (the empty components have zero power)

C.1.3 Thermal Model Generation

```
%% 2
        %VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
        %||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
        %%%%%%%%%%%%%%%%%%%%%%                              %%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%   THERMAL MODEL GENERATION   %%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%                              %%%%%%%%%%%%%%%%%%%%%%%
        %||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
        %AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

% Accurate Thermal model creation
Continuous_Discrete = -1;               % 1 discrete time, -1 continuous time
[A,B,C,D] = mat_modeling(Filename_FLOORPLAN, Filename_SENSORS, Filename_HOTSPOT,
    Chip_Dimensions, Type, Continuous_Discrete);
TModel = ss(A,B,C,D);                   % SS object creation
X0 = ones(size(A,1),1)*Tenvironment;    % Initial condition

% Continuous time simulation
Temp = lsim(TModel, [Power,Tenv], Time.array, X0);

% Modification of C to measure all the states
TModelX = ss(TModel.a,TModel.b,eye(size(TModel.a,1)), zeros(size(TModel.a,1),
    N_COMP+1));
x_plant = lsim(TModelX,[Power,Tenv], Time.array,X0);


%%%%%%%%%%%%%%%%%%%%%%%%   3D visualization  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Setting Parameters
data.OUT=x_plant(:,1:size(x_plant,2)/2);
data.IN=PowerCPU;
visual.OUT.xmin=1;
visual.OUT.xmax=20;
visual.OUT.ymin=1;
visual.OUT.ymax=18;
visual.OUT.zmin=310;
visual.OUT.zmax=460;
visual.OUT.x=1;
visual.OUT.y=1;
visual.OUT.z=1;
visual.IN.xmin=0;
visual.IN.xmax=2;
visual.IN.ymin=0;
visual.IN.ymax=4;
visual.IN.zmin=0;
visual.IN.zmax=30;
visual.IN.x=10;
visual.IN.y=-10;
visual.IN.z=10;

% visualization function call
Visualization3D(data,Time,visual,1)
```

In order to build the accurate thermal model we exploit the function *mat_modeling.m* that returns the classical state matrix A, the input matrix B, the output matrix C and the feedforward matrix D. The matrices are then used to create a SS Matlab object.
Then, the initial temperature of the cells has been set equal to the initial ambient temperature.
Finally, the model is simulated using the *lsim* function. The function returns as output the temperatures of the cells measured by the sensors usually located in the silicon layer (the lower) as specified in the *Filename_SENSORS* file. In our case the sensor are 8, one for each core.
In this part of code there are also the instructions to modify the model in order to obtain all the temperature values of the states (this is useful for the identification based on POD approach where we assume to have an accurate model of the thermal system before reducing its size) and to have a 3D visualization of the processor temperature distribution.

C.1.3.1 *mat_modeling.m*

```
function [A,B,C,D]=mat_modeling(FILENAME_FLOORPLAN, FILENAME_SENSORS,
    FILENAME_HOTSPOT, CHIP_DIMENSIONS, TYPE,TS)
% MAT_MODELING. Generate A, B, C, D matrices.
% The function take as input 6 parameters:
%
%   - FILENAME_FLOORPLAN: the string of the text file containing the floorplan
%
%   - FILENAME_SENSORS: the string of the text file containing the location of
    sensors
%
%   - FILENAME_HOTSPOT: the string of the text file containing the distribution of
    powers
%
%   - CHIP_DIMENSIONS: a structure with 2 fields:
%           ->  h = Height of the chip (um)
%           ->  L = Width of the chip (um)
%
%   - TYPE: the string containing the type of the model:
%           ->  'Full2L' for a model with high number of cell and 2 layers
%           ->  'Full2LNL' for a model with high number of cell and 2 layers, but
    non linear
%           ->  'Reduced2L' for a reduced model with two cells for each core and 2
    layers
%           ->  'Reduced1L' for a reduced model with one cells for each core and 1
    layer
%
%   - TS: the sampling time in seconds:
%           -> -1 to have continuous matrices (default)
%
%
```

```
% Example
%   [a,b,c,d]=mat_modeling('floorplan.txt','sensors.txt','hotspots.txt', chip_size,
    'Full2L',1e-3);
%
% NOTES: this function uses these functions:
%   fine2L_linear    coarsegrain2L      coarsegrain1L   Full2LNL   discretization
%


%% 0
    %|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
    %%%%%%%%%%%%%%%%%%%%%%%%%                                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%          PARAMETERS             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%            CHECK                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%                                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||

if nargin<5,
    error('model:mat_modeling:none','The function needs more parameters');
end

if ~isa(FILENAME_FLOORPLAN,'char'),
    error('model:mat_modeling:input','Floorplan file name must be a string.');
end


if ~isa(FILENAME_SENSORS,'char'),
    error('model:mat_modeling:input','Sensors file name must be a string.');
end


if ~isa(FILENAME_HOTSPOT,'char'),
    error('model:mat_modeling:input','Power Distribution file name must be a
    string.');
end


if ~isa(CHIP_DIMENSIONS,'struct'),
    error('model:mat_modeling:input','the fourth parameter must be a struct with 2
    fields .h and .L.');
elseif (~isa(CHIP_DIMENSIONS.L,'numeric') || ~isa(CHIP_DIMENSIONS.h,'numeric'))
    error('model:mat_modeling:input','the fields of the fourth parameter must be
    numerics.');
end


if ~isa(TYPE,'char'),
    error('model:mat_modeling:input','Model type must be a string.');
end


if nargin<6 || isempty(TS),
```

C. ACCURATE MODEL

```
    TS=-1;
    warning('model:nlmodel:default',' Matrices are calculated for a continuous
    model');
end


%% 1
    %|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
    %%%%%%%%%%%%%%%%%%%%%%%%                                %%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%            MODEL               %%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%            DATA                %%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%                                %%%%%%%%%%%%%%%%%%%%%%%%%%
    %|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||

% Parameters
clear TM

% Reading of the floorplan file
fp1=fopen(FILENAME_FLOORPLAN,'r');
k=0;
row = fgets(fp1);
while ischar(row)
    k=k+1;
    TM.components_layout(k,:)=sscanf(row,'%f\t')';
    row = fgets(fp1);
end
fclose(fp1);

% Reading of the sensors file
fp2=fopen(FILENAME_SENSORS,'r');
k=0;
row = fgets(fp2);
while ischar(row)
    k=k+1;
    TM.sensors_layout(k,:)=sscanf(row,'%f\t')';
    row = fgets(fp2);
end
fclose(fp2);

% Reading of the power distribution file
fp3=fopen(FILENAME_HOTSPOT,'r');
power_distribution=zeros(size(TM.components_layout,1),size(TM.components_layout,2));
k=0;
row = fgets(fp3);
while ischar(row)
    k=k+1;
    power_distribution(k,:)=sscanf(row,'%f\t')';
    row = fgets(fp3);
end
fclose(fp3);


```

```
124 % Chip properties
125 TM.cells_num_height=size(TM.components_layout,1);
126 TM.cells_num_width=size(TM.components_layout,2);
127 TM.layer_num=2;
128
129 % Cells dimensions
130 TM.cell_height_Si=CHIP_DIMENSIONS.h/size(TM.components_layout,1);   %[um]
131 TM.cell_width_Si=CHIP_DIMENSIONS.L/size(TM.components_layout,2);    %[um]
132 TM.cell_thick_Si=350;                                              %[um]
133 TM.cell_height_Cu=CHIP_DIMENSIONS.h/size(TM.components_layout,1);   %[um]
134 TM.cell_width_Cu=CHIP_DIMENSIONS.L/size(TM.components_layout,2);    %[um]
135 TM.cell_thick_Cu=2057;                                             %[um]
136
137 TM.c_Si=1.628e-12;                  % Si specific heat  [j/(K*um^3)]
138 TM.c_Cu=3.55e-12;                   % Cu specific heat  [j/(K*um^3)]
139
140 % Kelvin degree between heat spreader and ambient for dissipating 1 Watt (package
        data)
141 KperW=0.4;
142 % Heat exchange coefficient with the ambient [W/(K*um^2)] =1.041667e-7;
143 TM.env_sup=1/KperW/TM.cells_num_height/TM.cells_num_width/ TM.cell_height_Cu/
        TM.cell_width_Cu;
144
145 TM.thermal_conductivity_Si=1.25e-4;      % Silicon thermal conductivity  W/(K*um)
146 TM.thermal_conductivity_Cu=4e-4;         % Copper thermal conductivity   W/(K*um)
147
148 TM.components_num=max(max(TM.components_layout));
149
150
151 if ((size(TM.components_layout,1)~=size(TM.sensors_layout,1))||
        (size(TM.components_layout,2)~= size(TM.sensors_layout,2)))
152     error('Sensors and Floorplan files have different dimensions')
153 end
154
155 if ((size(TM.components_layout,1)~=size(power_distribution,1))||
        (size(TM.components_layout,2)~= size(power_distribution,2)))
156     error('Distribution Power and Floorplan files have different dimensions')
157 end
158
159 %% 2
160     %||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
161     %%%%%%%%%%%%%%%%%%%%%%%%%                                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
162     %%%%%%%%%%%%%%%%%%%%%%%%%          MATRICES              %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
163     %%%%%%%%%%%%%%%%%%%%%%%%%                                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
164     %||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
165
166 switch lower(TYPE)
167     case {'full2l'}
168         [A,B,C,D]=fine2L_linear(TM);
169
170     case {'reduced2l'}
171         [A,B,C,D]=coarsegrain2L(TM);
```

```
172
173     case {'reduced1l'}
174         [ACG_plant,BCG_plant,CCG_plant,DCG_plant]=coarsegrain2L(TM);
175         [A,B,C,D]=coarsegrain1L(TM,ACG_plant,BCG_plant);
176
177     otherwise
178         disp('Unknown method. Possible choices: Full2L - Reduced2L - Reduced1L')
179 end
180
181 % Matrix B modification for account the inhomogeneity of power distribution
182 for j=1:TM.components_num
183     support_matrix1=TM.components_layout==j;
184     support_matrix2=support_matrix1.*power_distribution;
185     maximum=max(max(support_matrix2));
186     for i=1:TM.cells_num_height*TM.cells_num_width
187         if B(i,j)~=0
188             B(i,j)=B(i,j)*sum(sum(support_matrix1))/maximum/
      sum(sum(support_matrix2==support_matrix2(floor((i-1)/ TM.cells_num_width)+1,
      mod((i-1), TM.cells_num_width)+1)));
189         end
190     end
191 end
192
193 % Discretization if required
194 if TS>0
195     [A,B,C,D]=discretization(A,B,C,D,TS);
196 end
```

The function takes as inputs the names of the files (*FILENAME_FLOORPLAN*, *FILENAME_SENSORS*, *FILENAME_HOTSPOT*) used to define the layout of the processor, the chip dimensions (*CHIP_DIMENSIONS*), and the variables that describe respectively the desired model type (*TYPE*) and the temporal characteristic (*TS*) of the model (discrete-time continuous-time). The function gives as output the matrices used to describe the linear model in the state-space formalism.

The first part checks if the input parameters are correct. In the second part the layout file are read and the useful parameters are collected in the structure *TM* (e.g. the cells number, the cells dimensions, the silicon and copper thermal conductivity). Finally, in the third part the function *fine2L_linear* is called to create the model. The function *discretization* converts the model from continuous-time to discrete-time if requested, and the matrix *B* is modified to account for the information contained in the power distribution file.

C.1.3.2 *fine2L_linear.m*

```
function [A,B,C,D]=fine2L_linear(TM)
% FINE2L_LINEAR generates the model matrices
%
% It takes as inputs
%       - cells_num_height: number of cells along y axe
%       - cells_num_width: number of cells along x axe
%       - layer_num: number of cell layers
%       - components_num: number of components
%       - cell_height_Si: height of the silicon cell
%       - cell_width_Si: width of the silicon cell
%       - cell_thick_Si: thickness of the silicon cell
%       - cell_height_Cu: height of the copper cell
%       - cell_width_Cu: width of the copper cell
%       - cell_thick_Cu: thickness of the copper cell
%       - c_Si: silicon specific heat
%       - c_Cu: copper specific heat
%       - env_sup: heat exchange coefficient with the ambient
%       - thermal_conductivity_Si: silicon thermal conductivity
%       - thermal_conductivity_Cu: copper thermal conductivity
%       - components_layout: components layout
%       - sensors_layout: sensors layout
%

%% 1
    %|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
    %%%%%%%%%%%%%%%%%%%%%%%%                                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%           MODEL                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%           DATA                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%                                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||


cells_num_height = TM.cells_num_height;
cells_num_width = TM.cells_num_width;
layer_num = TM.layer_num;
Si_cells_num = cells_num_height*cells_num_width;
components_num = TM.components_num;
input_num = components_num+1;                          % +1 for the T_amb;

cell_height_Si = TM.cell_height_Si;                    %[um]
cell_width_Si = TM.cell_width_Si;                      %[um]
cell_thick_Si = TM.cell_thick_Si;                      %[um]
cell_height_Cu = TM.cell_height_Cu;                    %[um]
cell_width_Cu = TM.cell_width_Cu;                      %[um]
cell_thick_Cu = TM.cell_thick_Cu;                      %[um]

c_Si = TM.c_Si;                     %calore specifico silicio  j/(K*um^3)
c_Cu = TM.c_Cu;                     %calore specifico rame     j/(K*um^3)

env_sup = TM.env_sup; %W/(K*um^2) %coefficiente di scambio del calore con l'ambiente
    W/(K*um^2)
```

C. ACCURATE MODEL

```
51
52  thermal_conductivity_Si = TM.thermal_conductivity_Si; % Silicon thermal conductivity
        [W/(K*um)]
53  thermal_conductivity_Cu = TM.thermal_conductivity_Cu; % Copper thermal conductivity
        [W/(K*um)]
54
55  components_layout = TM.components_layout;
56  sensors_layout = TM.sensors_layout;
57
58  thermal_capacity_Si = c_Si*cell_height_Si*cell_width_Si*cell_thick_Si;
        % [j/K]
59  thermal_capacity_Cu  =c_Cu*cell_height_Cu*cell_width_Cu*cell_thick_Cu;
        % [j/K]
60  G_Si_vertical=thermal_conductivity_Si*cell_width_Si*cell_height_Si/cell_thick_Si;
        % [W/K]
61  G_Si_horizontal=thermal_conductivity_Si*cell_width_Si*cell_thick_Si/cell_height_Si;
        % [W/K]
62  G_Cu_vertical=(thermal_conductivity_Cu*cell_width_Cu*cell_height_Cu/cell_thick_Cu)*
        (env_sup*cell_width_Cu*cell_height_Cu)/
        ((thermal_conductivity_Cu*cell_width_Cu*cell_height_Cu/cell_thick_Cu)+
        (env_sup*cell_width_Cu*cell_height_Cu)); % [W/K]
63  G_Cu_horizontal=thermal_conductivity_Cu*cell_width_Cu*cell_thick_Cu/cell_height_Cu;
        % [W/K]
64
65  disp('Horizontal resistance (Si/Cu):')
66  disp(1/G_Si_horizontal)
67  disp(1/G_Cu_horizontal)
68
69  disp('Vertical resistance (Si/Cu):')
70  disp(1/G_Si_vertical)
71  disp(1/G_Cu_vertical)
72
73  disp('Thermal Capacity (Si/Cu):')
74  disp(thermal_capacity_Si)
75  disp(thermal_capacity_Cu)
76
77
78  %% 2
79      %|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
80      %%%%%%%%%%%%%%%%%%%%%%%%                                %%%%%%%%%%%%%%%%%%%%%%%%
81      %%%%%%%%%%%%%%%%%%%%%%%%          MATRICES              %%%%%%%%%%%%%%%%%%%%%%%%
82      %%%%%%%%%%%%%%%%%%%%%%%%                                %%%%%%%%%%%%%%%%%%%%%%%%
83      %|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
84
85
86  Z=ones(cells_num_height,cells_num_width);
87  Z=[zeros(cells_num_height,1) Z zeros(cells_num_height,1)];
88  Z=[zeros(1,cells_num_width+2);Z;zeros(1,cells_num_width+2)];
89
90  %------------------------------>   A   <-----------------------------------
91  A=zeros(cells_num_height*cells_num_width*layer_num);
92
```

```
% A Matrix layer 1 (Silicon)
for i=2:(cells_num_height+1)
    for j=2:(cells_num_width+1)
        neighbors_counter=0;
        if (Z(i-1,j)==1)
            neighbors_counter=neighbors_counter+1;
            A(cells_num_width*(i-2)+j-1,cells_num_width*(i-3)+j-1)= G_Si_horizontal/
    (2*thermal_capacity_Si);
        end
        if (Z(i+1,j)==1)
            neighbors_counter=neighbors_counter+1;
            A(cells_num_width*(i-2)+j-1,cells_num_width*(i-1)+j-1)= G_Si_horizontal/
    (2*thermal_capacity_Si);
        end
        if (Z(i,j-1)==1)
            neighbors_counter=neighbors_counter+1;
            A(cells_num_width*(i-2)+j-1,cells_num_width*(i-2)+j-2)= G_Si_horizontal/
    (2*thermal_capacity_Si);
        end
        if (Z(i,j+1)==1)
            neighbors_counter=neighbors_counter+1;
            A(cells_num_width*(i-2)+j-1,cells_num_width*(i-2)+(j))= G_Si_horizontal/
    (2*thermal_capacity_Si);
        end
        A(cells_num_width*(i-2)+j-1, cells_num_width*(i-2)+j-1)=
    -(neighbors_counter* G_Si_horizontal/ (2*thermal_capacity_Si))-
    (G_Si_vertical/thermal_capacity_Si);
        A(cells_num_width*(i-2)+j-1, cells_num_width*(i-2)+j-1+Si_cells_num)=
    G_Si_vertical/ thermal_capacity_Si;
    end
end


% A Matrix layer 2 (Copper)
for i=2:(cells_num_height+1)
    for j=2:(cells_num_width+1)
        neighbors_counter=0;
        if (Z(i-1,j)==1)
            neighbors_counter=neighbors_counter+1;
            A(cells_num_width*(i-2)+j-1+Si_cells_num,
    cells_num_width*(i-3)+j-1+Si_cells_num)= G_Cu_horizontal/
    (2*thermal_capacity_Cu);
        end
        if (Z(i+1,j)==1)
            neighbors_counter=neighbors_counter+1;
            A(cells_num_width*(i-2)+j-1+Si_cells_num,
    cells_num_width*(i-1)+j-1+Si_cells_num)= G_Cu_horizontal/
    (2*thermal_capacity_Cu);
        end
        if (Z(i,j-1)==1)
            neighbors_counter=neighbors_counter+1;
```

```
133             A(cells_num_width*(i-2)+j-1+Si_cells_num,
        cells_num_width*(i-2)+j-2+Si_cells_num)= G_Cu_horizontal/
        (2*thermal_capacity_Cu);
134         end
135         if (Z(i,j+1)==1)
136             neighbors_counter=neighbors_counter+1;
137             A(cells_num_width*(i-2)+j-1+Si_cells_num,
        cells_num_width*(i-2)+(j)+Si_cells_num)= G_Cu_horizontal/
        (2*thermal_capacity_Cu);
138         end
139         A(cells_num_width*(i-2)+j-1+Si_cells_num,
        cells_num_width*(i-2)+j-1+Si_cells_num)= -(neighbors_counter* G_Cu_horizontal/
        (2*thermal_capacity_Cu))- (G_Cu_vertical/thermal_capacity_Cu)-
        (G_Si_vertical/thermal_capacity_Cu);
140         A(cells_num_width*(i-2)+j-1+Si_cells_num, cells_num_width*(i-2)+j-1)=
        G_Si_vertical/ thermal_capacity_Cu;
141     end
142 end
143
144
145 %----------------------------->   B   <----------------------------------
146 B=zeros(cells_num_height*cells_num_width*layer_num,input_num);
147 % Parameter that links the power consumption input and the silicon temeperature
148 coefficient_1c=1/thermal_capacity_Si;
149 % Parameter that links the power consumption input and the silicon temeperature
150 coefficient_2c=G_Cu_vertical/thermal_capacity_Cu;
151
152 % Compute the ratio 1/(cells number for each component)
153 for i=1:components_num
154     ratio(i)=1/length(find(components_layout==i));
155 end
156
157 % power contribution to B
158 for i=1:cells_num_height
159     for j=1:cells_num_width
160         B((i-1)*cells_num_width+j,components_layout(i,j))=
        ratio(components_layout(i,j))* coefficient_1c;
161     end
162 end
163
164 % Ambient temperature contribution to B
165 for i=Si_cells_num+1:(Si_cells_num*layer_num)
166     B(i,input_num)=coefficient_2c;
167 end
168
169
170 %----------------------------->   C e D   <---------------------------------
171 % C matrix
172 k=0;
173 n_sens=sum(sum(sensors_layout));
174 C=zeros(n_sens,Si_cells_num);
175 for i=1:cells_num_height
```

```
176     for j=1:cells_num_width
177         if sensors_layout(i,j)==1,
178             k=k+1;
179             C(k,cells_num_width*(i-1)+j)=1;
180         end
181     end
182 end
183 C=[C zeros(n_sens,Si_cells_num)];
184 
185 % D matrix
186 D=zeros(n_sens,input_num);
```

The only input of the function is the record *TM*. The values extracted from its fields are used to compute the conductances and the thermal capacities of the silicon and copper cells. Then the matrices *A*, *B*, *C*, *D* are build according to the equations (B.13) of Appendix B.

C.1.3.3 *discretization.m*

```
1  function [Ad,Bd,Cd,Dd]=discretization(A,B,C,D,TS)
2  % DISCRETIZATION discretizes the system defined by A, B, C, D
3  %
4  % It takes as inputs:
5  %       - A,B,C,D: the 4 matrices that defines the linear model;
6  %       - TS: the sampling time used for the discretization.
7  %
8  % Example
9  %   [a,b,c,d]=discretization(A,B,C,D,1e-3)
10 %
11 if TS<=0
12     error('model:discretization:time','Sampling time argument not valid');
13 end
14 
15 
16 system=ss(A,B,C,D);
17 discrete_system=c2d(system,TS,'zoh');
18 
19 %-------------------------------------------------------------------------> AdCG
20 Ad=discrete_system.a;
21 
22 %-------------------------------------------------------------------------> Ad
23 Bd=discrete_system.b;
24 
25 %-------------------------------------------------------------------------> Ad
26 Cd=discrete_system.c;
27 
28 %-------------------------------------------------------------------------> Ad
29 Dd=discrete_system.d;
```

The function discretizes the continuous-time model (the only input of the function).

C.1.3.4 *Visualization3D.m*

```
function []=Visualization3D(data,Time,visual,PowVis)
% VISUALIZATION3D: it shows the temperature variation during time with a 3D
% visualization.
%
% Input parameters:
%
% data: structure containing the output and input data (data.OUT, data.IN)
%
% Time: structure containing the initial time (Time.Start), the final time
% (Time.End), the time step (Time.Step) and the number of frame per second
% (Time.FramRate)
%
% visual: structure containing the xmin, xmax, ymin, ymax, zmin, zmax of
% the  input and output data, and the x, y, z coordinates of the camera:
% visual.OUT.xmin, visual.OUT.xmax, visual.OUT.ymin, visual.OUT.ymax,
% visual.OUT.zmin, visual.OUT.zmax, visual.OUT.x, visual.OUT.y,
% visual.OUT.z, visual.IN.xmin, visual.IN.xmax, visual.IN.ymin,
% visual.IN.ymax, visual.IN.zmin, visual.IN.zmax, visual.IN.x
% visual.IN.y, visual.IN.z
%
% PowVis: 0 for visualizing only output, 1 for visualizing input and output


time.Points = floor((Time.End - Time.Start)/Time.Step);
time.FR=floor((1/Time.Step)/Time.FrameRate);

% Traccia 1 solo Sens
screensize=get(0, 'ScreenSize');
f1=figure;
set(f1, 'Position', [0 0 screensize(3) screensize(4) ] );


for i=1:time.FR:time.Points
    if PowVis>0
        subplot(121),
    else
        subplot(111),
    end
    MatPlot=reshape(data.OUT(i,:),visual.OUT.xmax,visual.OUT.ymax);
    sur=mesh(1:1:visual.OUT.ymax,1:1:visual.OUT.xmax,MatPlot);
    axis([ visual.OUT.ymin visual.OUT.ymax visual.OUT.xmin visual.OUT.xmax
    visual.OUT.zmin visual.OUT.zmax]);
    view([visual.OUT.x,visual.OUT.y,visual.OUT.z])
    title(strcat('Time: ', '  ', num2str(i*Time.Step),'s'));
    drawnow

```

```
46      if PowVis>0
47          subplot(122),
48          MatPow=reshape(data.IN(i,:),visual.IN.xmax,visual.IN.ymax);
49          MatPow=MatPow';
50          h=bar3(MatPow);
51          for j = 1:length(h)
52              zdata = get(h(j),'ZData');
53              set(h(j),'CData',zdata)
54              % Add back edge color removed by interpolating shading
55              set(h,'EdgeColor','k')
56          end
57          axis([ visual.IN.xmin visual.IN.xmax+1 visual.IN.ymin visual.IN.ymax+1
      visual.IN.zmin visual.IN.zmax]);
58          view([visual.IN.x,visual.IN.y,visual.IN.z])
59          title('Distributed MPC frequency response');
60          xlabel('cores');ylabel('cores');zlabel('Core Frequency [MHz]')
61          drawnow
62      end
63
64  end
```

For completeness we reported the 3D visualization function. The first input parameter is a record containing the data to be visualized (*data*). It has two fields: the input data and the output data. Both are stored as matrices, where the number of columns is equal to the number of inputs or the outputs respectively, whereas the rows are the value sampled at each time interval. The parameter *Time* is a structure containing the time information of the simulation data (the initial time *Time.Start*, the final time *Time.End*, the time step *Time.Step* and the number of frame per second *Time.FramRate*). Also the *visual* parameter is a record which contains the visualization settings decided by the user. Finally *PowVis* allows the users to choose if visualizing simply the outputs o both the outputs and the inputs.
The body of the function is a loop that at each iteration,

1. scans the data line by line;
2. reshapes each line as specified in the *visual* parameters;
3. plots the values.

C.2 The thermal model identification

In this Section we reported the code used to reduce the order of the accurate thermal model. The procedures are explained in Chapter 4 and consist in the *distributed ARX identification*

solution, the H_∞ *identification solution* and the *POD-based solution*.

C.2.1 distributed ARX identification

As shown in the code below the ARX identification approach is obtained by calling the function *MPSoC_Id_Distr.m*. The goal is to obtain a set of single-core models, which takes as inputs the power consumption, the ambient temperature, and the neighbors temperatures, returning as output the future core temperature.

```
%% 3
        %VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
        %||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
        %%%%%%%%%%%%%%%%%%%%%%                              %%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%    SYSTEM IDENTIFICATION     %%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%                              %%%%%%%%%%%%%%%%%%%%%%%
        %||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
        %AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

% Parameters
model_order=2;                          % order for each single-core model
ARX_Method=1;                           % Identification solver
Cores_Deployment = [1 2; 3 4; 5 6; 7 8];  % relative cores position

% Identification
models=MPSoC_Id_Distr(Cores_Deployment, PowerCPU, Tenv, Temp, Time.Step,
    model_order, ARX_Method);
```

First some parameters have been defined. The *model_order* represents the desired model order for each single-core model. The ARX identification procedure allows the user to choose the identification algorithm to use for finding the models. Actually two methods are implemented, the first (*ARX_Method*=1) uses the arx function of Matlab, whereas the second (*ARX_Method*=2) uses an ad-hoc approach implemented with CVX toolbox. The first method has resulted more efficient than the second one. For this reason we will show only the first one. The *Cores_Deployment* matrix, instead, contains the information about the relative position of the cores. In the current case the core 5 has as neighbors the cores 3, 6, and 7.
Finally, the identification procedure is called.

C.2.1.1 *MPSoC_Id_Distr.m*

```
function Models= MPSoC_Id_Distr(Deployment, Power, Tenvi, Temperature, Tsampling,
    ModelOrder, varargin)
% MPSoC_Id_Distr returns a structure containing the single-core models
```

```
%
% Inputs parameters:
%
%   - Deployment : a map of the core
%   - Power : a matrix with all the power data of the cores
%   - Tenvi : the environment temperature data
%   - Temperature : the temperature data of the cores
%   - Tsampling : the sampling time
%   - ModelOrder : the desired order of the model
%   - varargin :
%       -> parameter 1: put 1 for Matlab ARX identification method or 2 for CVX
%       -> parameter 2: string containing the destination path where to
%       save the models
%
% Example:
% Models = MPSoC_Id_Distr(Deployment, Power, Tenv, Temperature, Tsampling,
    ModelOrder, 1,'path')
%
% Notes: The size of the matrices b are equalt to the cores temperatures + 2

%% 0

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%         CHECK PARAMETERS
    %%%%%%%%%%%%%%%%%%%%%%%%%%

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Method and path management
if isempty(varargin)
    Method=1;
    save_model=0;
else
    save_model=0;
    if isempty(varargin{1})
      Method=1;
   else
       if ischar(varargin{1})
           error('Libreria:MPSoC_Id_Distr:METHOD','The method variable must be a
    number');
       else
           if varargin{1}==1
               Method=1;
           else
               Method=varargin{1};
               disp('CVX method will be used')
           end
       end
    end

    % To save the single-core model to the path specified in varargin{2}
    if length(varargin)==2
```

```
50          if isempty(varargin{2})
51              save_model=0;
52          else
53              if isnumeric(varargin{2})
54                  error('Libreria:MPSoC_Id_Distr:ADDRESS','The address must be a
    string')
55              else
56                  save_model=1;
57                  cd_old=cd;
58                  cd(varargin{2})
59                  mkdir('model_data');
60                  address=strcat(varargin{2},'\model_data\');
61                  cd(cd_old);
62              end
63          end
64      end
65  end
66
67  % Number of cores
68  n_core=0;
69  for i=1:size(Deployment,1)
70      for j=1:size(Deployment,2)
71          if Deployment(i,j)>n_core
72              n_core=Deployment(i,j);
73          end
74      end
75  end
76
77  % Understanding of the neighborhood relation
78  k=1;
79  neighbors=zeros(n_core);
80  Deployment_ext=[zeros(1,size(Deployment,2)+2); zeros(size(Deployment,1),1)
        Deployment zeros(size(Deployment,1),1);zeros(1,size(Deployment,2)+2)];
81  while k<=n_core
82      for i=2:size(Deployment_ext,1)-1
83          for j=2:size(Deployment_ext,2)-1
84              if ((Deployment_ext(i-1,j)~=k) && (Deployment_ext(i-1,j)~=0) &&
    (Deployment_ext(i,j)==k))
85                  neighbors(k,Deployment_ext(i-1,j))=1;
86              end
87              if ((Deployment_ext(i+1,j)~=k) && (Deployment_ext(i+1,j)~=0) &&
    (Deployment_ext(i,j)==k))
88                  neighbors(k,Deployment_ext(i+1,j))=1;
89              end
90              if ((Deployment_ext(i,j-1)~=k) && (Deployment_ext(i,j-1)~=0) &&
    (Deployment_ext(i,j)==k))
91                  neighbors(k,Deployment_ext(i,j-1))=1;
92              end
93              if ((Deployment_ext(i,j+1)~=k) && (Deployment_ext(i,j+1)~=0) &&
    (Deployment_ext(i,j)==k))
94                  neighbors(k,Deployment_ext(i,j+1))=1;
95              end
```

```
        end
    end
    k=k+1;
end

%% 1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%          IDENTIFICATION          %%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Iterative procedure repeated for each core
for k=1:n_core
    clc;
    disp('Identification of the model');disp(k);
    j=0;
    TempIn(:,1)=Tenvi;  % TempIn contains the ambient and the neighbors temperatures
    for i=1:n_core
        if neighbors(k,i)==1
            j=j+1;
            TempIn(:,j)=Temperature(:,i);
        end
    end

    % Identification: function call (Method=1 --> ARX, CVX otherwise)
    if Method==1
        mod=SCI(Power(:,k),Tenvi,TempIn,Temperature(:,k),Tsampling,ModelOrder);
    else
        mod=SCI_cvx(Power(:,k),Tenvi,TempIn,Temperature(:,k),Tsampling,ModelOrder);
    end

    % Change of coordinates to have C=[1 0 .. 0]
    model_ARX_MISO=give_physics(mod);

    % B matrix extension for the sake of future operation convenience:
    % inclusion of all the temperatures as neighbors inputs
    % (the unused inputs have correspondent B elements zeroed)
    b(:,1:2)=model_ARX_MISO.b(:,1:2);
    j=2;
    for i=1:n_core
        if neighbors(k,i)==1
            j=j+1;
            b(:,i+2)=model_ARX_MISO.b(:,j);
        else
            b(:,i+2)=zeros(size(model_ARX_MISO.b,1),1);
        end
    end
    model_ARX_MISO.b=b;
    model_ARX_MISO.d=zeros(1,n_core+2);

```

C. ACCURATE MODEL

```
    % Computation of the initial state:
    %  - Case 1: initial state computed assuming the model in equilibrium
    %            with the ambient temperature, the initial power equal to
    %            0 and the neighbors temperature equal to the ambient one
    %  - Case 2: initial equilibrium different from the previous one. The
    %            initial power, hte initial neighbors temperature, and the
    %            core temperature is equal to the first sampled data.
    states='syms';
    ics=' x';
    for i=1:(ModelOrder-1)
        states=strcat(states,strcat(ics,num2str(i)));
    end
    eval(states)
    for i=1:(ModelOrder-1)
        xeq2(i,1)=eval(strcat(ics,num2str(i)));
    end

    if (Temperature(1,k)-1>(Tenvi(1,1)))
        % Case 2
        init_state=Temperature(1,k);
      init_power=Power(1,k);
        init_Tneigh=Temperature(1,:)';
    else
        % Case 1
        init_state=Tenvi(1,1);
        init_power=zeros(1,1);
        init_Tneigh=Tenvi(1,1)*ones(n_core,1);
    end
    % initial state estimation
   xeq=[init_state;xeq2];
   equilibrium= (model_ARX_MISO.a- eye(size(model_ARX_MISO.a,1)))*xeq+
    model_ARX_MISO.b* [init_power;Tenvi(1,1);init_Tneigh];
      solution=solve(equilibrium(1:(ModelOrder-1)));
    if ModelOrder==2
        x02=double(solution);
    else
        ics='double(solution.x';
        x02=zeros((ModelOrder-1),1);
        for i=1:(ModelOrder-1)
            x02(i,1)=eval(strcat(ics,num2str(i),')'));
        end
    end
   model_ARX_MISO.x0=[init_state;x02];     % Initial state of the model

    % Gain matrix estimation for Luenberger observer
    eigenvalues=eig(model_ARX_MISO.a)*0.4;
    model_ARX_MISO.k_obsv=(place(model_ARX_MISO.a', model_ARX_MISO.c'
    ,eigenvalues))';

    % The k-th single-core model is saved to the address indicated
    if (save_model==1)
        cd(address)
```

```
195         eval(['Mod' num2str(k) '=model_ARX_MISO;']);
196         eval(['save(strcat(address,''Mod' num2str(k) '''),''Mod' num2str(k) ''');']);
197         cd(cd_old);
198     end
199
200     % Sampling time
201     model_ARX_MISO.Ts=Tsampling;
202
203
204     % The k-th single-core model assigned to the output structure
205     eval(['Models.m' num2str(k) '=model_ARX_MISO;']);
206
207
208     clear TempIn mod model_ARX_MISO xeq2 init_Tneigh
209     disp('Done')
210 end
```

This function takes as inputs the deployment of the cores, the data achieved by simulating the accurate thermal model (the power consumptions and the temperature of all cores, and the ambient temperature), the sampling time used for collecting the data, the desired model order for the final models, and two optional inputs that are the method used for solving the ARX identification algorithm and the address where to save the model. By default the algorithm is solved using the *arx* function of Matlab and the models are not saved. The output of the function consists of a record whose fields are described below.

$$models.\begin{cases} m1.\begin{cases} a \\ b \\ c \\ d \\ k\_obsv \\ Ts \\ x0 \end{cases} \\ \vdots \\ mN \end{cases}$$

where $m1,\ldots,mN$ are the models, a, b, c, d are the model matrices, k_obsv is the gain matrix of the Luenberger observer computed by using the *place* Matlab function, Ts is the sampling time, and $x0$ is the initial state of the model.

In the first part the input parameters are checked in order to save the ARX method to be applied and the address where to save the model if present. Then, for each core, the relation of proximity with the other cores is described with the *neighbors* matrix. It is a square matrix with dimension equal to the number of cores. The row number indicates the considered i-th core

and the columns are the possible neighbors. If the core j is a neighbor of the i-th core then the element (i, j) is equal to 1, 0 otherwise (the same applies to the element (j,i)).
In the second part, a loop executes the following operations for each core,

1. the neighbors temperature data and the ambient temperature data are collected in *TempIn*;
2. the *SCI.m* function is called to solve the ARX algorithm;
3. the *give_physics.m* function is called to change the coordinate of the state space model returned by the *SCI.m* function, in order to have the first state that represents the measured core temperature;
4. the input matrix b is expanded in order to take as inputs all the neighbor temperatures (the temperature of the cores that do not belong to the neighborhood are associated to 0 coefficients in the matrix b);
5. the initial state of the model is computed according to two cases: (i) the model is in equilibrium when the initial power is 0 and the initial temperature of all cores are equal to the ambient temperature, (ii) the model is in equilibrium when the initial power and the initial temperature of all cores are equal to the first sample of the data used for identification;
6. the gain matrix of the Luenberger observer is computed placing the eigenvalues of the matrix $(a + k\_obsv \cdot c)$ at the values obtained by multiplying the eigenvalues of a by 0.4;
7. the model is saved and assigned to the output record.

C.2.1.2 *SCI.m*

```
1  function model=SCI(MyPow,Tenv,NeighTemp,MyTemp,T_sampling,Degree,varargin)
2  % SCI Single Core Identification
3  % The function takes as input six parameters:
4  %       - MyPow : the power of the core;
5  %       - Tenv : the environment temperature;
6  %       - NeighTemp : the neighbors temperature;
7  %       - MyTemp : the temperature of the core;
8  %       - T_sampling : the sampling time;
9  %       - Degree : the order of the identified model;
10 %       - varargin: visualization mode(0=nothing, 1=simulator, 2=predictor, 3=white
       test).
11 %
12 % The result is a SS object.
```

```
%
% Example
%   model=SCI(MyPow(:,1),Tenv(:,1),NeighTemp(:,1:2),MyTemp(:,1),1e-3,2,2);
%

%% 0

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%        CHECK PARAMETERS
    %%%%%%%%%%%%%%%%%%%%%%%%%%%

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    if size(MyPow,2)>1
    error('SCI:input','MyPow size mismatch')
end
if size(MyTemp,2)>1
    error('SCI:input','MyTemp size mismatch')
end
if size(Tenv,2)>1
    error('SCI:input','Tenv size mismatch')
end
if ((isempty(varargin)) || (varargin{1}>3))
    visu_mode=0;
else
    visu_mode=varargin{1};
end

% Adjusting data
num_neighbor=size(NeighTemp,2);
Out=MyTemp;
In=[MyPow Tenv NeighTemp];
data=iddata(Out,In,'Ts',T_sampling);
data.OutputName = {'T'};
data.OutputUnit = {'K'};

%% 1

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%         IDENTIFICATION
    %%%%%%%%%%%%%%%%%%%%%%%%%%%

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% ARX function call
model1=arx(data,'na',Degree,'nb',Degree*ones(1,num_neighbor+2),'Focus','Prediction');
% model1=arx(data,'na', Degree, 'nb', Degree*ones(1,num_neighbor+2), 'nk',
    ones(1,num_neighbor+2)); % Equivalent to the upper one
% model1=pem(data,'na', Degree, 'nb', Degree*ones(1,num_neighbor+2), 'Focus',
    'Prediction');
% model1=pem(data,'na', Degree, 'nb', Degree*ones(1,num_neighbor+2), 'SearchMethod',
    'lm', 'Tolerance',sqrt(eps));
```

```

switch visu_mode
  case 1
    disp('Comparison between real data and simulated data');
    compare(data,model1);
    pause();
    close;
  case 2
    disp('Comparison between real data and predicted data');
    compare(data,model1,1);
    pause();
    close;
  case 3
    disp('White test on the error as predictor')
    [yh,fit,x_init]=compare(data,model1,1);
    er=Out-yh{1}.OutputData;
    disp('White test result:');
    disp(wtest(er));
    plot(1:size(er,1),er);title('Residual')
    pause();
  otherwise
    disp('No visualization selected.')
end


%% 2
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%        STATE-SPACE MODEL
    %%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Input/Output model convertion into a state-space form
for i=1:size(model1.b,1)
    matrix(:,i)=tf(model1.b(i,:),model1.a,model1.Ts);
end
model=ss(matrix,'min');
```

The *SCI.m* takes as inputs the power consumption (*MyPow*), the temperature (*MyTemp*) and the temperature of the neighbors (*NeighTemp*) of the *k*-th core. Moreover, it takes the ambient temperature data (*Tenv*), the value of the sampling time (*T_sampling*) according to with the data are collected, the order of the searched model (*Degree*) and as optional the visualization mode. The output is the model in a state-space form.

In the first part the function parameters are checked and the data are collected in a Matlab *IDDATA* data object opportunely divided into inputs and outputs. The visualization mode is assigned to the variable *visu_mode*.

In the second part of the code, the Matlab *arx* function is called. It takes as input the data previously managed, the order of the polynomials related to the inputs and the outputs. The commented instructions represent other functions to obtain the same result. As a results we obtain a input-output model of the form,

$$a(q) \cdot y(t) = B(q) \cdot u(t) + e(t)$$

where *a* and *b* are polynomials contained as fields in the record *model1*. According to the *visu_mode* variable the data obtained using the model as a simulator or a predictor can be plotted and compared to the real data (*visu_mode=1* and *visu_mode=2*, respectively). Setting *visu_mode=3* it is possible to plot the residual error between the predicted and real data and the result of the white test.
In the third part the model is converted in the state-space form.

C.2.1.3 *give_physics.m*

```
function model=give_physics(mdl)
% GIVE_PHYSICS: The function transforms the ss-model identified with SCI
% function in a State-Space model with C matrix in the form [I_n | 0_n]
% where n is the number of cores
% The function take as input one parameters:
%       - mdl : the model identified with SCI function;
%
% The result is a structure. The A,B,C,D matrices are held in a,b,c,d
%
%
% Example
%   m=give_physics(SCI_model);
%

% Observability matrix
T = (obsv(mdl))^-1;

% Change coordinate
model.a=T^-1*mdl.a*T;
model.b=T^-1*mdl.b;
model.c=mdl.c*T;
model.d=zeros(size(model.c,1),size(model.b,2));
```

The function uses the observability matrix, *T*, to change the coordinate of the state-space model obtained from the *SCI.m* function. As a result the first state of the new model correspond to the measured temperature, indeed *C* has the form $[1\,0\ldots0]$ where the number of zeros depends on the desired model order.

C.2.2 H_∞ identification

The H_∞ identification approach relies on the *MPSoC_Id_Hinf.m* function. The code below is used in the main program to call the previously mentioned function that returns as output the set of single-core models collected in a record structure.

```
%% 3
        %VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
        %||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
        %%%%%%%%%%%%%%%%%%%%%%                              %%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%    SYSTEM IDENTIFICATION     %%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%                              %%%%%%%%%%%%%%%%%%%%%%%
        %||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
        %AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

% Parameters
model_order=2;                            % order for each single-core model
Cores_Deployment = [1 2; 3 4; 5 6; 7 8];  % relative cores position

% Identification
models=MPSoC_Id_Hinf(Cores_Deployment, PowerCPU, Tenv, Temp, Time.Step, model_order);
```

First the order of the single-core models (*model_order*) and the cores relative position (*Core_Deployment*) are defined, then the function is called.

C.2.2.1 *MPSoC_Id_Hinf.m*

```
function Models=MPSoC_Id_Hinf(Deployment, Power, Tenvi, Temperature, Tsampling,
    ModelOrder, varargin)
% MPSoC_Id_Hinf returns a structure containing the single-core models
%
% Inputs parameters:
%
%   - Deployment : a map of the core
%   - Power : a matrix with all the power data of the cores
%   - Tenvi : the environment temperature data
%   - Temperature : the temperature data of the cores
%   - Tsampling : the sampling time
%   - ModelOrder : the desired order of the model
%   - varargin :
%       -> parameter 1: string containing the destination path where to
%       save the models
%
% Example:
% Models= MPSoC_Id_Hinf(Deployment, Power, Tenv, Temperatureerature, Tsampling,
    ModelOrder, 'path')
%

```

```
%% 0
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%        CHECK PARAMETERS %%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Save management
if isempty(varargin)
    save_model=0;
else
    if isnumeric(varargin{1})
        error('Libreria:MPSoC_Id_Hinf:ADDRESS','The address must be a string')
    else
        save_model=1;
        cd_old=cd;
        cd(varargin{1})
        mkdir('model_data');
        address=strcat(varargin{1},'\model_data\');
        cd(cd_old);
    end
end


% Number of cores
n_core=0;
for i=1:size(Deployment,1)
    for j=1:size(Deployment,2)
        if Deployment(i,j)>n_core
            n_core=Deployment(i,j);
        end
    end
end



%% 0
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%         IDENTIFICATION %%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fin=size(Temperature,1);
Deployment_ext=[zeros(size(Deployment,1)+2,1) [zeros(1,size(Deployment,2)); Deployment; zeros(1,size(Deployment,2))] zeros(size(Deployment,1)+2,1)];

% Iterative procedure repeated for each core
for j=1:n_core

```

```
65      disp('Identification of the model');disp(j);
66
67      % Optimization variables definition
68      a=sdpvar(ModelOrder,1) ;
69      bpow=sdpvar(ModelOrder,1);
70      bTenv=sdpvar(ModelOrder,1);
71      b=sdpvar(ModelOrder,n_core);
72      x=sdpvar(1,1);
73
74      % Understanding neighbors
75      B=zeros(2,n_core+2);
76      B(:,1:2)=ones(2,2);%[bpow bTenv];
77      for rows=2:size(Deployment_ext,1)-1
78          for colums=2:size(Deployment_ext,2)-1
79              if Deployment_ext(rows-1,colums)~=0 && Deployment_ext(rows-1,colums)==j
80                  B(:,Deployment_ext(rows,colums)+2)=ones(ModelOrder,1);
81              end
82              if Deployment_ext(rows+1,colums)~=0 && Deployment_ext(rows+1,colums)==j
83                  B(:,Deployment_ext(rows,colums)+2)=ones(ModelOrder,1);
84              end
85              if Deployment_ext(rows,colums-1)~=0 && Deployment_ext(rows,colums-1)==j
86                  B(:,Deployment_ext(rows,colums)+2)=ones(ModelOrder,1);
87              end
88              if Deployment_ext(rows,colums+1)~=0 && Deployment_ext(rows,colums+1)==j
89                  B(:,Deployment_ext(rows,colums)+2)=ones(ModelOrder,1);
90              end
91          end
92      end
93
94      % Constraints definition
95      Constraints = [Temperature(3:fin,j)-
         a(1)*Temperature(2:fin-1,j)-a(2)*Temperature(1:fin-2,j)- ((B(1,:).*[bpow(1,1)
         bTenv(1,1) b(1,:)])* [Power(2:fin-1,j) Tenvi(2:fin-1,1)
         Temperature(2:fin-1,1:n_core)]')'- ((B(2,:).*[bpow(2,1) bTenv(2,1) b(2,:)])*
         [Power(1:fin-2,j) Tenvi(1:fin-2,1) Temperature(1:fin-2,1:n_core)]')'>=-x,
96          Temperature(3:fin,j)- a(1)*Temperature(2:fin-1,j)-
         a(2)*Temperature(1:fin-2,j)- ((B(1,:).*[bpow(1,1) bTenv(1,1)
         b(1,:)])*[Power(2:fin-1,j) Tenvi(2:fin-1,1) Temperature(2:fin-1,1:n_core)]')'-
         ((B(2,:).*[bpow(2,1) bTenv(2,1) b(2,:)])* [Power(1:fin-2,j) Tenvi(1:fin-2,1)
         Temperature(1:fin-2,1:n_core)]')'<=0];
97
98      % Cost function definition
99      Objective = x;
100
101     % Solving optimization problem
102     options = sdpsettings('verbose',1,'solver','');
103     sol = solvesdp(Constraints,Objective,options);
104
105     % The j-th single-core model
106     eval(['Models.m' num2str(j) '.a=[double(a(1)) 1;double(a(2)) 0];']);
107     eval(['Models.m' num2str(j) '.b=[(B(1,:).*[double(bpow(1,1)) double(bTenv(1,1))
        double(b(1,:))]);(B(2,:).*[double(bpow(2,1)) double(bTenv(2,1))
```

```
        double(b(2,:))])];']);
108     eval(['Models.m' num2str(j) '.b(isnan(Models.m' num2str(j) '.b))=0;']);
109     eval(['Models.m' num2str(j) '.c=[1 0];']);
110     eval(['Models.m' num2str(j) '.d=zeros(1,10);']);
111     eval(['Models.m' num2str(j) '.Ts=Tsampling;']);
112     eval(['Models.m' num2str(j) '.x0=[Tenvi(1,1); (Models.m' num2str(j)
        '.a(2,1)*Tenvi(1,1)+Models.m' num2str(j)
        '.b(2,2:n_core+2)*(Tenvi(1,1)*ones(n_core+1,1)))/(1-Models.m' num2str(j)
        '.a(2,2))];']);
113
114     % The j-th single-core model is saved to the address indicated
115     if (save_model==1)
116         cd(address)
117         eval(['Mod' num2str(j) '=Models.m' num2str(j) ';']);
118         eval(['save(strcat(address,''Mod' num2str(j) '''),''Mod' num2str(j) ''');']);
119         cd(cd_old);
120     end
121
122     disp('Done')
123 end
```

This function takes as inputs the deployment of the cores (*Deployment*), the data achieved by simulating the accurate thermal model (the power consumptions, *Power*, the temperature of all cores, *Temperature*, and the ambient temperature *Tenvi*), the sampling time used for collecting the data (*Tsampling*), the desired model order for the final models (*ModelOrder*), and an optional input containing the address where to save the models (called *address* in the rest of the code). The output of the function consists of a record whose fields are described below.

$$models.\begin{cases} m1.\begin{cases} a \\ b \\ c \\ d \\ Ts \\ x0 \end{cases} \\ \vdots \\ mN \end{cases}$$

where $m1,\ldots,mN$ are the models, a, b, c, d are the model matrices, Ts is the sampling time, and $x0$ is the initial state of the model.

The first part of the code checks the input parameters to verify the consistency of the address path entered by the user.

In the second part of the code, for each core i, the following instructions are executed through

the use of a loop:

1. the identification variables are instanced;
2. a double loop is used to detect the neighbors cores of the *i*-th core and for building the matrix B that has 1 coefficient in correspondence of the neighbor cores input, and 0 otherwise;
3. the constraints of the optimization problem are defined (*Constraints*);
4. the cost function of the optimization problem is defined (*Object*);
5. the problem is solved with opportunely defined options (*options*);
6. the matrices of the *i*-th model are build according to the results of the optimization problem;
7. the model is saved and assigned to the output record.

It is worth to note that for defining the problem we used the Yalmip toolbox (3).

C.2.3 POD approach

The code below is used in the main program to call the *POD_redu.m* function for realizing the model reduction of the accurate thermal system using the POD approach.

```
%% 3
        %VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
        %|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
        %%%%%%%%%%%%%%%%%%%%%%%                             %%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%         POD APPROACH        %%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%                             %%%%%%%%%%%%%%%%%%%%%%%
        %|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
        %AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA


% Model Reduction
POD=POD_redu(x_plant,TModel,0.99,3);

% Model discretization
POD_discrete_Model=c2d(POD.Model,Time.Step,'zoh');
```

The code simply calls the function *POD_redu.m* that takes as inputs all the data concerning the states of the accurate thermal model and other parameters, and returns as output a record structure with two fields:

Model: contains the reduced model;

Order: contains the final model order.

Then, the model is dicretized.

C.2.3.1 *POD_redu.m*

```
function POD=POD_redu(Data,Model,Eig_importance,Order_bias)
% POD_redu computes the reducted model using the POD approach to find
% the opttimal basis and the Galerkin projection to project the system
% in the new space.
%
% Inputs:
%    - Data: the data of the experiment (all the states);
%    - Model: the original system model (SS object);
%    - Eig_importance: the value of
%            sum_1^order(eigs(Correlation))/sum_1^(all)(eig(Correlation));
%    - Order_bias: value added to the order that is automatically found;
%
% Example: POD=POD_redu(Data,Model,0.99,3)
%

%% 0
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%         CHECK PARAMETERS
    %%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if ~isnumeric(Data)
    error('Libreria:POD_redu:ADDRESS','Data must be a numeric matrix');
end
if ~isobject(Model)
    error('Libreria:POD_redu:ADDRESS','Model must be a Matlab SS object');
end
if ~isnumeric(Eig_importance)
    error('Libreria:POD_redu:ADDRESS','The Eig_importance variable must be a
    number');
end
if ~isnumeric(Order_bias)
    error('Libreria:POD_redu:ADDRESS','The Order_bias variable must be a number');
end


%% 1
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
    %%%%%%%%%%%%%%%%%%%%%%%%         MODEL REDUCTION
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% snapshot matrix creation using the measurements x_plant
T_snap= Data';

% mean subtraction
mean_value = mean(T_snap,2);
T_snap = T_snap - repmat(mean_value,1,size(T_snap,2));

% Find correlation matrix Corr
Corr=1/size(T_snap,2)*T_snap*T_snap';

% Solve eigenvalue problem
[eigenvectors,eigenvalues]=eig(Corr);
eigenvalues=diag(eigenvalues);

% Find the basis order M
M=0;
numerator=0;
denominator=sum(eigenvalues);
P_M=0;
while (P_M<Eig_importance)
    M=M+1;
    numerator=numerator+eigenvalues(M);
    P_M=numerator/denominator;
end
POD_coeff=eigenvectors'*T_snap;

% Eigenvalue spectrum
plot(1:size(eigenvalues,1),eigenvalues,'*b');

% Reduced basis \Phi_M
n_modes=M+Order_bias;
POD.Order=n_modes;
basis=eigenvectors(:,1:n_modes);

% Reduced model (Galerkin)
Ar=basis'*Model.a*basis;
Br=basis'*Model.b;
Cr=Model.c*basis;
Dr=Model.d;
POD.Model=ss(Ar,Br,Cr,Dr,Model.Ts);
```

This function takes as inputs the data achieved by simulating the accurate thermal model *Data* (i.e. a matrix with the value of all the model states on columns for each sampling interval on rows), the accurate model *Model*, that must be a SS Matlab object, and two other parameters

for specifying the accuracy of the resulting model: the *Eig_importance*, that is a number in the interval $[0,1]$ and allows the function to automatically determine the model order, and the *Order_bias*, that allows the user to modify the order automatically obtained. The result is a structure containing the reduced model and its final order.
In the first part of the code the input parameters are checked.
In the second part, the algorithm shown in Section 4.1.3 is implemented. First, the correlation matrix of the data is found, then the basis order is automatically achieved (the variable M) according to the *Eig_importance* input parameter. Then the order is modified according to the *Order_bias* input parameter, and finally the reduced order model is obtained by exploiting the Galerkin projection mechanism on the original model *Model*.

C.3 The distributed MPC control solution

In this Section we reported the code of the distributed MPC thermal controller implemented using different toolboxes.

C.3.1 Hybrid Toolbox

The Hybrid Toolbox (2) is a MATLAB/Simulink toolbox that allows the user to design a constrained optimal controller for hybrid dynamical systems with either implicit or explicit form. It also provides a Simulink library, multiparametric solvers for QP and LP problems, visualization functions for polyhedral objects and a C-code generator for embedded applications.

We used this toolbox because it is more flexible of the MPC toolbox in the problem definition and it allows us to manage explicit formulations of the controller. In the explicit solution the state-space is divided in polyhedral partitions each one associated with a linear control law. The number of these partitions is a good metric for measuring the complexity of the controller.

C.3.1.1 Textual version

In the piece of code reported below, we present a possible way for implementing the distributed MPC thermal solution by using the hybrid toolbox. The code simulates the thermal behavior of a chip controlled by using the aforementioned distributed MPC solution. It has been entirely realized using textual instructions.

```
%% 4
            %VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
```

C. ACCURATE MODEL

```
        %||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
        %AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

%% 4.1 Hybrid Toolbox

%% 4.1.2 Hybrid Toolbox for textual simulation

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Initialization
Vdd=1.35;          % Vdd assumed constant for P_static (in P_dyn V_dd=h(f))
Temp_plant=zeros(Time.Points,N_CORE);
power_cores_cntrl_ideal=zeros(Time.Points,N_CORE);
frequency_cores_cntrl=zeros(Time.Points,N_CORE);
x_plant=zeros(Time.Points-1,size(TModel.a,1));
x_obsv=zeros(Time.Points-1,size(models.m1.a,1));

% Controller parameters
rho=10^5;      % weight of the slack variable (epsilon) for soft constraints
Vy_max=0;      % 0=hard 1=soft constraints
T_CRIT=330;     % Critical temperature
R_u=1;          % weight of each power error P_T-P_C


% For each core "i":
%
% min P_C_i' * Q_qp_i * P_C_i + f_qp_i' * P_C_i
% s.t.
% A_qp_i * P_C_i <= b_qp_i
%
% where
% f_qp_i = [- P_T_i * f_1 , 0]
% b_qp_i = T_CRIT + b_1_i * x_obsv + b_2_i * [Tenvironment, Tneigh]

ModelsName='models.m';
for i=1:N_CORE
    j=0;
    eval(['Q_qp_' num2str(i) '= [R_u zeros(size(R_u,1),1); zeros(1,size(R_u,1))
    rho].*2;']);
    eval(['f_1_' num2str(i) '=2*R_u;']);
    support=strcat(ModelsName,num2str(i));
    eval(['aa_' num2str(i) '=' ModelsName  num2str(i) '.a;']);
    eval(['bb_' num2str(i) '=' ModelsName  num2str(i) '.b(:,1:N_CORE+2);']);
    eval(['cc_' num2str(i) '=' ModelsName  num2str(i) '.c;']);
    eval(['dd_' num2str(i) '=' ModelsName  num2str(i) '.d(:,1:N_CORE+2);']);
    eval(['A_qp_' num2str(i) '=[cc_' num2str(i) '*bb_' num2str(i) '(:,1)
    -Vy_max];']);
    eval(['b_1_' num2str(i) '=-cc_' num2str(i) '*aa_' num2str(i) ';']);
```

```
53      eval(['b_2_' num2str(i) '=-cc_' num2str(i) '*bb_' num2str(i) '(:,2:end);']);
54      % Initial state of each observer
55      eval(['x_obsv_' num2str(i) '=' ModelsName  num2str(i) '.x0'';']);
56  end
57
58  % Plant
59  TModel_discrete=c2d(TModel,Time.Step,'zoh');
60  x_plant(1,:)=X0';
61  Temp_plant(1,:)=(TModel_discrete.C*X0)';
62
63  % Target trace (Fluidanimate)
64  frequency_cores_target = OFluidanimate.Freq(1:Time.Points, N_CORE);
65  CPI_cores_target = OFluidanimate.CPI(1:Time.Points ,N_CORE);
66  power_cores_target = F_CPI_2_P(frequency_cores_target, CPI_cores_target, IDLE,
        Tenv(1,1), Vdd);
67
68
69      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%   SIMULATION   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
70
71  % Simulation loop
72  for i=1:Time.Points-1
73
74      % Plant temperature and state
75      x_plant(i+1,:)= (TModel_discrete.a* x_plant(i,:)'+ TModel_discrete.b*
        [power_cores_cntrl_ideal(i,:) Tenv(i,1)]')';
76      Temp_plant(i+1,:)= TModel_discrete.c* x_plant(i+1,:)';
77
78      % Observer estimation
79      Temp_neighs(1,:)= zeros(1,N_CORE);
80      for j=1:N_CORE
81          eval(['x_obsv_' num2str(j) '(i+1,:)=aa_' num2str(j) '*x_obsv_' num2str(j)
        '(i,:)''+ bb_' num2str(j) '* [power_cores_cntrl_ideal(i,j) Tenv(i,1)
        Temp_plant(i,:)]''+ ' ModelsName num2str(j) '.k_obsv*(Temp_plant(i,j)''- cc_'
        num2str(j) '*x_obsv_' num2str(j) '(i,:)'');']);
82          eval(['Temp_neighs(1,j) = x_obsv_' num2str(j) '(i+1,1); ']);
83      end
84
85      % Solution of the QP and updating of QP matrices
86      for j=1:N_CORE
87          eval(['f_qp_' num2str(j) '=[(-power_cores_target(i+1,j)*f_1_' num2str(j)
        ')'';0];']);
88          eval(['b_qp_' num2str(j) '=T_CRIT+b_1_' num2str(j) '*x_obsv_' num2str(j)
        '(i+1,:)''+b_2_' num2str(j) '*[Tenv(i+1,1) Temp_neighs(1,:)]'';']);
89          eval(['s=qpsol(Q_qp_' num2str(j) ',f_qp_' num2str(j) ',A_qp_' num2str(j) ',
        b_qp_' num2str(j) ', [], [], [' ModelsName  num2str(i) '.x0(j) ' ModelsName
        num2str(i) '.x0(j+N_CORE)], 4, inv(Q_qp_' num2str(j) '));']);
90          power_cores_cntrl_ideal(i+1,j) = s(1)';
91      end
92
93      % Power to frequency conversion (ideal)
94      frequency_cores_cntrl(i+1,:)= P_CPI_2_F(power_cores_cntrl_ideal(i+1,:),
        CPI_cores_target(i+1,:), IDLE, Tenv(1,1),Vdd);
```

```
95
96      % Frequency to power conversion
97      power_cores_cntrl_ideal(i+1,:)= F_CPI_2_P(frequency_cores_cntrl(i+1,:),
        CPI_cores_target(i+1,:), IDLE, Tenv(1,1),Vdd);
98
99  end
100
101
102     %%%%%%%%%%%%%%%%%%%%%%%%%%%%    VISUALIZATION    %%%%%%%%%%%%%%%%%%%%%%%%%%%%
103
104 y_NoContr=dlsim(TModel_discrete.a, TModel_discrete.b, TModel_discrete.c,
        TModel_discrete.d, [power_cores_target(1:Time.Points,:) Tenv(1:Time.Points,:)],
        X0);
105
106 core_num=1;
107 ax(1) =subplot(611);plot(1:1:Time.Points-1, [Temp_plant(1:Time.Points-1,core_num),
        y_NoContr(1:Time.Points-1,core_num)], 'LineWidth',2); title('Temperature');
        legend('MPC','no MPC','Location','Best');
108 ax(2) =subplot(612);plot(1:1:Time.Points-1,
        frequency_cores_cntrl(1:Time.Points-1,core_num), 'LineWidth',2); title('Provided
        Frequency')
109 ax(3) =subplot(613);plot(1:1:Time.Points-1,
        power_cores_cntrl_ideal(1:Time.Points-1,core_num), 'LineWidth',2);
        title('Provided power');%
110 ax(4) =subplot(614);plot(1:1:Time.Points-1,
        frequency_cores_target(1:Time.Points-1,core_num), 'LineWidth',2); title('Target
        Frequency')
111 ax(5) =subplot(615);plot(1:1:Time.Points-1,
        power_cores_target(1:Time.Points-1,core_num), 'LineWidth',2); title('Target
        power');
112 ax(6) =subplot(616);plot(1:1:Time.Points-1,
        CPI_cores_target(1:Time.Points-1,core_num), 'LineWidth',2); title('Target CPI');
113 linkaxes(ax,'x');
```

In the first part we initialized the variables used to storage the temperature and the states of the plant (*Temp_plant* and *x_plant* respectively), the controlled frequency and power consumption (*frequency_cores_cntrl* and *power_cores_cntrl_ideal* respectively), and the observer state (*x_obsv*). Notice that the variable name of the controlled power consumption, that is the power provided to cores after being regulated, ends with the word "ideal". This because we assume that the controller perfectly know the future workload (CPI) applied to the cores in at the next sampling interval, even though it is usually unpredictable. In other solutions we will show how to avoid this assumption by assigning as future CPI value the one measured at the previous sampling interval hypothesizing a low variability of the workload between two sampling instants.

In lines 24-56 we have defined the controller parameters, those which could be define off-line.

Assume the optimization problem for each local MPC controller is given by,

$$\min_{w_i} \frac{1}{2} \cdot w_i^T(t) \cdot Q_{qp,i} \cdot w_i(t) + f_{qp,i}^T \cdot w_i(t) \tag{C.1a}$$

$$s.t.$$

$$A_{qp,i} \cdot w_i(t) \leq b_{qp,i} \tag{C.1b}$$

where $w_i = [P_{C,i}\ \varepsilon]$ is the manipulated variable. $P_{C,i}$ represents the power consumption of the core i, namely *power_cores_cntrl_ideal* in the code. ε instead is a slack variable used to manage the rigidity of the constraints: the greater is ε the softer is the constraint.
T_CRIT is the critical temperature threshold of the MPC local controller, the weights matrix $Q_{qp,i}$ in (C.1a) contains *R_u*, the weight for the power consumption, and *rho*, the weight for the slack variable. The rest of the parameters of each local controller are defined inside the loop at lines $41-56$. *Q_qp_i*, *A_qp_i* are $Q_{qp,i}$ and $A_{qp,i}$ respectively, *aa_i*, *bb_i*, *cc_i*, and *dd_i* are the matrices names of the reduced order single-core thermal model of the i-th core (used in place of the structure variables containing the model created during the identification process for the sake of simplifying notation), and *x_obsv_i* is the state of the observe which is initialized. The arrays $f_{qp,i}$ and $b_{qp,i}$ cannot be defined at this stage because are time-varying. We may only defined some parameters (*f_1*, *b_1*, and *b_2*) that will be used to speed-up the computation of $f_{qp,i}$ and $b_{qp,i}$.
The accurate model of the system (i.e. the plant) is discretized with the same sampling time of the controller. It is worth to note that this is a strong simplification because we do not consider the thermal variations between the sampling interval. However, we realized Simulink simulations where the plant is continuous-time. We also realized textual simulations where the sampling time of the controller is far slower than the one used for updating the system, but we intentionally decide to not complicate this example with extra code lines.
Finally the frequency and the CPI of the trace to be simulated is defined. The target power of the core can be obtained by exploiting the Power Model defined in Appendix B. Notice that the Power Model accounts the V_{dd} (set as constant at the beginning of the code) by including in the dynamic power equation the $V_{dd} = h(frequency)$ function. V_{dd} is simply used to compute the static power.

The second part of the code realized the simulation. It consists of a loop, where each iteration represents a sampling interval. The operations executed during the loop are:

1. updating of plant states and outputs;
2. estimation of the unknown state of each single-core thermal model by using the local observers;
3. updating of the matrices *f_qp_i* and *b_qp_i*;
4. solution of the QP prolem for each local controller;
5. power-to-frequency conversion;
6. frequency-to-power conversion (not executed by the real control algorithm);

The instructions in the third part realized a visual comparison between the controlled and uncontrolled solutions.

C.3.1.2 Simulink version

The same simulations can be obtained using a Simulink block diagram. The scheme is shown in Fig. C.3 where a 48-core processor is controlled by using 48 local controllers.

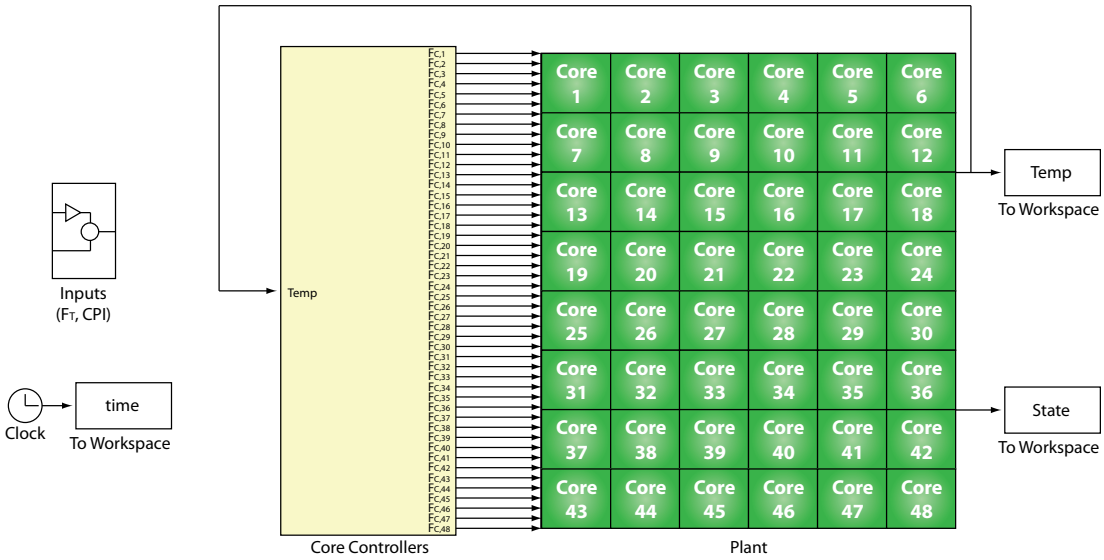


Figure C.3: Simulink control scheme using the Hybrid Toolbox

In the figure we can notice three main blocks. The first on the left provides the input benchmarks (frequency and CPI) for all cores, the central one contains the controllers, while the

right one contains the continuous time accurate model. Focusing on the block of the controllers, Fig. C.4 and Fig. C.5 show the details of the implementation.

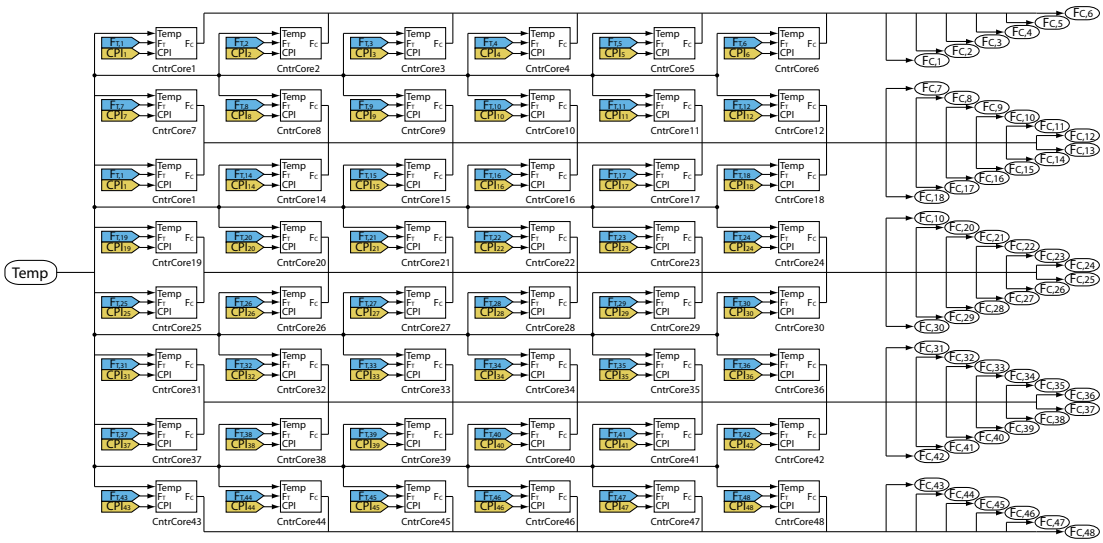


Figure C.4: The 48 core controllers

In Fig. C.4 the single local controllers are shown. As expected the *i*-th controller takes as inputs the frequency and the CPI of the core *i* and the temperature of all cores. Notice that to simplify the implementation we gave as inputs all cores temperatures, however, as already mentioned, the coefficients of the input matrix *B* of the single-core model used for predictions are equal to zero in correspondence of unusable temperatures (i.e. only the temperatures of the *i*-th core and other ones of the neighbors are admitted).

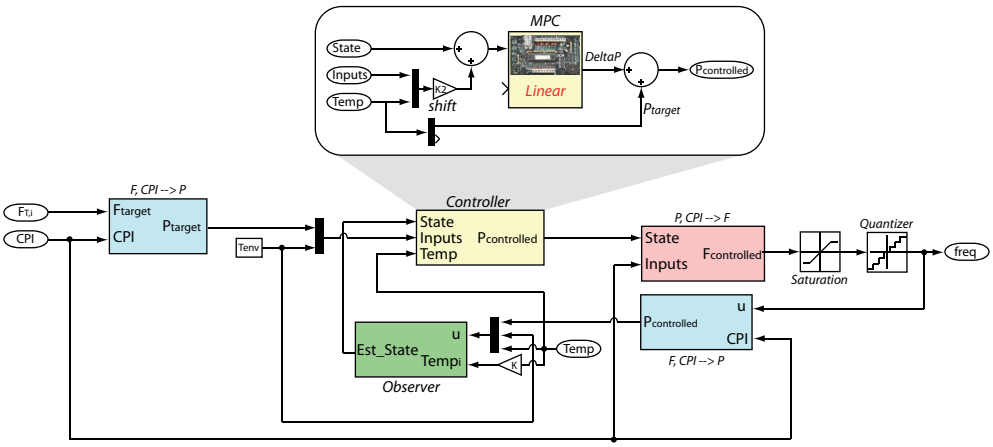


Figure C.5: The single local controller

Fig. C.5 instead shows the single local controller. The target frequency and CPI are translated in a power consumption requirement. The obtained power and the ambient temperature enter as inputs in the controller block that returns as output the controlled power consumption which maintains the temperature under the critical value. The controller power is subsequently converted into the controlled frequency that, after being quantized (we introduce quantization as a disturbance element), is used to feed the plant. Notice that we also impose a saturation to limit the power between the maximum and minimum values that are the power dissipated by the core when it executes respectively at maximum and at minimum speed. The frequency really assigned to the plant (the quantized one) is then used, coupled with the temperature information, as input of the observer block which estimates the unknown second state (indeed we assume, as we did during all this thesis, two states per single-core model). In Fig. C.5 it is also possible to see how the controller is implemented. The *MPC* block belongs to the Hybrid Simulink library. The block can manage two types of problems: the regulation problem and the tracking problem. The equation of the two problems are shown below.

$$\min x'(t+N|t)Px(t+N|t)+\sum_{k=0}^{N-1} x'(t+k|t)Qx(t+k|t)+u'(t+k)Ru(t+k)+\rho\varepsilon^2 \tag{C.2a}$$

$s.t.$

$$y_{min}-\varepsilon \leq y(t+k|t) \leq y_{max}+\varepsilon\,,\ k=1,\ldots,N_y \tag{C.2b}$$

$$u_{min} \leq u(t+k) \leq u_{max}\,,\ k=0,\ldots,N_{cu} \tag{C.2c}$$

$$u(t+k)=Kx(t+k|t)\,,\ k\geq N_c \tag{C.2d}$$

$$x(t+k+1|t)=Ax(t+k|t)+Bu(t+k) \tag{C.2e}$$

$$y(t+k|t)=Cx(t+k|t)+Du(t+k) \tag{C.2f}$$

$$\min \sum_{k=0}^{h_p-1} (y'(t+k|t)-r(t))S(y(t+k|t)-r(t))+\Delta u'(t+k)T\Delta u(t+k)+\rho\varepsilon^2 \tag{C.3a}$$

$s.t.$

$$y_{min}-\varepsilon \leq y(t+k|t) \leq y_{max}+\varepsilon\,,\ k=1,\ldots,N_y \tag{C.3b}$$

$$u_{min} \leq u(t+k) \leq u_{max}\,,\ k=0,\ldots,N_{cu} \tag{C.3c}$$

$$\Delta u_{min} \leq \Delta u(t+k) \leq \Delta u_{max}\,,\ k=0,\ldots,N_{cu} \tag{C.3d}$$

$$u(t+k)=Kx(t+k|t)\,,\ k\geq N_c \tag{C.3e}$$

$$x(t+k+1|t)=Ax(t+k|t)+Bu(t+k) \tag{C.3f}$$

$$y(t+k|t)=Cx(t+k|t)+Du(t+k) \tag{C.3g}$$

As it is possible to see the problem are slightly different respect the one we solved because it is not possible to track the inputs (the power dissipation in our case). Our idea has been to use the regulation problem where the input is the power consumption error *DeltaP* (i.e. the difference between the target power consumption P_{target} and the controlled power consumption $P_{controlled}$). However, this solution is not correct because not all the model inputs have to be computed by the optimization problem. The input signals that enter in the model can be classified in two families, the manipulated inputs, which are the inputs that are calculated by the controller, and the measured disturbances, namely the inputs that cannot be modified by the controller (e.g. the ambient temperature and the temperature of the neighbors). Making some tests we noticed that setting the weight of the measured disturbances to zero does not prevent their modification. Our idea has been to exploit the superposition principle of the linear model, updating the state with the measured disturbances contributions and giving this state (we called it *shifted state*) as input to the regulation optimization problem. The controller returns *DeltaP* that has to be subtracted to the P_{target} to obtain $P_{controlled}$. The controller parameters inside the *MPC* block can be obtained calling the function *distrMPC_Hybrid.m* with the code below.

```
%% 4.1.2 Hybrid Toolbox for Simulink simulation

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%    PARAMETERS    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

T_CRIT=360*ones(1,N_CORE);      % Critical temperature threshold
H_p=2;                          % prediction horizon
H_c=1;                          % control horizon

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%    CENTRALIZED    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```

Ctrl_Centr=centrMPC_Hybrid(model,T_CRIT,H_p,H_c,Time.Step);


    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%  DISTRIBUTED  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Ctrl_Distributed=distrMPC_Hybrid(models,T_CRIT,H_p,H_c,Time.Step);
```

distrMPC_Hybrid.m

```
function Controller=distrMPC_Hybrid(models,T_CRIT,N,Nc)
% distrMPC_Hybrid returns:
%   - Controller.c#.ctrl.impl = Implicit controller
%   - Controller.c#.ctrl.expl = Explicit controller
%   - Controller.c#.shift = the gain matrix for shifting the state (in
%     Simulink scheme)
%   - Controller.c#.Kobsvdiscr = the gain matrix of the observer
%   - Controller.c#.model = the model of the single core system
%
% The input parameters are:
%   - models : a structure with the a, b, c, d matrix of each single-core model
%   - T_CRIT : an array with the temperature limits of all the cores
%   - N : the prediction horizon
%   - Nc : the control horizon


%% 0

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%        CONTROL PARAMETERS
    %%%%%%%%%%%%%%%%%%%%%%%%%%%

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

n_core=size(Modello.c,1);

% Control parameter definition for each core
for k=1:n_core


    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%     IMPLICIT CONTROLLER
    %%%%%%%%%%%%%%%%%%%%%%%%%%%

    % System model definition
    eval(['sys=ss(models.m' num2str(k) '.a, models.m' num2str(k) '.b(:,1), models.m'
    num2str(k) '.c, models.m' num2str(k) '.d(:,1), models.m' num2str(k) '.Ts);'])
    INname='Delta P';
```

```
    OUTname='T';
    STATEname='x';
    sys.InputName(1,1)={strcat(INname,num2str(k))};
    sys.OutputName(1,1)={strcat(OUTname,num2str(k))};
    sys.StateName(1,1)={strcat(OUTname,num2str(k))};
    ModelOrder=size(sys.a,1);
    for i=2:ModelOrder
        sys.StateName(1+(i-1),1)={strcat(STATEname,num2str(k),'_',num2str(i))};
    end
    % Limits
    clear limits
    limits.umin=-Inf;                    % Lower bounds on Input
    limits.umax=Inf;                     % Upper bounds on Input
    limits.ymin=0;                      % Lower bounds on Output
    limits.ymax=T_CRIT(k);               % Upper bounds on OUTput

    % Costs
    clear cost
    cost.Q=diag(zeros(1,ModelOrder));           % State weight
    cost.R=0.0001;                              % Input weight
    cost.P=diag(zeros(1,ModelOrder));           % Final State weight
    cost.rho=Inf;                               % Hard constraints

    % Intervals
    clear interval
    interval.Nu=Nc;              % input horizon  u(0),...,u(Nu-1)
    interval.N=N;                % output horizon   \sum_{k=0}^{Ny-1}
    interval.Ncy=1;              % output constraints horizon   k=0,...,Ncy
    interval.Ncu=1;              % input constraints horizon    k=0,...,Ncu

    % Controller
    eval(strcat('Controller.c',num2str(k),'.ctrl.impl = lincon(sys,''reg'', cost,
     interval, limits);'));



    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%      EXPLICIT CONTROLLER
    %%%%%%%%%%%%%%%%%%%%%%%%%%

    % Range & Options
    clear range options
    range.xmin=[0 0];
    range.xmax= [400 400];
    range.umin=-50;
    range.umax=5;

    % Options
    options.reltol=1e-7;
    options.join=1;
    options.verbose=1;
    options.uniteeps=1e-3;
```

```
    options.qpsolver='qpact';   % Use active-set QP

    % Explicit Controller
    eval(strcat('Controller.c',num2str(k),'.ctrl.expl = expcon(Controller.c',num2str(k),'.ctrl.impl, range, options);'));



    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%       SHIFT %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % matrix for computing the contribution of measured disturbances
    eval(strcat('Controller.c',num2str(k),'.shift = (models.m', num2str(k),'.a^-1)*models.m', num2str(k), '.b;'));



    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%       OBSERVER %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % Observer
    eigenvalues = eig(model.a)*0.4;
    eval(strcat('Controller.c',num2str(k),'.Kobsvdiscr = (place(models.m',num2str(k),'.a'', models.m',num2str(k),'.c'', eigenvalues))'';'));
    eval(strcat('Controller.c',num2str(k),'.model = models.m',num2str(k),';'));
end
```

The function takes as inputs a structure containing the identified single-core models (*models*), the prediction (*N*) and the control horizon (*Nc*), respectively set to 2 and 1, and the critical temperature threshold *T_CRIT*. The function returns as output the structure:

$$controllers.\begin{cases} c1.\begin{cases} ctrl\begin{cases} impl \\ expl \end{cases} \\ Kobsvdiscr \\ model \\ shift \end{cases} \\ \vdots \\ cN \end{cases}$$

where *ci* contains the parameters useful for the *i*-th core simulation, *ctrl* contains the controller parameters (*impl* the parameters of the implicit controller, and *expl* those of the explicit controller), *Kobsvdiscr* is the gain matrix of the Luenberger observer, *model* contains the matrices a, b, c, d of the model, and *shift* is the gain matrix that multiplied by the measured disturbances

and added to the current state returns the shifted state.
Analyzing the code, the instructions to be executed to each core are contained inside a loop. First a SS Matlab object containing the model is obtained. Subsequently the parameters of the controller are set and the function *lincon*, belonging to the Hybrid Toolbox library, is called to generate the implicit controller data structure. The same steps are executed to obtain the explicit controller data structure, but this time the function called is *expcon*. Finally the observer gain matrix and the shift matrix are computed.

C.3.2 Yalmip Toolbox

The distributed MPC thermal solution has been implemented also using the Yalmip toolbox (3), a language for modeling and solving convex and non-convex optimization problems. It is a free toolbox for MATLAB that allows the user to describe the problem at high level without caring about how the problem will be solved.

C.3.2.1 Textual version

As for the Hybrid Toolbox, the piece of code reported below simulates the thermal behavior of a chip controlled by the distributed MPC thermal solution. The code is entirely realized using textual instructions.

```
%% 4.2 Yalmip Toolbox

%% 4.2.1 Yalmip Toolbox for textual simulation

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%    PARAMETERS    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Trace name (structure with fields)
trace1 = 'Fluidanimate';
trace2 = 'Facesim';
trace3 = 'Bodytrack';
trace4 = 'Dedup';
trace5 = 'Raytracing';
n_traces = 5;       % number of traces to be simulated
Vdd = 1.35;         % Vdd assumed constant for P_static (in P_dyn V_dd=h(f))
IDLE = 1;
P_MIN = 0.254043439071652;  % power dissipated when cores run the min freq

% Controller parameters
R_u = eye(1);       % weight of each power error P_T-P_C
rho = 10^5;         % weight of the slack variable (epsilon) for soft constraints
Vy_max = zeros(1,1);            % 0=hard 1=soft constraints
MPC_Thresh = 360*ones(N_CORE);  % MPC temperature threshold
```

C. ACCURATE MODEL

```
Q_qp = [R_u zeros(size(R_u,1),1); zeros(1,size(R_u,1)) rho];  % weight matrix


    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%    SIMULATION    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% For each trace
for ii=1:n_traces

    % Initialization
    clear frequency_cores_target CPI_cores_target CPI_cores_target_delayed
     power_cores_target_delayed power_cores_cntrl power_tot_cntrl
     frequency_cores_cntrl
    time_trace =size( eval(strcat(eval(['trace' num2str(ii) '']),'.Freq')),1);
    power_cores_cntrl = zeros(time_trace,N_CORE);
    power_tot_cntrl = zeros(time_trace,N_COMP);
    frequency_cores_cntrl = zeros(time_trace,N_CORE);

    % Plant
    TModel_discrete = c2d(TModel,Time.Step,'zoh');
    x_plant = zeros(time_trace-1,size(A,1));
    x_plant(1,:) = X0';
   Temp_plant = zeros(time_trace,N_CORE);
    Temp_plant(1,:) = (TModel_discrete.C*X0)';

    % Target Benchmarks
    frequency_cores_target = eval(strcat(eval(['trace' num2str(ii) '']),
    '.Freq(1:time_trace,1:N_CORE)'));
    CPI_cores_target = eval(strcat(eval(['trace' num2str(ii) '']),
    '.CPI(1:time_trace,1:N_CORE)'));
    % Delayed CPI: to simulate unpredictability of workload the CPI used by
    % the MPC is equal to the CPI measured the previous sampling time
    CPI_cores_target_delayed = [CPI_cores_target(1,:); CPI_cores_target(1:end-1,:)];
    power_cores_target_delayed = F_CPI_2_P(frequency_cores_target,
    CPI_cores_target_delayed, IDLE, Tenv(1,1), Vdd);

      % Observer initialization
    for j=1:N_CORE
        eval(['x_obsv_' num2str(j) '=zeros(time_trace,size(models.m' num2str(j)
    '.a,1));']);
        eval(['x_obsv_' num2str(j) '(1,:)=models.m' num2str(j) '.x0'';']);
        eval(['K_obs_' num2str(j) '=(place(models.m' num2str(j) '.a'', models.m'
    num2str(j) '.c'', (eig(models.m' num2str(j) '.a).*0.4)))'';']);
    end

    % Simulation loop (for each sampling time)
    for j=1:time_trace

        % Plant output
        Temp_plant(j,:)=TModel_discrete.c*x_plant(j,:)';

        % Computation of the Local MPC solution (QP problem)
        for jj=1:N_CORE
```

```
68              clear pot
69              pot=sdpvar(2,1);
70
        Objective=pot'*Q_qp*pot+pot'*[(-power_cores_target_delayed(j,jj)*2*R_u)';0];
71              eval(['Constraints=[[models.m' num2str(jj) '.c*models.m' num2str(jj)
        '.b(:,1) -Vy_max]*pot<=MPC_Thresh-models.m' num2str(jj) '.c*models.m'
        num2str(jj) '.a*x_obsv_' num2str(jj) '(j,:)''-models.m' num2str(jj)
        '.c*models.m' num2str(jj) '.b(:,2:end)* [Tenv(j,1) Temp_plant(j,:)]'',
        pot(1,1)>=P_MIN];']);
72              Options=sdpsettings('verbose',0,'solver','');
73              sol = solvesdp(Constraints,Objective,Options);
74              power_cores_cntrl(j,jj)=double(pot(1,1));
75          end
76
77          % Frequency to power conversion (with delayed CPI)
78          frequency_cores_cntrl(j,:)= P_CPI_2_F(power_cores_cntrl(j,:),
        CPI_cores_target_delayed(j,:), IDLE, Tenv(1,1), Vdd);
79
80          % power to frequency conversion (without delayed CPI)
81          power_cores_cntrl(j,:)= F_CPI_2_P(frequency_cores_cntrl(j,:),
        CPI_cores_target(j,:), IDLE, Tenv(1,1), Vdd);
82
83          % Total power array (need to add caches power)
84          power_tot_cntrl(j,:)=[power_cores_cntrl(j,1),(power_cores_cntrl(j,1)+
        power_cores_cntrl(j,3))./4.*percentuale,
        (power_cores_cntrl(j,2)+power_cores_cntrl(j,4))./4.*percentuale,
        power_cores_cntrl(j,2:3),
        (power_cores_cntrl(j,1)+power_cores_cntrl(j,3))./4.*percentuale,
        (power_cores_cntrl(j,2)+power_cores_cntrl(j,4))./4.*percentuale,
        power_cores_cntrl(j,4), zeros(1,4), power_cores_cntrl(j,5),
        (power_cores_cntrl(j,5)+power_cores_cntrl(j,7))./4.*percentuale,
        (power_cores_cntrl(j,6)+power_cores_cntrl(j,8))./4.*percentuale,
        power_cores_cntrl(j,6:7),
        (power_cores_cntrl(j,5)+power_cores_cntrl(j,7))./4.*percentuale,
        (power_cores_cntrl(j,6)+power_cores_cntrl(j,8))./4.*percentuale,
        power_cores_cntrl(j,8)];
85
86
87          % Computation of the plant state
88          x_plant(j+1,:)= (TModel_discrete.a*x_plant(j,:)'+ TModel_discrete.b*
        [power_tot_cntrl(j,:) Tenv(j,1)]')';
89
90          % Observer
91          for jj=1:N_CORE
92              eval(['x_obsv_' num2str(jj) '(j+1,:)= models.m' num2str(jj) '.a*x_obsv_'
        num2str(jj) '(j,:)''+ models.m' num2str(jj) '.b*[power_cores_cntrl(j,jj)
        Tenv(j,1) Temp_plant(j,:)]''+ K_obs_' num2str(jj) '*(Temp_plant(j,jj)''-
        models.m' num2str(jj) '.c*x_obsv_' num2str(jj) '(j,:)'');']);
93          end
94      end
95      eval(['save(''data' num2str(ii) ''', ''Temp_plant'', ''frequency_cores_cntrl'',
        ''power_cores_cntrl'', ''frequency_cores_target'', ''power_cores_target'',
```

```
    ''CPI_cores_target'');']) 
end


    %%%%%%%%%%%%%%%%%%%%%%%%%%%   VISUALIZATION   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

y_NoContr=dlsim(TModel_discrete.a, TModel_discrete.b, TModel_discrete.c,
    TModel_discrete.d, [power_cores_target(1:Time.Points,:) Tenv(1:Time.Points,:)],
    X0);

core_num=1;
ax(1) =subplot(611);plot(1:1:Time.Points-1, [Temp_plant(1:Time.Points-1,core_num),
    y_NoContr(1:Time.Points-1,core_num)], 'LineWidth',2); title('Temperature');
    legend('MPC','no MPC','Location','Best');
ax(2) =subplot(612);plot(1:1:Time.Points-1,
    frequency_cores_cntrl(1:Time.Points-1,core_num), 'LineWidth',2); title('Provided
    Frequency')
ax(3) =subplot(613);plot(1:1:Time.Points-1,
    power_cores_cntrl(1:Time.Points-1,core_num), 'LineWidth',2); title('Provided
    power');%
ax(4) =subplot(614);plot(1:1:Time.Points-1,
    frequency_cores_target(1:Time.Points-1,core_num), 'LineWidth',2); title('Target
    Frequency')
ax(5) =subplot(615);plot(1:1:Time.Points-1,
    power_cores_target(1:Time.Points-1,core_num), 'LineWidth',2); title('Target
    power');
ax(6) =subplot(616);plot(1:1:Time.Points-1,
    CPI_cores_target(1:Time.Points-1,core_num), 'LineWidth',2); title('Target CPI');
linkaxes(ax,'x');
```

In the first part the parameters of the controllers are set. Differently from the textual version realized with the Hybrid Toolbox, the code uses a loop to simulate a set of benchmarks instead of a single one. Inside the benchmark loop the variables, the plant, and the observers are initialized, and the target frequency *frequency_cores_target* and the CPI *CPI_cores_target* of the *ii*-th benchmark are loaded. Before starting the simulation the target frequency and the CPI are used to compute the target power *power_cores_target_delayed*. It is worth to note that, differently from the code realized with the Hybrid Toolbox, we assumed the unpredictability of the CPI, therefore we delayed the CPI (*CPI_cores_target_delayed*) of one sampling time. Clearly this affects the controller efficiency since the predicted CPI provides a wrong estimation of the target power consumption.

These operations are realized off-line. The second loop realizes the simulation by updating the plant output, and solving the optimization problem at each sampling time. The optimization problem has been defined using the functions of the Yalmip library. *sdpvar* defines the name

and the dimension of the optimization variable (i.e. *pot*, a 2×1 array where the first element is the controlled power of the core and the second element represents the slack variable ε). In *Objective* and *Constraints* the cost function and the constraints are respectively defined. Finally, the problem is solved using the function *solvesdp*, according to the options defined in the *options* variable, and the solution is assigned to the power controlled variable *power_cores_cntrl*. This power is translated to frequency, the *frequency_cores_cntrl*, and assigned to the plant. At this point the plant simulator requires the computation of the power consumption from the controlled frequency and the CPI information in order to compute the temperature of the chip. However, it is worth to note that the CPI we used in this case is not the delayed, but the real one. Thus the controlled power consumption that feeds the plant is suboptimal.
When the simulation is completed the data are saved and then visualized.

C.3.2.2 Simulink version

Also the Yalmip-based implementation can be used in Simulink, however there are no Simulink libraries to use. The idea is to build a Matlab function that computes the solution of the optimization problem. Then using a standard *MATLAB function* block it is possible to call the function. The function is reported below.

distrMPC_Yalmip.m

```
function [power_cores_cntrl] = distrMPC_Yalmip(N_CORE, models, power_target_past,
    MPC_Thresh, Tenv, Temp, Temp_past, pow_past)

% Controller parameters initialization
Vy_max=zeros(1,1);
rho=10^5;
R_u=eye(1);
Q_qp=[R_u zeros(size(R_u,1),1); zeros(1,size(R_u,1)) rho];
pot=sdpvar(2,1);
power_cores_cntrl=zeros(8,1);

% Local MPC controller solution
for i=1:N_CORE
    % x_2 estimation (no need of observer)
    eval(['state=[Temp(i); models.m' num2str(i) '.a(2,1)* Temp_past(i)+ models.m'
    num2str(i) '.b(2,:)* [pow_past(i); Tenv; Temp_past]];']);

    % Objective function definition
    Objective=pot'*Q_qp*pot+ pot'* [(-power_target_past(i)*2*R_u)'; 0];

```

```
    % Constraints definition
    eval(['Constraints=[[models.m' num2str(i) '.c*models.m' num2str(i) '.b(:,1)
    -Vy_max]* pot<= MPC_Thresh- models.m' num2str(i) '.c* models.m' num2str(i)
    '.a*state- models.m' num2str(i) '.c* models.m' num2str(i) '.b(:,2:end)*
    [Tenv;Temp],pot(1,1)>= 0.254043439071652];']);

    % QP problem solution
    Options=sdpsettings('verbose', 0, 'solver','');
    sol = solvesdp(Constraints, Objective, Options);
    power_cores_cntrl(i)= double(pot(1,1));
end
```

The function take as input the number of the core *N_CORE*, the structure with the identified models *models*, the power consumption of the cores estimated with the delayed CPI measurements *power_target_past*, the critical temperature threshold *MPC_Thresh*, the ambient temperature *Tenv*, the current cores temperature *Temp*, and the temperature *Temp_past* and power *pow_past* at the previous sampling time. The function gives as output the controlled power consumption.
In the first part the parameters of the optimization problem are set. Subsequently a loop solve independently the local optimization problem. Focusing on the loop, the first instruction is to estimate the unknown state of the model (in this case we do not use the Luenberger observer). To complete this operation we need to storage the cores temperature and power consumption of the prior sampling interval. Then the objective function and the Constraints are defined and the problem solved using the specified options.

C.3.3 qpOASES

In this Section we have reported the distribute algorithm realized in C/C++ language. The code is briefly explained below.

```
// The program solves the problem below uteratively
//
// min (Pd-P)'*H*(Pd-P)
// s.t.
// T<T_CRIT
//
// that can be translated into a QP problem
//
// min x*Q*x + g'*x
// s.t.
// A*x < ubA
// 0 < x
//
```

```
14 // with x=[P_1 P_2...P_n eps]' where eps is the slack variable for constraints
       rigidity
15
16 #include <QProblem.hpp>
17 #include <sys/time.h>
18 #include <iostream>
19 #include <sstream>
20 #include <string>
21 #include <cstdlib>
22 #include "Distributed_Matrices.h.h"
23 #define Ka_DEFAULT  3.8696e-008 // frequency to power parameters
24 #define Kb_DEFAULT  2.4090
25 #define Kc_DEFAULT  1.1025
26 #define Kd_DEFAULT  0.0051
27 #define Ke_DEFAULT  -4.1376
28 #define Kf_DEFAULT   -0.3016
29 #define Z_DEFAULT  2.59E+02
30 #define K       1.38e-23
31 #define q       1.6e-19
32 #define alpha   1.5
33 #define KIDLE_STATIC 0.33
34 #define F_MIN 1600         // Minimum frequency
35 #define F_MAX 3000         // Maximum frequency
36 #define TOLLERANZA 1.e-6
37 #define MAX_ITERATION 1
38
39 #define NITER 5000         // Iteration number
40 #define HARD_SOFT 0.0      // 0/1 = Hard/Soft
41 #define WEIGHT1 2.0        // Weights of the Hessian matrix (x 2)
42 #define WEIGHT2 200000.0   // Weights of the Hessian matrix (slack variable)
43 #define TEMP_LIM 330    // Maximum temperature allowed
44 #define NUM_IT_QP_SOLVER 10   // Maximum iteration number (solver algorithm)
45 #define N_CORE 4        // Cores number
46 // Input Files
47 #define FIN_DEF_FREQ "inputFreqFluid.txt" // Frequency input
48 #define FIN_DEF_CPI "inputCPIFluid.txt"      // CPI input
49 // Output Files
50 #define FOUT_DEF_TEMP "outputCTempFreqDistrNF.txt"    // Temperatures
51 #define FOUT_DEF_POW "outputCPowFreqDistrNF.txt"      // Power controlled
52 #define FOUT_DEF_FREQ "outputCFreqDistrNF.txt"        // Frequency controlled
53 #define FOUT_DEF_TIME "outputCTempiFreqDistrNF.txt"      // Times
54
55
56 // I/O Function
57 void readVect(int dim,FILE* fin, double* vett);    // Reading of a vector from a file
58 void writeVect(double* y, int dim, FILE* fout);    // Writing of a vector in a file
59 void printVect(double* vett, int dim);          // Writing of a vector on the monitor
60 void printMat(double* mat, int righe, int columns);   // Writing of a matrix on the
       monitor
61
62
63 // Conversion function (frequency/power)
```

```
double Psta(float *p);
double f2p(double freq, float *p);
double f2p_inv(double freq, float *p);
double Amsterdam_Method( double (*f)(double,float*), double a, float *ap, double c,
    double tolerance, int max_iterations, int *err);


typedef struct {
   double H[2*2];       //={{WEIGHT1,0},{0,WEIGHT2}};
   double g[2];         //={0.0, 0.0};
   double A[1*2];       //={0.0,HARD_SOFT};
   double b_1[2];       //={0.0,0.0};
   double b_2[N_CORE+1];   //={0.0};
   double ubA[1];
   double lb[1];        //=0.0;
   double x_obsv[2];    //={0,0};
   double u_found_pow[N_CORE+2];
 } QP_Object;
```

The first inclusion is related to the qpOASES library (4) that we employed to solve the quadratic programming inside the MPC controller. qpOASES is an open-source C++ implementation of the recently on-line active set strategy proposed in (5), particularly suited for model predictive control (MPC) applications. Among the other classical inclusions, it is possible to notice the file *Distributed_Matrices.h* which contains the accurate thermal model matrices and the matrices of all the single-core models. The file is obtained using a Matlab script and the matrices are defined as follows:

Distributed_Matrices.h

```
// Plant
int dim1P_A=192;    //how many rows for the A matrix?
int dim2P_A=192;  //how many rows for the A matrix?
double P_A[192][192]={{0.520788,...}};
int dim1P_A=192;  //how many rows for the B matrix?
int dim2P_A=192;  //how many rows for the B matrix?
double P_A[192][192]={{0.520788,...}};
int dim1P_C=4;    //how many rows for the C matrix?
int dim2P_C=192;  //how many rows for the C matrix?
double P_C[4][192]={{0.000000,...}};
int dim1P_D=4;    //how many rows for the D matrix?
int dim2P_D=5;    //how many rows for the D matrix?
double P_D[4][5]={{0.000000,...}};
int dim1P_X0=192;
int dim2P_X0=1;
double P_X0[192][1]={{310.000000},{...}};
// Single-core Models
int dim1M_A=4;      // how many A matrices?
int dim2M_A=2;      // how many rows for each A matrix?
```

```
20 int dim3M_A=2;      // how many rocolumns for each A matrix?
21 double M_A[4][2][2]={{{0.000000, 1.000000},{-0.511909, 1.509581}},{{0.000000,
       1.000000},{-0.511610, 1.507963}},{{0.000000, 1.000000},{-0.510852,
       1.507153}},{{0.000000, 1.000000},{-0.511115, 1.508757}}};
22 int dim1M_B=4;      // how many B matrices?
23 int dim2M_B=2;      // how many rows for each B matrix?
24 int dim3M_B=6;      // how many rocolumns for each B matrix?
25 double M_B[4][2][6]={{{0.031718, 0.000414, 0.000000, -0.002307, 0.000000,
       0.000000},{0.017709, 0.001040, 0.000000, 0.000322, 0.000000,
       0.000000}},{{0.031760, 0.000448, -0.002290, 0.000000, -0.010885,
       0.000000},{0.017826, 0.001124, 0.000532, 0.000000, -0.004477,
       0.000000}},{{0.031744, 0.000488, 0.000000, -0.008217, 0.000000,
       -0.005814},{0.017783, 0.001223, 0.000000, -0.003106, 0.000000,
       -0.001285}},{{0.031734, 0.000404, 0.000000, 0.000000, -0.001108,
       0.000000},{0.017719, 0.001014, 0.000000, 0.000000, 0.000981, 0.000000}}};
26 int dim1M_C=4;      // how many C matrices?
27 int dim2M_C=1;      // how many rows for each C matrix?
28 int dim3M_C=2;      // how many rocolumns for each C matrix?
29 double M_C[4][1][2]={{{1.000000, 0.000000}},{{1.000000, 0.000000}},{{1.000000,
       0.000000}},{{1.000000, 0.000000}}};
30 int dim1M_D=4;      // how many D matrices?
31 int dim2M_D=1;      // how many rows for each D matrix?
32 int dim3M_D=6;      // how many rocolumns for each D matrix?
33 double M_D[4][1][6]={{{0.000000, 0.000000, 0.000000, 0.000000, 0.000000,
       0.000000}},{{0.000000, 0.000000, 0.000000, 0.000000, 0.000000,
       0.000000}},{{0.000000, 0.000000, 0.000000, 0.000000, 0.000000,
       0.000000}},{{0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000}}};
34 int dim1M_X0=4;     // how many initial state vectors?
35 int dim2M_X0=1;     // how many rows for each initial state vectors?
36 int dim3M_X0=2;     // how many rocolumns for each initial state vectors?
37 double M_X0[4][1][2]={{{310.000000, 310.586759}},{{310.000000,
       313.945288}},{{310.000000, 314.198166}},{{310.000000, 310.218260}}};
38 int dim1K_Obsv=4;   // how many observer gain matrix?
39 int dim2K_Obsv=2;   // how many elements
40 double K_Obsv[4][2]={{0.905748, 0.937297},{0.904778, 0.934619},{0.904292,
       0.933791},{0.905254, 0.936473}};
41 double x_plant[192];        // plant state dimension
42 double x_plant_old[192];
```

Subsequently, we defined the constants values: the parameters of the power model (for frequency to power conversions), the parameters of the optimization problem, and the name of the file where the input traces are read and the computed values are stored.

Then, the functions used inside the code are defined. Some functions simplify the reading and writing of the files and the writing of the data on the monitor, the others are used to convert power to frequency and frequency to power. In particular, the *Amsterdam_Method* function allows us to invert the nonlinear equation which relates frequency to power (it can be found as open source file on the web).

C. ACCURATE MODEL

Finally, the type of structure *QP_Object* is defined. It is a support structure that groups all the parameters necessary to define the local MPC controller (QP problem parameters, observer states, and controlled power consumption). For each core (in the code we used 4 cores) there will be an instance of the *QP_Object*.

```

int main(int argc,char **argv)
{

//--------------------------------------------------------------------
// Variabili
//--------------------------------------------------------------------
   using namespace qpOASES;

   // Time variables
   timeval t_start, t_stop, t_step;
   double time[NITER];

   // loop variables
   int i,ii,j,k, numPar, numVinc, nWSR;

   // QP object
   QP_Object QP[N_CORE];

   // Input arrays
   double inPow[N_CORE+1];
   double input[N_CORE+2];
   double inFreq[N_CORE+1];
   double inCPI[N_CORE];

   // Output arrays
   double Temp_plant[N_CORE];
   double Temp_plant_old[N_CORE];

   // Controlled power arrays (P_C)
   double u_found_pow[N_CORE+1];
   double u_found_freq[N_CORE+1];
   double u_neig_tot[N_CORE+1];

   // Observers
   double inno;
   double x_obsv_old[2];

   // Variables necessary for frequency conversions
   float param[5] = {1,1,310,1.35,0.5};  // param = [CPI, Idle, Temp, Vdd, Vt]
   float pinv[2] = {1,3};
   double P_static=0.0;
   double P_dyn=0.0;
   int err;

```

```
46    // File pointers inizialization
47    FILE* fin_Freq;
48    FILE* fin_CPI;
49    FILE* fout_Temp;
50    FILE* fout_Pow;
51    FILE* fout_Freq;
52    FILE* fout_Time;
53    fin_Freq=fopen(FIN_DEF_FREQ,"r");
54    fin_CPI=fopen(FIN_DEF_CPI,"r");
55    fout_Temp=fopen(FOUT_DEF_TEMP,"w");
56    fout_Pow=fopen(FOUT_DEF_POW,"w");
57    fout_Freq=fopen(FOUT_DEF_FREQ,"w");
58    fout_Time=fopen(FOUT_DEF_TIME,"w");
```

The first instructions in the main are devoted to instantiate the variables used in the code: the time variables (to store the time elapsed for finding the control decision), the counter variables, a vector containing the four instances of the *QP_Object*, the vectors that will contain the input trace data and the computed output data (plant temperature and controlled power), the support variables used to update the observers states and the observer states values, some parameters used for frequency to power conversion, and the pointer variables addressing the files where the data will be saved.

```
1
2   //---------------------------------------------------------------
3   // Initializations
4   //---------------------------------------------------------------
5
6      // Parameters number
7           numPar=dim2M_C+1;
8
9      // Constraints number
10          numVinc=dim2M_C;
11
12     // QP objects initializations
13     for(i=0;i<N_CORE;i++){
14        QP[i].H={WEIGHT1,0,0,WEIGHT2};
15        QP[i].g={0.0, 0.0};
16        QP[i].A={0.0,HARD_SOFT};
17        QP[i].b_1={0.0,0.0};
18        QP[i].b_2={0.0};
19        QP[i].lb={0.0};
20        QP[i].x_obsv={0.0,0.0};
21     }
22
23     // Definition of the constraints matrix A
24     for(k=0;k<N_CORE;k++){
25        for(i=0;i<2;i++){
26           QP[k].A[0]+=M_C[k][0][i]*M_B[k][i][0];
```

```
27          }
28       }
29
30
31       // Definition of b_1 and b_2
32       for(k=0;k<N_CORE;k++){
33          for(i=0;i<dim2M_A;i++){
34             for(j=0;j<dim2M_A;j++){
35                QP[k].b_1[j]+=-M_C[k][0][i]*M_A[k][i][j];
36             }
37             for(j=0;j<N_CORE+1;j++){
38                QP[k].b_2[j]+=-M_C[k][0][i]*M_B[k][i][j+1];
39             }
40          }
41       }
42
43
44       // State initialzation of observers
45       for(k=0;k<N_CORE;k++){
46          for(i=0;i<2;i++){
47             QP[k].x_obsv[i]=M_X0[k][0][i];
48          }
49       }
50
51
52       // Setting up QProblem object.
53       QProblem MPC_1(numPar,numVinc);
54       QProblem MPC_2(numPar,numVinc);
55       QProblem MPC_3(numPar,numVinc);
56       QProblem MPC_4(numPar,numVinc);
57
58
59       // Plant
60       for(j=0;j<dim1P_C;j++){
61          Temp_plant[j]=0;
62          for(k=0;k<dim2P_C;k++){
63             Temp_plant[j]+=P_C[j][k]*P_X0[k][0];
64          }
65       }
66       writeVect(Temp_plant, dim1P_C, fout_Temp);
```

In this piece of code the variables are initialized. The number of optimization variables and constraints are set, the constant parameters of the QP problem are set inside the vector of *QP_Object*, the *QProblem* object of the qpOASES library are defined (*MC_1*,...,*MPC_4*), and the plant is initialized.

```
1
2  //------------------------------------------------------------
3  // Algorithm
```

```
4  //---------------------------------------------------------------
5
6
7     //-------------------------------------------------------------------------
8     //             Cold start
9     //-------------------------------------------------------------------------
10
11    // Reading of inputs
12    readVect(N_CORE+1,fin_Freq,inFreq);
13    readVect(N_CORE,fin_CPI,inCPI);
14
15
16    // Frequency to power conversion
17    for(i=0;i<N_CORE;i++){
18       param[0]=inCPI[i];
19       P_static = Psta(param);
20       inPow[i]=f2p( inFreq[i], param);
21       inPow[i]+=P_static;
22    }
23    inPow[N_CORE]=inFreq[N_CORE];
24
25
26    // Input grouping: u_i=[P_i Tenv Tvicini(tutti)]
27    u_neig_tot[0]=inPow[N_CORE];
28    input[1]=inPow[N_CORE];
29    for(j=0;j<N_CORE;j++){
30       input[j+2]=Temp_plant[j];
31       u_neig_tot[j+1]=0.0;
32    }
33
34
35    // Plant future state
36    for(j=0;j<dim1P_A;j++){
37       x_plant[j]=0;
38       for(k=0;k<dim1P_A;k++){
39          x_plant[j]+=P_A[j][k]*P_X0[k][0];
40       }
41       for(k=0;k<dim2P_B;k++){
42          x_plant[j]+=P_B[j][k]*inPow[k];
43       }
44    }
45
46
47    // Observers states
48    for(k=0;k<N_CORE;k++){
49       input[0]=inPow[k];
50       inno=0.0;
51       for(i=0;i<dim3M_C;i++){
52          inno+=M_C[k][0][i]*M_X0[k][0][i];
53       }
54       inno=Temp_plant[k]-inno;
55       for(i=0;i<dim2M_A;i++){
```

C. ACCURATE MODEL

```
56          QP[k].x_obsv[i]=0.0;
57          for(j=0;j<dim3M_A;j++){
58             QP[k].x_obsv[i]+=M_A[k][i][j]*M_X0[k][0][j];
59          }
60          for(j=0;j<dim3M_B;j++){
61             QP[k].x_obsv[i]+=M_B[k][i][j]*input[j];
62          }
63          QP[k].x_obsv[i]+=K_Obsv[k][i]*inno;
64       }
65    }
66
67
68    // Plant output updating
69    for(j=0;j<dim1P_C;j++){
70       Temp_plant[j]=0;
71       for(k=0;k<dim2P_C;k++){
72          Temp_plant[j]+=P_C[j][k]*x_plant[k];
73       }
74    }
75    writeVect(Temp_plant, dim1P_C, fout_Temp);
76
77
78    // Reading of input predictions
79    readVect(N_CORE+1,fin_Freq,inFreq);
80    readVect(N_CORE,fin_CPI,inCPI);
81
82
83    // Frequency to power conversion
84    for(i=0;i<N_CORE;i++){
85       param[0]=inCPI[i];
86       P_static = Psta(param);
87       inPow[i]=f2p( inFreq[i], param);
88       inPow[i]+=P_static;
89    }
90    inPow[N_CORE]=inFreq[N_CORE];
91
92
93    // Updating of g and input array total=[Tenv Tneights] building
94    u_neig_tot[0]=inPow[N_CORE];
95    for(j=0;j<N_CORE;j++){
96       QP[j].g[0]=-inPow[j]*2;
97       u_neig_tot[j+1]=QP[j].x_obsv[0];
98    }
99
100
101   // Computing ubA (b) --as---> uba= y_max - b_1*x_obsv - b_2*Tenv
102   for(j=0;j<N_CORE;j++){
103      QP[j].ubA[0]=TEMP_LIM+QP[j].b_1[0]*QP[j].x_obsv[0] +
       QP[j].b_1[1]*QP[j].x_obsv[1] + QP[j].b_2[0]*u_neig_tot[0] +
       QP[j].b_2[1]*u_neig_tot[1] + QP[j].b_2[2]*u_neig_tot[2] +
       QP[j].b_2[3]*u_neig_tot[3] + QP[j].b_2[4]*u_neig_tot[4];
104   }
```

```
105
106
107     // Cold QP solutions
108     nWSR=NUM_IT_QP_SOLVER;
109     gettimeofday(&t_start,NULL);  // <--------------- TIC
110     MPC_1.init(QP[0].H,QP[0].g,QP[0].A,QP[0].lb,NULL,NULL,QP[0].ubA, nWSR,0
         );nWSR=NUM_IT_QP_SOLVER;
111     MPC_2.init(QP[1].H,QP[1].g,QP[1].A,QP[1].lb,NULL,NULL,QP[1].ubA, nWSR,0
         );nWSR=NUM_IT_QP_SOLVER;
112     MPC_3.init(QP[2].H,QP[2].g,QP[2].A,QP[2].lb,NULL,NULL,QP[2].ubA, nWSR,0
         );nWSR=NUM_IT_QP_SOLVER;
113     MPC_4.init(QP[3].H,QP[3].g,QP[3].A,QP[3].lb,NULL,NULL,QP[3].ubA, nWSR,0
         );nWSR=NUM_IT_QP_SOLVER;
114     gettimeofday(&t_stop,NULL);      // <--------------- TOC
115     timersub(&(t_stop),&(t_start),&(t_step));
116     time[0]=(t_step.tv_sec + t_step.tv_usec/1000000.0)/N_CORE;
117
118
119     // Storing of the controlled powers values
120     MPC_1.getPrimalSolution(&(u_found_pow[0]));
121     MPC_2.getPrimalSolution(&(u_found_pow[1]));
122     MPC_3.getPrimalSolution(&(u_found_pow[2]));
123     MPC_4.getPrimalSolution(&(u_found_pow[3]));
124     u_found_pow[N_CORE]=inPow[N_CORE];
125
126
127     // Power -> Frequency & Frequency -> Power conversions
128     for(i=0;i<N_CORE;i++){
129        param[0]=inCPI[i];
130        P_static=Psta(param);
131        P_dyn=u_found_pow[i]-P_static;
132        if(P_dyn<0) P_dyn = 0;
133        if (f2p(F_MIN,param)>=P_dyn){
134           u_found_freq[i]=F_MIN;
135        }
136        else{
137           if(f2p(F_MAX,param)<=P_dyn){
138              u_found_freq[i]=F_MAX;
139           }
140           else{
141              pinv[0]=param[0];
142              pinv[1]=P_dyn;
143              u_found_freq[i]=Amsterdam_Method(f2p_inv, F_MIN, pinv, F_MAX,
          TOLLERANZA, MAX_ITERATION, &err);
144           }
145        }
146        u_found_pow[i]=f2p( u_found_freq[i], param);
147        u_found_pow[i]+=P_static;
148     }
149
150     for(j=0;j<N_CORE;j++){
151        QP[j].u_found_pow[0]=u_found_pow[j];
```

```
152         QP[j].u_found_pow[1]=inPow[N_CORE];
153     }
```

Although efficient QP solvers based on active-set methods and interior point methods are available, the computational overhead for finding the solution demands significant on-line computation effort. The solving algorithm implemented in qpOASES library is more efficient, since, after having computed the first solution of a QP problem, it can compute the new solution starting the search from the previous one (this property is named *hot start*). In the reported part of code we show the algorithm to find the first solution of each QP problem (we called it *cold start*).

The first sample of the input trace is read. The function *readVect* reads the frequency and the CPI of each core from the input file. Then, the frequency is converted to power using the function *f2p*, and the input vector for each single-core model is prepared in order to estimate the future temperature of the plant. The plant states, the observer states, and the plant output are computed. The estimations of the next input are read (we assume to know exactly the workload of the next sample interval), the time-varying matrices of the QP problem are updated, and then each QP problem is solved invoking the function *MPC_i.init* of the qpOASES library. The function *MPC_i.getPrimalSolution* (of the qpOASES library) assigns to the controlled power vector *u_found_pow* the solution of the problems. Notice that the solving time is computed using the function *gettimeofday* subtracting to the time *t_stop*, saved after the QP problem has been solved, the time *t_start*, saved at the beginning of the computation. Finally the controlled power of the cores obtained by solving the QP problems is converted in frequency using the *Amsterdam_Method* function.

```
1
2     //-------------------------------------------------------------------------
3     //             Hot start
4     //-------------------------------------------------------------------------
5
6     for (i=0;i<NITER-1;i++){
7
8        // Plant future state
9        for(j=0;j<dim1P_A;j++){
10          x_plant_old[j]=x_plant[j];
11       }
12
13       for(j=0;j<dim1P_A;j++){
14          x_plant[j]=0.0;
15          for(k=0;k<dim1P_A;k++){
16             x_plant[j]+=P_A[j][k]*x_plant_old[k];
```

```
17          }
18          for(k=0;k<dim2P_B;k++){
19             x_plant[j]+=P_B[j][k]*u_found_pow[k];
20          }
21       }
22
23
24       // Storing past Temperature & grouping of neighbors in the input array
25       for(j=0;j<N_CORE;j++){
26          Temp_plant_old[j]=Temp_plant[j];
27          for(k=0;k<N_CORE;k++){
28             QP[k].u_found_pow[j+2]=Temp_plant[j];
29          }
30       }
31
32
33       // Plant outputs updating
34       for(j=0;j<dim1P_C;j++){
35          Temp_plant[j]=0.0;
36          for(k=0;k<dim2P_C;k++){
37             Temp_plant[j]+=P_C[j][k]*x_plant[k];
38          }
39       }
40       writeVect(Temp_plant, dim1P_C, fout_Temp);
41
42
43       // Start Clock
44       gettimeofday(&t_start,NULL); // <--------------- TIC
45
46
47       // Observers states
48       for(k=0;k<N_CORE;k++){
49          for(j=0;j<dim2M_A;j++){
50             x_obsv_old[j]=QP[k].x_obsv[j];
51          }
52          inno=0.0;
53          for(j=0;j<dim3M_C;j++){
54             inno+=M_C[k][0][j]*x_obsv_old[j];
55          }
56          inno=Temp_plant_old[k]-inno;
57          for(ii=0;ii<dim2M_A;ii++){
58             QP[k].x_obsv[ii]=0.0;
59             for(j=0;j<dim3M_A;j++){
60                QP[k].x_obsv[ii]+=M_A[k][ii][j]*x_obsv_old[j];
61             }
62             for(j=0;j<dim3M_B;j++){
63                QP[k].x_obsv[ii]+=M_B[k][ii][j]*QP[k].u_found_pow[j];
64             }
65             QP[k].x_obsv[ii]+=K_Obsv[k][ii]*inno;
66          }
67       }
68
```

C. ACCURATE MODEL

```
69
70        // Reading of inputs
71        readVect(N_CORE+1,fin_Freq,inFreq);
72        readVect(N_CORE,fin_CPI,inCPI);
73
74
75        // Frequency --> Power conversion
76        for(j=0;j<N_CORE;j++){
77           param[0]=inCPI[j];
78           P_static = Psta(param);
79           inPow[j]=f2p( inFreq[j], param);
80           inPow[j]+=P_static;
81        }
82        inPow[N_CORE]=inFreq[N_CORE];
83        u_neig_tot[0]=inPow[N_CORE];
84
85
86        // Update of g and input array totale=[Tenv Tneights] building
87        for(j=0;j<N_CORE;j++){
88           QP[j].g[0]=-inPow[j]*2;
89           u_neig_tot[j+1]=QP[j].x_obsv[0];
90        }
91
92
93        // Computing ubA (b)  --as--> uba= y_max - b_1*x_obsv- b_2*Tenv
94        for(j=0;j<N_CORE;j++){
95           QP[j].ubA[0]=TEMP_LIM+QP[j].b_1[0]*QP[j].x_obsv[0] +
       QP[j].b_1[1]*QP[j].x_obsv[1] + QP[j].b_2[0]*u_neig_tot[0] +
       QP[j].b_2[1]*u_neig_tot[1] + QP[j].b_2[2]*u_neig_tot[2] +
       QP[j].b_2[3]*u_neig_tot[3] + QP[j].b_2[4]*u_neig_tot[4];
96      }
97
98
99        // Hot QP solutions
100       MPC_1.hotstart(QP[0].g,QP[0].lb,NULL,NULL,QP[0].ubA, nWSR,0
       );nWSR=NUM_IT_QP_SOLVER;
101       MPC_2.hotstart(QP[1].g,QP[1].lb,NULL,NULL,QP[1].ubA, nWSR,0
       );nWSR=NUM_IT_QP_SOLVER;
102       MPC_3.hotstart(QP[2].g,QP[2].lb,NULL,NULL,QP[2].ubA, nWSR,0
       );nWSR=NUM_IT_QP_SOLVER;
103       MPC_4.hotstart(QP[3].g,QP[3].lb,NULL,NULL,QP[3].ubA, nWSR,0
       );nWSR=NUM_IT_QP_SOLVER; // !!!! <------------- ogni volta viene azzerato
       (tenuto quello del ciclo precedente) !!!!
104
105
106
107       // Storing of the controlled powers values
108       MPC_1.getPrimalSolution(&(u_found_pow[0]));
109       MPC_2.getPrimalSolution(&(u_found_pow[1]));
110       MPC_3.getPrimalSolution(&(u_found_pow[2]));
111       MPC_4.getPrimalSolution(&(u_found_pow[3]));
112       u_found_pow[N_CORE]=inPow[N_CORE];
```

```
113
114
115         // Power -> Frequency conversion
116         for(j=0;j<N_CORE;j++){
117            param[0]=inCPI[j];
118            P_static=Psta(param);
119            P_dyn=u_found_pow[j]-P_static;
120            if(P_dyn<0)P_dyn = 0;
121            if (f2p(F_MIN,param)>=P_dyn){
122               u_found_freq[j]=F_MIN;
123            }
124            else{
125               if(f2p(F_MAX,param)<=P_dyn){
126                  u_found_freq[j]=F_MAX;
127               }
128               else{
129                  pinv[0]=param[0];
130                  pinv[1]=P_dyn;
131                  u_found_freq[j]=Amsterdam_Method( f2p_inv, F_MIN, pinv, F_MAX,
      TOLLERANZA, MAX_ITERATION, &err);
132               }
133            }
134
135         }
136
137
138         // Stop Clock
139         gettimeofday(&t_stop,NULL); // <-------------- TOC
140         timersub(&(t_stop),&(t_start),&(t_step));
141         time[i+1]=(t_step.tv_sec + t_step.tv_usec/1000000.0)/N_CORE;
142
143         // Frequency --> Power conversion
144         for(j=0;j<N_CORE;j++){
145            param[0]=(float)inCPI[j];
146            P_static=Psta(param);
147            u_found_pow[j]=f2p(u_found_freq[j], param);
148            u_found_pow[j]+=P_static;
149         }
150
151         for(j=0;j<N_CORE;j++){
152            QP[j].u_found_pow[0]=u_found_pow[j];
153            QP[j].u_found_pow[1]=inPow[N_CORE];
154         }
155      }
156
157
158   // Writing of the time file
159   for(i=0;i<NITER;i++)
160      fprintf(fout_Time,"%f\n",time[i]);
161
162   return 0;
163   }
```

The code described for the cold start, is repeated for all the other sampling times with a loop. The only difference regards the name of the function called for solving the QP problems that is *MPC_i.hotstart* that allows the algorithm to compute the next solution starting from the previous one.

```
//-----------------------------------------------------------------
// Functions
//-----------------------------------------------------------------

// Reading of a vector from a file
void readVect(int dim,FILE* fin, double* vett)
{
   int i;
   char ch;
   for(i=0;i<dim;i++){
      fscanf(fin,"%lf ",&vett[i]);
   }
   fscanf(fin,"\r%c\n",&ch);
}


// Writing of a vector in a file
void writeVect(double* y, int dim, FILE* fout)
{
   int i;
   for(i=0;i<dim;i++){
      fprintf(fout,"%f ",y[i]);
   }
   fprintf(fout,";\r\n");
}


// Writing of a vector on the Monitor
void printVect(double* vett, int dim)
{
   int i;
   for(i=0;i<dim;i++)
      printf("%f ",vett[i]);
   printf(";\n");
}

// Writing of a matrix on the Monitor
void printMat(double* mat, int righe, int columns)
{
   int i,j;
   for(i=0;i<righe;i++){
      for(j=0;j<columns;j++){
         printf("%f ",mat[i*columns+j]);
```

```
44        }
45     printf(";\n");
46     }
47  }
48
49
50
51  // Functions for frequency conversions
52
53  double Psta(float *p)
54  {
55     return (double)(Z_DEFAULT*p[3]*p[2]*p[2]*exp((-q*(p[4]))/(K*p[2])));
56  }
57
58  double f2p(double freq, float  *p)
59  {
60      double Pdyn;
61      if (p[0]!=0){
62        Pdyn = (Ka_DEFAULT*(pow((freq),Kb_DEFAULT)) + Kc_DEFAULT) +
       (Ke_DEFAULT+(Kd_DEFAULT*freq))*pow(p[0],Kf_DEFAULT);}
63     else{Pdyn=0;}
64          Pdyn = Pdyn*p[1];
65      return Pdyn ;
66  }
67
68  double f2p_inv(double freq, float  *p)
69  {
70     return ((Ka_DEFAULT*(pow((freq),Kb_DEFAULT))+ Kc_DEFAULT +
       (Ke_DEFAULT+(Kd_DEFAULT*freq))*pow(p[0],Kf_DEFAULT)) - p[1]);
71  }
```

Finally the implementation of the functions.

C.4 The complex MPC control solutions

In this Section we provided the code of the complex control solutions presented in Chapter 5. These algorithms re-call the functions shown in the previous Section.

C.4.1 A feasible two-layer distributed MPC approach to thermal control of Multiprocessor Systems on Chip

The simulation of the two-layer solution has been entirely executed in the Simulink environment. The block diagram of the solution is represented in Fig. C.6.

The scheme comprises four main blocks:

Inputs contains some typical benchmark traces selectable with the *trace_selector* block;

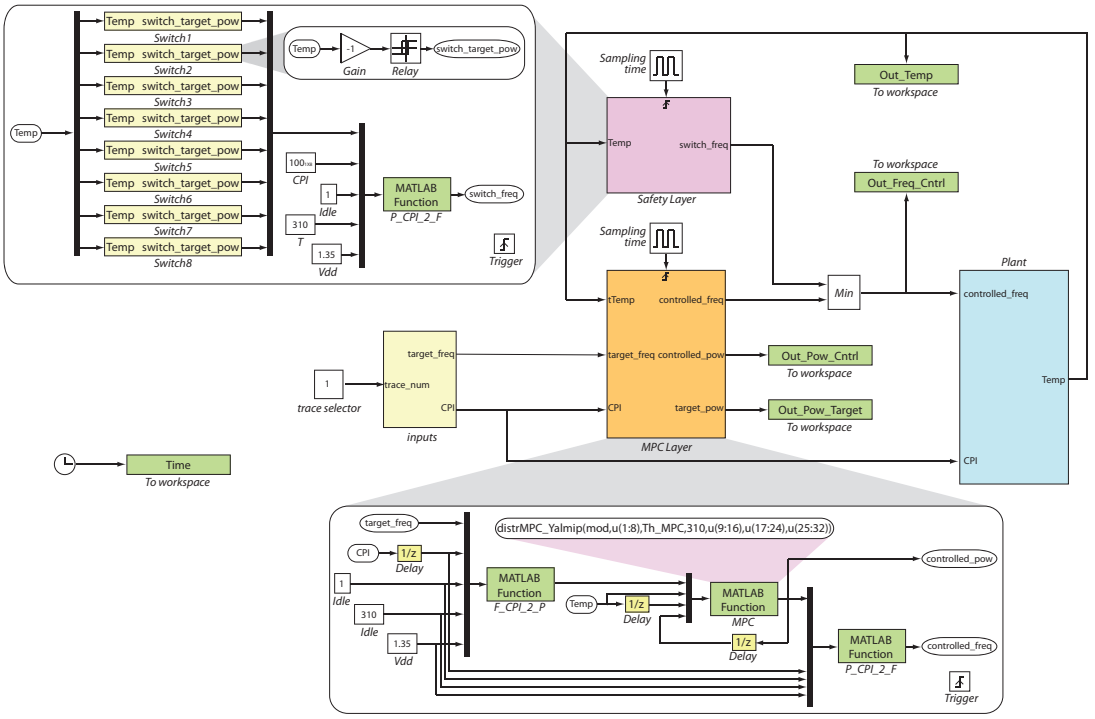


Figure C.6: Simulink block diagram of the two-layer solution.

MPC Layer contains the distributed MPC thermal controller;

Safety Layer contains the switch controllers;

Plant contains the accurate thermal model of the processor.

The *Inputs* and *Plant* blocks are not described here. We will focus on the *MPC Layer* and the *Safety Layer* blocks. With regard to the former, the block receives the frequency and the CPI values as inputs. Each sampling interval (10*ms*) these values are converted into a power consumption requirement (using the block *F_CPI_2_P*), which is an input of the *MPC* Matlab function block. This block calls the function *distrMPC_Yalmip*, already shown in the Paragraph C.3.2.2, in order to solve the optimization problem of each local controller. The function returns the controlled power of all cores which is converted into frequency by the function block *P_CPI_2_F*. This frequency vector is the output of the *MPC Layer* block. It is worth to note that inside the *MPC Layer* block three delay blocks are present (highlighted in yellow). The first on the left simulates the unpredictability of the workload (i.e. the CPI at the current time is equal to the one computed in the previous sampling interval). The other two store the

past temperature and power consumption of each core in order to estimate the unknown states of each single-core model inside the *MPC block*.

The *Safety Layer* block, instead, contains the set of switch controllers used for guaranteeing the feasibility of the global controller. As it is possible to see in Fig. C.6, each switch controller (one per core) can be simulated using a standard Simulink block, the *Relay* block, which implements an hysteresis function. The gain block is only needed to adopt our requirements to the hysteresis block. The *Safety Layer* block takes as inputs the temperatures measured on the cores. The temperature of the i-th core enters as input into the correspondent *Switch#i* block. At each sampling time ($0.1ms$), each temperature is compared to the switch temperature threshold (τ_{SWITCH}). If the temperature violates the threshold the controller trims the power consumption of that core to the minimum value (P_{MIN}) until the temperature decrease below the lower hysteresis threshold. Otherwise the controller return the maximum power consumption. The power is finally converted in frequency and given as output of the *Safety Layer* block.

Notice that a *minimum* block allows the controller to choose, for each core, the correct frequency between the one outgoing from the *MPC Layer* block and the one outgoing from the *Safety Layer* block. That frequency feeds the cores of the plant.

C.4.2 Communication-aware solution

We reported below the code used for simulating the communication-aware solution.

```
%% 5

    %VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
    %|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
    %%%%%%%%%%%%%%%%%%%%%%%%                         %%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%       CONTROLLER        %%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%                         %%%%%%%%%%%%%%%%%%%%%%%%%%
    %|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
    %AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

    %  Communication-Aware Approach:
    %   - performance maximization
    %   - thermal capping
    %   - matching frequency between cores



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%% Parameters
Vdd=1.35;       % Supply voltage
IDLE=1;
```

C. ACCURATE MODEL

```
F_MAX=3000;     % maximum frequency
F_MIN=1600;     % minimum frequency
P_MIN=F_CPI_2_P(1600,100,IDLE,Tenv(1,1),Vdd);   % minimum power consumption
time_trace = 30000;                             % trace duration


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%% MPC Controller Init
T_CRIT=360*ones(NC,1);      % critical temperature
Vy_max=zeros(NC,1);         % 0=hard 1=soft constraints
rho=10^5;                   % weight of the slack variable
weight=100;                 % weight of the communication
w=1;                        % weight of the power error
Q_qp=w*eye(NC);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%% Benchmarks
% Time
clear power_tot_target power_cache_target power_cores_target frequency_cores_target
    CPI_cores_target

% Inputs
frequency_cores_target = idinput([time_trace,NC],'prbs',[0 0.005],[1600 3000]);
CPI_cores_target=1.5*ones(time_trace,NC);    % Note: we know the future CPI
CPI_cores_target(:,2) = 0.5+(100-0.5).*rand(time_trace,1);

% Target Power (Conversion using the Power Model function CPI2Pow)
power_cores_target=CPI2Pow(frequency_cores_target,CPI_cores_target,IDLE,Tenv(1,1),Vdd);

% Caches Power (the 30% of the adjacent core powers)
clear power_caches_target power_tot_target
power_caches_target(:,1) =
    ((power_cores_target(:,1)+power_cores_target(:,3))./2).*percentuale;
power_caches_target(:,2) =
    ((power_cores_target(:,2)+power_cores_target(:,4))./2).*percentuale;
power_caches_target(:,3) =
    ((power_cores_target(:,5)+power_cores_target(:,7))./2).*percentuale;
power_caches_target(:,4) =
    ((power_cores_target(:,6)+power_cores_target(:,8))./2).*percentuale;

% Total Power
power_tot_target=[power_cores_target(:,1) power_caches_target(:,1)./2
    power_caches_target(:,2)./2 power_cores_target(:,2) power_cores_target(:,3)
    power_caches_target(:,1)./2 power_caches_target(:,2)./2 power_cores_target(:,4)
    zeros(time_trace,4) power_cores_target(:,5) power_caches_target(:,3)./2
    power_caches_target(:,4)./2 power_cores_target(:,6) power_cores_target(:,7)
    power_caches_target(:,3)./2 power_caches_target(:,4)./2 power_cores_target(:,8)];


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%% Initialization
% Plant
```

```
sysd=c2d(TModel,Time.Step,'zoh');   % Discretization of the plant
x_plant=zeros(time_trace-1,size(A,1));
x_plant(1,:)=X0';
Temp_plant=zeros(time_trace,NC);
Temp_plant(1,:)=(sysd.C*X0)';

% Controlled Inputs
power_cores_cntrl=zeros(time_trace,NC);
power_tot_cntrl=zeros(time_trace,N_COMP);
frequency_cores_cntrl=zeros(time_trace,NC);

% Observer
x_obsv(1,:)=model.x0';
K_obs=(place(model.a',model.c',(eig(model.a).*0.4)))';
```

We skip the code used for the implementation of the accurate model and the identification of the prediction model (this solution is not distributed). In the first part of the code the usual parameters of the controller are defined. Subsequently, the input benchmark is defined, the frequency (a PRBS signal) and the CPI (a random input) are translated in power requirements. The plant and the observer are initialized.

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%% Simulazione
for j=1:time_trace
    % Calcolo uscita plant
    Temp_plant(j,:)=sysd.c*x_plant(j,:)';

    % Communication request
    if j<(time_trace/3)
        Communicating_Core_vector=[1 1 0 0 0 0 0 0];
    else
        Communicating_Core_vector=[1 0 0 0 1 0 0 0];
    end

    % Matrices Update
    [Q_additive g_additive]=communication(Communicating_Core_vector,
    CPI_cores_target(j,:), F_MAX, F_MIN, weight);
    g_qp=(-2*power_cores_target(j,:)*Q_qp)';
    g_qp_tot=g_qp+g_additive';
    Q_qp_tot=[Q_qp.*2+Q_additive zeros(size(Q_qp,1),1); zeros(1,size(Q_qp,1))
    rho.*2];

    % QP problem
    clear pow
    pow=sdpvar(NC+1,1);
    Objective=[0.5*pow'*Q_qp_tot*pow+pow'*[g_qp_tot;0]];
    Constraints=[[model.c* model.b(:,1:NC) -Vy_max]* pow <= T_CRIT-model.c* model.a*
    x_obsv(j,:)'-model.c* model.b(:,NC+1:end)* Tenv(j,1),pow(1:NC,1) >=
```

C. ACCURATE MODEL

```
    0.254043439071652];
    Options=sdpsettings('verbose',0,'solver','');
    sol = solvesdp(Constraints,Objective,Options);
    power_cores_cntrl(j,1:NC)=double(pow(1:NC,1));


    % Power-to-frequency conversion
    frequency_cores_cntrl(j,:)=
    Pow2Freq_main2(power_cores_cntrl(j,:),CPI_cores_target(j,:),IDLE,Tenv(1,1),Vdd);

    % Frequency-to-Power conversion
    power_cores_cntrl(j,:)=
    CPI2Pow(frequency_cores_cntrl(j,:),CPI_cores_target(j,:),IDLE,Tenv(1,1),Vdd);

    % Total power (we add the power of the caches)
    power_tot_cntrl(j,:)= [power_cores_cntrl(j,1),
    (power_cores_cntrl(j,1)+power_cores_cntrl(j,3))./4.*percentuale,
    (power_cores_cntrl(j,2)+power_cores_cntrl(j,4))./4.*percentuale,
    power_cores_cntrl(j,2:3),
    (power_cores_cntrl(j,1)+power_cores_cntrl(j,3))./4.*percentuale,
    (power_cores_cntrl(j,2)+power_cores_cntrl(j,4))./4.*percentuale,
    power_cores_cntrl(j,4), zeros(1,4), power_cores_cntrl(j,5),
    (power_cores_cntrl(j,5)+power_cores_cntrl(j,7))./4.*percentuale,
    (power_cores_cntrl(j,6)+power_cores_cntrl(j,8))./4.*percentuale,
    power_cores_cntrl(j,6:7),
    (power_cores_cntrl(j,5)+power_cores_cntrl(j,7))./4.*percentuale,
    (power_cores_cntrl(j,6)+power_cores_cntrl(j,8))./4.*percentuale,
    power_cores_cntrl(j,8)];

    % The future state of the plant
    x_plant(j+1,:)= (sysd.a*x_plant(j,:)'+sysd.b*[power_tot_cntrl(j,:) Tenv(j,1)]')';

    % Observer
    x_obsv(j+1,:)= model.a*x_obsv(j,:)'+ model.b*[power_cores_cntrl(j,:)
    Tenv(j,1)]'+ K_obs*(Temp_plant(j,:)'- model.c*x_obsv(j,:)');

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Visualization
core1=[1 2 5];
ax(1) =subplot(611); plot(1:1:time_trace-1, Temp_plant(1:time_trace-1,core1));
    title('Temperature'); legend('MPC','Location','Best');
ax(2) =subplot(612); plot(1:1:time_trace-1,
    frequency_cores_cntrl(1:time_trace-1,core1),'LineWidth',2); title('Provided
    Frequency')
ax(3) =subplot(613); plot(1:1:time_trace-1,
    power_cores_cntrl(1:time_trace-1,core1),'LineWidth',2); title('Provided power');%
ax(4) =subplot(614); plot(1:1:time_trace-1,
    frequency_cores_target(1:time_trace-1,core1),'LineWidth',2); title('Target
    Frequency')
```

```
ax(5) =subplot(615); plot(1:1:time_trace-1,
    power_cores_target(1:time_trace-1,core1)); title('Target power');
ax(6) =subplot(616); plot(1:1:time_trace-1, CPI_cores_target(1:time_trace-1,core1));
    title('Target CPI');
linkaxes(ax,'x');
```

In this part of the code the simulation of the input trace is performed. The code is similar to the code shown in the previous example, except for two minor differences. First, the vector *Communicating_Core_vector* which contains the information on the communicating cores. The dimension of the vector corresponds to the number of cores. The indexes of the elements with 1 represent the cores that must have the same frequency. In the code we imposed the communication between the cores 1 and 2 for the first 10 seconds and between the cores 1 and 5 for the rest of the simulation. Notice that the *Communicating_Core_vector* vector is assumed to be provided by a high level software manager. The second difference regards the use of the function *communication* that takes as inputs the *Communicating_Core_vector*, the target CPI vector, the maximum and the minimum frequencies and the weight constant of the communication and it returns the arrays *Q_additive* and *g_additive*. Assuming the QP problem to be solved to find the control decision has, in the nominal case (that is the case without cores communications) the form,

$$\min_{pow} \frac{1}{2} \cdot pow^T(t) \cdot Q_{qp} \cdot pow(t) + g_{qp}^T \cdot pow(t) \tag{C.4a}$$

$$s.t.$$

$$A_{qp} \cdot pow(t) \leq b_{qp} \tag{C.4b}$$

then, the arrays *Q_additive* and *g_additive* added to the matrices Q_{qp} and g_{qp} allows the controller to take into account the communication between the cores as explained in Section 5.3.1.1. Notice that the communication between more than two cores is possible by calling multiple times the function *communication*.

C.4.2.1 *communication.m*

In the code below the *communication* function is shown

```
function [H_additive g_additive]=communication(Communicating_Core_vector,
    CPI_vector, F_MAX, F_MIN, weight)
% COMMUNICATION returns the weight matrices of the cost function of the QP
% problem.
```

C. ACCURATE MODEL

```
4  %
5  % The input parameters:
6  %   - Communicating_Core_vector : vector with dimension 1 x n_cores, the
7  %     indexes of the element equal to 1 are the core communicating the other
8  %     elements are 0 (only two value can be different from zero)
9  %   - CPI_vector : vector containing the CPI of all cores
10 %   - F_MAX,F_MIN : maximum and minimum value of the frequency
11 %   - weight : weight value for the communication in the QP problem
12 %
13 %
14 % Example:
15 %
16 % Communicating_Core_vector=[1 1 0 0 0 0]; % the cores are 6, the communicatin core
       are the 1 and the 2
17 % CPI_vector=[0.1 1 23 100 0.5 8];
18 % F_MAX=3000;     F_MIN=1600;     weight=100;
19 % [H_additive g_additive]=communication(Communicating_Core_vector, CPI_vector,
       F_MAX, F_MIN, weight)
20
21
22 if size(Communicating_Core_vector)~=size(CPI_vector)
23         error('communication:communication:none','The first two inputs must have the
       same dimensions');
24 end
25
26 if sum(Communicating_Core_vector)~=2
27         error('communication:communication:none','Only two core can communicate!
       Modify the CPI vector');
28 end
29
30 % Power Model Fitting Parameters
31 KA1=3.8696e-008;
32 KA2=1.1025;
33 KB=2.4090;
34 KC=-4.1376;
35 KD=0.0051;
36 KE=-0.3016;
37
38 % Finding of the communicating cores
39 index = find(Communicating_Core_vector ~= 0);
40
41 % Preparing of the parameters for computing alpha and beta
42 Delta_CPI=CPI_vector(index(1))^KE-CPI_vector(index(2))^KE;
43
44 F_lim=[F_MIN, F_MAX];
45 P_lim=(KA1.*F_lim.^KB +KA2) + (KC+KD.*F_lim).*(CPI_vector(index(2))).^KE;
46
47 Delta_P=P_lim(2)-P_lim(1);
48 Delta_F=F_lim(2)-F_lim(1);
49
50 % Computing of alpha and beta
51 alpha=KD*Delta_F*Delta_CPI/Delta_P;
```

```
52 beta=(KC+(KD*(F_lim(1)*P_lim(2)-F_lim(2)*P_lim(1)))/Delta_P)*Delta_CPI;
53
54 % Computing of the H_additive array
55 H_additive=zeros(max(size(CPI_vector)));
56 H_additive(index(1),index(1))=2*weight;
57 H_additive(index(1),index(2))=-2*weight*(1+alpha);
58 H_additive(index(2),index(1))=-2*weight*(1+alpha);
59 H_additive(index(2),index(2))=2*weight*(1+alpha)^2;
60
61 % Computing of the g_additive array
62 g_additive=zeros(1,max(size(CPI_vector)));
63 g_additive(index(1))=-2*beta*weight;
64 g_additive(index(2))=2*beta*weight*(1+alpha);
```

The code simply computes the α and β parameters as shown in equation (5.22) in Section 5.3.1.1. Then the two parameters are used to create the *Q_additive* and *g_additive* matrices as shown in equation (5.26).

C.4.3 Guaranteed Re-sprinting in MPSoCs exploiting MPC

The simulation of the re-sprinting solution has been executed in the Simulink environment in order to simplify the management of the sampling time. Indeed the plant is continuous time, whereas the two hierarchical MPC controllers has different sampling times. The block diagram of the solution is represented in Fig. C.7.

The scheme comprises five main parts:

1. On the left of Fig. C.7, we can see a *Multiport switch* block used for selecting the desired input trace. The output of the block is a vector signal containing the power consumptions of the 16 cores (remember that the first one is the power of the leader core which is always active).

2. The *U bound* block contains the time-varying limit on PCM internal energy.

3. The *PCM MPC layer* block contains the centralized MPC controller used for the PCM management. As explained in Chapter 5 this control layer is necessary for ensuring the re-sprinting capabilities of the regulated system when mixed criticalities tasks are present.

4. The *Thermal MPC layer* block contains the distributed MPC thermal controller.

5. The *plant* block contains the accurate thermal model of the processor.

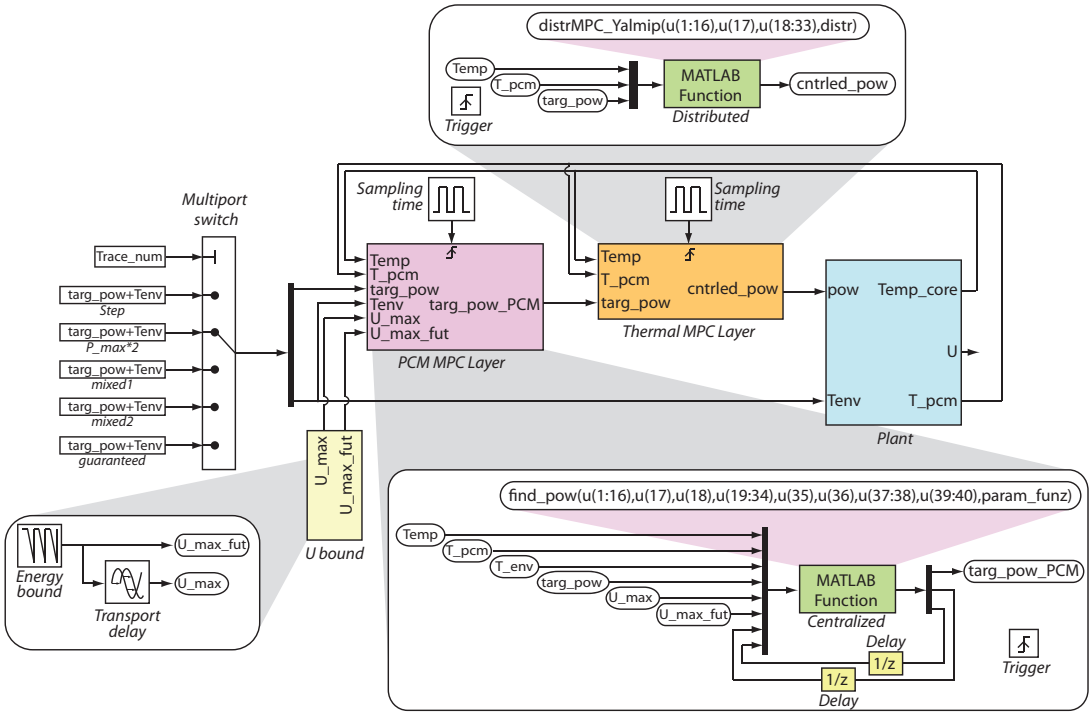


Figure C.7: Simulink block diagram of the re-sprinting solution.

The part of the code related to the inputs and the plant are skipped in order to focus our attention on the control part of the code. However, before illustrating the details of the remaining parts of the block diagram we need to show the script where the parameters used in the code are defined.

C.4.3.1 Simulation Initialization

```
%Thermal + Energy control based on MPC (ALL CORES CONTROLLED EXCEPT CORE1)+ Frequency

% Outline:
% 1 - Parameters initialization
% 2 - Thermal Model Generationn
% 3 - Prediction model generation
% 4 - MPC controller definition

clear all
clc

% ********************************************************** %
% ******************    (1) PARAMETERS     ******************* %

```

```
N_CORE = 16 ;      % Number Of Cores

% Ambient temperature
Tenvironment_max = 273+45;      % [K] Maximum ambient temperature
Tenvironment = 273+25;          % [K] Nominal ambient temperature

% Power data of the cores
P_MAX=16;                   % [W] Chip maximum power (all cores=1W)
P_MIN=1+15*0.05;            % [W] Chip minimum power (core1=1W, other=0.05W)
P_max=1;                    % [W] Maximum power consumption of one core
P_min=0.15;                 % [W] minimum power consumption of one core
P_idle=0.05;                % [W] Idle power of the cores

% PCM layer parameters (mixed Cu + Climsel C70)
T_melt=273+70;              % [K] PCM melting temperature
T_Sprint_Max=1;             % [s] Sprinting duration
PCM_density= 1700;          % [Kg/m^3] PCM density
PCM_spec_lat_heat=396;      % [KJ/Kg] PCM specific latent heat
% PCM specific heat Solid/Liquid
parameters.specific_heat_solid=3.526520000000000e-012;     % [J/(K*um^3)]
parameters.specific_heat_liquid=3.526520000000000e-012;    % [J/(K*um^3)]
% PCM layer area
parameters.cell_width=6800;                                % [um]
parameters.cell_height=6800;                               % [um]
% PCM resistance
Chip_Dimensions.R_PCM= 7.9;                                % [*K/W]
% Energy quantity from when PCM starts melting to when the PCM is melted
Delta_U=((P_MAX- (T_melt-Tenvironment_max)/ Chip_Dimensions.R_PCM)* T_Sprint_Max);
% PCM layer thickness
parameters.cell_thickness= (Delta_U)/ 1000/ PCM_spec_lat_heat/ PCM_density/ 10^-18/
    parameters.cell_width/ parameters.cell_height; % [um]
% Energy when PCM start melting
parameters.u_min=T_melt* parameters.specific_heat_solid* parameters.cell_width*
    parameters.cell_height* parameters.cell_thickness;
% Energy when PCM is completely melted
parameters.u_max= parameters.u_min + Delta_U;
% PCM conductivity
Cu20PCM80_conductivity=3.2012e-004;         % 20% Cu + 80% Climsel C70

% Chip Data
FileName_FLOORPLAN='floorplan.txt';
FileName_SENSORS='sensori.txt';
FileName_HOTSPOT='potenze.txt';
% Silicon layer area
Chip_Dimensions.h=parameters.cell_height;
Chip_Dimensions.L=parameters.cell_width;
% Two layers thickness
Chip_Dimensions.thick_1L=350;                       % silicon layer thickness
Chip_Dimensions.thick_2L=parameters.cell_thickness; % PCM layer thickness

% Simulation data
guaranteed_window=0.2;    % [s]
```

In this first part of the code we set the data of the processor we have simulated. First, the number of cores, the nominal and maximum ambient temperatures, and the power consumptions of the chip and of the single cores are defined. Subsequently, the PCM layer data are inserted in order to guarantee a maximum sprinting time of 1*s*, when all the cores at the maximum speed. Notice that also the energy values in which the PCM starts and finishes melting are extrapolated (*parameters.u_min* and *parameters.u_max*). The names of the files containing the layout of the cores, the locations of the sensors, and the power consumption distribution are entered. Finally, the guaranteed time window for re-sprints is defined by the user (*guaranteed_window*).

```
1
2  %%
3  % ********************************************************************** %
4  % ******************** (2) THERMAL MODEL GENERATION ********************* %
5
6  TM3=EmbeddedModeling(FileName_FLOORPLAN, FileName_SENSORS, FileName_HOTSPOT,
       Chip_Dimensions, Cu20PCM80_conductivity);
```

The code calls the function *EmbeddedModeling.m*, a function similar to *mat_modeling.m* in Section C.1.3.1, for generating the accurate model of the chip. The state vector comprises the temperatures of each core and the internal energy of the PCM cell. As output the function returns the input, output and state matrices of the model.

```
1
2  %%
3  % ********************************************************************** %
4  % ******************* (3) PREDICTION MODEL GENERATION ****************** %
5
6  model=EmbeddedCGModeling(FileName_FLOORPLAN, FileName_SENSORS, FileName_HOTSPOT,
       Chip_Dimensions, Cu20PCM80_conductivity);
7  model.x0=[Tenvironment*ones(N_CORE,1) ;parameters.u_min];
```

Then, it is called the function *EmbeddedCGModeling.m*, used for the generation of the prediction model. The function does not use any identification procedure, but physical approximations. The cells belonging to one component are reduced to a single cell by making the parallel of the vertical resistances of the equivalent electric circuit shown in Section B.4.1, neglecting the horizontal resistances of the cells inside the components, and parallelizing the horizontal resistances linking the cell of one component to the ones of another component.

```
%%
% ************************************************************************* %
% ****************** (4) MPC CONTROLLERS DEFINITION ******************** %

% ---------------->  Centralized MPC Energy manager (PCM)  <-------------- %

% Sampling time
time_centr=10e-3;

% prediction model (MPC for PCM layer)
centr.model.a=model.a(end,end);
centr.model.b=[model.a(end,1:end-1) model.b(end,end-1:end)];  % ingressi=[T1 T2 ...
    T_N_CORE T_pcm T_amb]
centr.model.c=model.c(end,end);
centr.model.d=zeros(1,N_CORE+2);
centr.model=c2d(ss(centr.model.a, centr.model.b, centr.model.c, centr.model.d),
    time_centr, 'zoh');
centr.init_state=0;

% Controller parameters if we use a QP problem solver
centr.rho=10^5;
centr.Vy_max=zeros(1,1);                % 0=hard 1=soft constraints
centr.ubA=parameters.u_max;             % maximum energy (PCM is completely melted)
centr.ubA_modified=parameters.u_max+5;  % maximum energy + margin
centr.reference=360*ones(N_CORE-1,1);       % Reference trajectory
centr.input_weight=eye(N_CORE-1);           % Hessian matrix in QP problem
% QP matrices
centr.Q_qp=[centr.input_weight zeros(size(centr.input_weight,1),1);
    zeros(1,size(centr.input_weight,1)) centr.rho].*2;
centr.f_1=2*centr.input_weight;
centr.A_qp=[centr.model.c*centr.model.b(1,2:N_CORE) -centr.Vy_max];
centr.b_1=-centr.model.c*centr.model.a;
centr.b_2=-centr.model.c*[centr.model.b(:,1) centr.model.b(:,N_CORE+1:N_CORE+2)];

% Simulink function parameters for PCM layer (if we don't use a QP solver)
param_fun.model.a=centr.model.a;
param_fun.model.b=centr.model.b;   % ingressi=[T1 T2 ... T_N_CORE T_pcm T_amb]
param_fun.model.c=centr.model.c;
param_fun.model.d=centr.model.d;
param_fun.model.x0=centr.init_state;
param_fun.R_pcm=Chip_Dimensions.R_PCM+0.02;
param_fun.P_MIN=P_MIN;
param_fun.P_max=P_max;
param_fun.P_idle=P_idle;
param_fun.P_min=P_min;
param_fun.T_melt=T_melt;
param_fun.T_amb_max=Tenvironment_max;
param_fun.Ts=time_centr;
param_fun.time_centr=time_centr;
param_fun.N_CORE=N_CORE;
```

```
param_fun.u_min=parameters.u_min;
param_fun.u_max=parameters.u_max;
param_fun.C_si=6.586887999999999e-005*400; % Silicon equivalent Capacity
param_fun.G_si=0.004166666666667*400;      % Silicon equivalent Conductance
% Energy bound: Final value of the energy bound
deltaU=(P_MAX-(T_melt-Tenvironment_max)/ param_fun.R_pcm)* guaranteed_window;
U_N=parameters.u_max-deltaU;
% Energy bound: duration of slanted side of the trapezoid
param_fun.sliding_time=deltaU/(P_MIN-(T_melt-Tenvironment_max)/param_fun.R_pcm);
param_fun.u_n=U_N;
% Margin on Energy bound parameters
tau=param_fun.C_si/param_fun.G_si*(P_MAX-P_MIN);
param_fun.u_max_marg=param_fun.u_max-tau;
param_fun.u_n_marg=param_fun.u_n-tau;
```

In this part of the code we initialized the parameters of the centralized MPC that manages the PCM energy. First, the sampling time is assigned to the *time_centr* variable, the prediction model of the centralized controller is extrapolated from the general model found with the *EmbeddedCGModeling.m* function, and the parameters of the optimization problem are defined. Notice that the parameter of the controller will be used into the function called in the Simulink block diagram to find the control decision of the centralized MPC controller. However, in the proposed Simulink scheme we used a simpler ad hoc algorithm that uses the parameters contained into the structure *param_fun* considerably reducing the computational complexity. The structure will be given as input to the function called in the Simulink file to find the control decision of the centralized controller. The last data are used for building the energy bound, *U_N* and *sliding_time* are computed on the basis of the data specified in the first part of the code. These data are finally shifted of a margin to account uncertainties and sample time.

```

% -----------------> Distribuito per Temperatura cores <----------------- %

% Sampling time
time_distr=2.5e-3;

% Thermal MPC parameters
distr.rho=10^5;                                 % slack variable weight
distr.Vy_max=0;                                 % 0=hard 1=soft
distr.ubA=360*ones(1,N_CORE);                   % Maximum temperatures
distr.input_weight=1;                           % R_u


stringa='distr.models.m';
for i=1:N_CORE
    j=0;
    % Building of the single-core prediction model from the centralized
```

```
18      % prediction model
19      eval([stringa num2str(i) '.a= model.a(i,i);'])
20      eval([stringa num2str(i) '.b= [model.b(i,i) model.b(i,end-1) model.a(i,
        1:end-1)];'])
21      eval([stringa num2str(i) '.b(1,i+2)= 0;'])
22      eval([stringa num2str(i) '.c= model.c(i,i);'])
23      eval([stringa num2str(i) '.d= zeros(1,N_CORE+2);'])
24      eval([stringa num2str(i) '= c2d(ss(' strcat(stringa,num2str(i)) '.a, '
        strcat(stringa,num2str(i)) '.b, ' strcat(stringa,num2str(i)) '.c, '
        strcat(stringa,num2str(i)) '.d), time_distr, ''zoh'');']);
25      eval(['distr.init_state_' num2str(i) '=Tenvironment;'])
26
27      % QP problem parameters
28      eval(['distr.Q_qp_' num2str(i) '=[distr.input_weight
        zeros(size(distr.input_weight,1),1); zeros(1,size(distr.input_weight,1))
        distr.rho].*2;']);
29      eval(['distr.f_1_' num2str(i) '=2*distr.input_weight;']);
30      acca=strcat(stringa,num2str(i));
31      eval(['distr.aa_' num2str(i) '=' stringa  num2str(i) '.a;']);
32      eval(['distr.bb_' num2str(i) '=' stringa  num2str(i) '.b(:,1:N_CORE+2);']);
33      eval(['distr.cc_' num2str(i) '=' stringa  num2str(i) '.c;']);
34      eval(['distr.dd_' num2str(i) '=' stringa  num2str(i) '.d(:,1:N_CORE+2);']);
35      eval(['distr.A_qp_' num2str(i) '=[distr.cc_' num2str(i) '*distr.bb_' num2str(i)
        '(:,1) -distr.Vy_max];']);
36      eval(['distr.b_1_' num2str(i) '=-distr.cc_' num2str(i) '*distr.aa_' num2str(i)
        ';']);
37      eval(['distr.b_2_' num2str(i) '=-distr.cc_' num2str(i) '*distr.bb_' num2str(i)
        '(:,2:end);']);
38      eval(['distr.x_obsv_' num2str(i) '=distr.init_state_' num2str(i) ''';']);
39  end
```

The same data defined for the central controller are defined for the distributed one. First, the sampling time of the controller is defined (*time_distr*), then the parameters of each local controller are stored in the *distr* structure: the thermal model of each core (extrapolated from the general model found with the *EmbeddedCGModeling.m*), the optimization problem parameters and the initial state and the gain matrix of the observer. Notice that the observer in this case is not necessary since each model has only one state coincident with the measured output. It is also worth to note that the optimization problem solved by each controller is very simple, therefore, as for the case of the centralized controller, we proposed a simpler ad hoc function that returns at any sampling time the control decision using a simple “if” statement. Using this function the control computational complexity considerably reduces.

C.4.3.2 Simulink block diagram details

In this Section we provide details of the block diagram shown in Fig. C.7.

The *U bound* block The block *U bound* gives as output the maximum energy that the PCM layer can storage at each sampling time in order to ensure the re-sprinting every period. The signal outgoing from this block has the shape of a trapezius repeated every period defined by the user. It has been realized using the standard Simulink block *Repeating Sequence*. The signal is then delayed by one sampling time (*time_centr=10ms*). The delayed value of the energy bound, *U_max*, represents the actual value of the constraints, while the original value is the future value of the bound (*U_max_fut*). This trick is necessary since the centralized controller needs the future value of the constraint in order to regulate the power consumption of the cores for the next interval and prevent the energy bound violation.

The *PCM MPC Layer* block The block *PCM MPC Layer* takes as input the information on the energy bound (*U_max_fut* and *U_max*), the temperature of the cores, of the PCM and of the ambient, and the target power consumption required by the high level manager. It returns as output the target power of the cores (*targ_pow_PCM*) opportunely trimmed to prevent the energy bound violation. This reduction is realized by the MPC used for PCM management. Looking at the Fig. C.7 we notice that the MPC is implemented calling the function *find_pow.m* with the Matlab function block *Centralized*. The code below show the implementation of the function.

```
1  function  [out] = find_pow(Temp, T_pcm, T_amb, P_core_in, u_max, u_max_future,
       state, memory, parameters)
2
3  N_CORE=16;                      % number of cores
4  % Initialization of the variables
5  P_core_out=P_core_in;
6  state_out=state;
7  memory_out=memory;
8  memory_out(2)=P_core_in(2);     % Storing of past power (core 2)
9  % Energy estimation
10 u_estim=parameters.model.a*memory(1)+parameters.model.b*[Temp(1:end-1); T_pcm;
       T_amb];
11 memory_out(1)=u_estim;          % Storing of energy
12
13 % State 1
14 if state(1)==1
15     if (u_estim>=u_max_future)
16         % If violation of energy bound, then computing of ideal target power
17         P_ideal= (((u_max_future-u_max)/ parameters.Ts)- P_core_in(1)+ (T_pcm-T_amb)
       /parameters.R_pcm)/ (N_CORE-1);
18         % If ideal target power lower than P_MIN --> assign P_idle to cores
       2,...,N_CORE
19         if (P_ideal>parameters.P_idle && P_ideal<parameters.P_min)
```

```
            P_ideal=parameters.P_idle;
        end
        if (P_ideal<parameters.P_idle)
            P_ideal=parameters.P_idle;
        end
        % Target power consumption
        P_core_out=[P_core_in(1); P_ideal*ones(parameters.N_CORE-1, 1)];
        % Changing of the state
        state_out(1)=0;
        state_out(2)=1;
    end
end

% State 2
if state(2)==1
    % Computing of ideal target power
    P_ideal= (((u_max_future-u_max)/ parameters.Ts)- P_core_in(1)+ (T_pcm-T_amb)/
    parameters.R_pcm)/ (N_CORE-1);
    % If ideal target power lower than P_MIN --> assign P_idle to cores 2,...,N_CORE
    if (P_ideal>parameters.P_idle && P_ideal<parameters.P_min)
        P_ideal=parameters.P_idle;
    end
    if (P_ideal<parameters.P_idle)
        P_ideal=parameters.P_idle;
    end
    P_core_out= [P_core_in(1); min(P_core_out(2:parameters.N_CORE),
    P_ideal*ones(parameters.N_CORE-1, 1))];
    % If the energy bound minimum value is reached or another sprinting is requested
    if  ((u_max_future>u_max+0.1)||(P_core_in(2)>memory(2)))
        % Target power consumption
        P_core_out=P_core_in;
        % Changing of the state
        state_out(1)=1;
        state_out(2)=0;
    end
end

out=[P_core_out; state_out; memory_out];    % Outputs update

return
```

The function, beside the parameters entering the *PCM MPC Layer* block, takes as input the variable *parameters* that is the structure containing the value defined in Section C.4.3.1. Notice also the parameters *state* and *memory*. These are not really inputs, but variables that should be kept in memory at each sampling time. Thus, we feedback their values at each sampling time adding a one sample delay block. The output of the function is an array containing the power consumption value and the values to be stored (i.e. *state* and *memory*).

In the first part of the code we initialized the output variables, and we computed a prediction of the internal energy of the system (*u_pred*) at the next sampling time. The variable *memory* is a vector with two elements. The first contains the past power consumption of the core number 2, while the second contains the predicted energy. The controller can be implemented as a two states automata. The controller remains in the state 1 if the predicted energy is below the future value of the energy bound. In this case there is no need to trim the target power consumption requested to the core (*targ_pow_PCM=targ_pow*). Otherwise, if there is a energy bound violation, the controller switch to the state 2 where the target power consumption in input is reduced to the value *P_ideal*, that is the value that maintains the energy of the next sampling interval close to the energy bound. This value is assigned to the output variable *targ_pow_PCM*) until the energy bound reach the minimum value or another re-sprint is requested (the input power of the core 2 is greater than the past one).

The *Thermal MPC Layer* block The block *Thermal MPC Layer* takes as input the information on the temperature of the cores and of the PCM, and the target power consumption opportunely updated by the *PCM MPC Layer* block. It returns as outputs the controlled power of the cores (*cntrled_pow*). Fig. C.7 shows that the MPC is implemented calling the known function *distrMPC_Yalmip.m* with the Matlab function block *Distributed*. However, as already mentioned, it is possible to use a simpler function for reducing complexity.

```
function [power_cores_cntrl] = distrMPC_Simple(Temp,T_pcm,power_cores_target,distr)

N_CORE=16;      % number of cores
P_idle=0.05;    % power consumption of a core when it is turned off
P_min=0.15;     % power consumption of a core when the freq is the minimum
for j=1:NC
    % Computing of the future ideal power by inverting the model to obtain T_MAX
    eval(['p_ideal= (distr.ubA(j)-distr.model.m' num2str(j) '.c* distr.model.m'
    num2str(j) '.a* Temp(j)- distr.model.m' num2str(j) '.c* distr.model.m'
    num2str(j) '.b* [0; T_pcm; Temp(1:end-1)])/ (distr.model.m' num2str(j)
    '.c*distr.model.m' num2str(j) '.b(1));']);
    % If the ideal power is lower than the requested --> trim, otherwise do nothing
    if power_cores_target(j) <= p_ideal
        power_cores_cntrl(j)=power_cores_target(j);
    else
        power_cores_cntrl(j)=p_ideal;
    end
end

% if P_idel < ideal power < P_MIN --> ideal power = P_MIN
power_cores_cntrl(power_cores_cntrl<=P_idle)=P_idle;
power_cores_cntrl((power_cores_cntrl<P_min)& (power_cores_cntrl>P_idle))=P_min;
```

```
20
21 return
```

The function takes as inputs the same parameters of the *Thermal MPC Layer* block and the *distr* parameters defined in Section C.4.3.1. The output is the vector containing the controlled power of each local controller and that will feed the cores of the chip. For each core the function computes, by inverting the single-core model, the ideal power (*p_ideal*) needed at the next sampling time to maintain the temperature exactly at the critical value. The controlled power given as output will be the minimum between the *p_ideal* and the target power requested by the *PCM MPC Layer*.

Bibliography

[1] The MathWorks Inc. *MATLAB 7.9.0*, Natick, MA, 2009.

[2] A. Bemporad, *Hybrid Toolbox - User's Guide*, http://cse.lab.imtlucca.it/∼bemporad/hybrid/toolbox, 2004. 265

[3] J. Löfberg, *YALMIP : A Toolbox for Modeling and Optimization in MATLAB*, in Proc. of the CACSD Conference, Taipei, Taiwan, 2004. 262, 277

[4] qpOASES, *Homepage*, http://homes.esat.kuleuven.be/∼optec/software/qpOASES/ 284

[5] H. J. Ferreau, H. G. Bock, M. Diehl, *An online active set strategy to overcome the limitations of explicit MPC*, International Journal of Robust and Nonlinear Control, Vol. 18(8):816-830, 2008. 284