POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

# An autotuning controller for CPU power/performance/thermal management

Tesi di Laurea Magistrale in
Automation and Control Engineering - Ingegneria dell'Automazione

Author: **Riccardo De Rosi**

Student ID: 976820
Advisor: Prof. Alberto Leva
Co-advisors: Federico Terraneo
Academic Year: 2022-23

# Abstract

This thesis is part of a long-term research work that aims to define new control strategies for the combined management of temperature, power, and performance in multicore microprocessors. In this context, the thesis goal is to develop a controller capable of adapting itself automatically to a specific processor in its installation conditions, regardless of the particular architecture of the said processor.

In this work, the focus is in some sense implicitly set on future generations of processors, thinking in particular to three-dimensional architectures as these exhibit higher power density together with the disadvantage, as opposed to their "planar" counterparts, of not having all layers in contact with the heatsink, increasing the risk of thermal runaway. In such a *scenario* – but worth noticing, also for modern planar architectures – developing a processor-integrated controller that can manage power and performance, while ensuring adequate temperature control to improve processor reliability and reduce power consumption, is critical.

At present, the control strategies adopted by the industry on commercial processors tend to separate power/performance management from temperature control, leaving the former to the operating system – by means of components such as the Linux governors – while the latter is left to the processor hardware. This division of roles is, however, becoming questionable owing to the increasing power density and as a consequence of the faster and faster thermal dynamics, and the compound of the above leads to non-optimal control solutions.

The purpose of this thesis is to pursue the attempt to address these problems in a coordinated manner – so that a single controller handles them – and most relevant, to give that controller the ability of adapting to the processor to which it is connected, making its large-scale deployment feasible as no human tuning intervention is required. Thus, the purpose of the proposed control is to manage the temperature of individual cores in a multi-core processor, simultaneously solving the problems of temperature, power, and performance, through the use of control theory and rules suitable for self-calibration based on experiments conducted in an initial control phase (for example, when a system

is booted up).

A validation of the control strategy through simulations is also an integral part of this thesis. Several experiments are proposed, as proof of the choices made, comparing the techniques considered for the autotuning phase in order to (preliminarily) show their advantages and disadvantages. To this end, the proposed autotuning controller was also implemented as C++ code – a nontrivial software design and engineering task – so as to have it ready for deployment on a real processor. Correspondingly, simulation tests were conducted both with simplified models for system-level studies (using the Modelica language) and with an accurate 3D chip thermal simulation (3D-ICE, to which the C++ autotuning controller was coupled).

**Keywords:** autotuner; thermal control; dark silicon; power/performance management; microprocessors.

# Sommario

Questa tesi fa parte di un lavoro di ricerca a lungo termine che mira a definire nuove strategie di controllo per la gestione combinata di temperatura, potenza e prestazioni nei microprocessori multi-core. In questo contesto, l'obiettivo della tesi è sviluppare un controllore in grado di adattarsi automaticamente a uno specifico processore nelle sue condizioni di installazione, indipendentemente dalla sua architettura.

In questo lavoro l'attenzione è in un certo senso implicitamente posta sulle future generazioni di processori, pensando in particolare alle architetture tridimensionali in quanto esse presentano una maggiore densità di potenza unita allo svantaggio, rispetto alle loro controparti "planari", di non avere tutti gli strati a contatto con il dissipatore di calore, il che aumenta il rischio di fuga termica. In tale *scenario* – ma, va osservato, anche per le moderne architetture planari – lo sviluppo di un controllore integrato nel processore che sia in grado di gestire potenza e prestazioni, garantendo anche un adeguato controllo della temperatura per migliorare l'affidabilità del processore e contenere il consumo energetico, è critico.

Attualmente, le strategie di controllo adottate dall'industria nei processori commerciali tendono a separare la gestione di potenza e prestazioni dal controllo della temperatura, lasciando la prima al sistema operativo – tramite componenti come i governor di Linux – mentre la seconda è lasciata all'hardware del processore. Questa divisione dei ruoli sta tuttavia diventando discutibile a causa della crescente densità di potenza e di conseguenza della dinamica termica sempre più rapida; la combinazione di tutti i fattori sin qui evidenziati porta a soluzioni di controllo non ottimali.

Scopo di questa tesi è perseguire il tentativo di affrontare questi problemi in modo coordinato – cosicché siano gestiti da un solo controllore – e, cosa più rilevante, dare a quel controllore la capacità di adattarsi al processore a cui è collegato, rendendo fattibile la sua implementazione su larga scala in quanto non è richiesto alcun intervento di messa a punto da parte di operatori umani. Pertanto, lo scopo del controllo proposto è quello di gestire la temperatura dei singoli core in un processore multi-core, risolvendo contemporaneamente i problemi di temperatura, potenza e prestazioni, attraverso l'uso della teoria

del controllo e regole adatte all'autocalibrazione basate su esperimenti condotti in una prima fase di controllo (ad esempio, all'avvio di un sistema).

Anche una validazione della strategia di controllo attraverso simulazioni è parte integrante di questa tesi. Vengono proposti diversi esperimenti, a supporto delle scelte effettuate, confrontando le tecniche considerate per la fase di autotuning al fine di mostrarne (preliminarmente) vantaggi e svantaggi. A tal fine, il controllore con autotuning proposto è stato anche implementato come codice C++ – un'attività di progettazione e ingegneria del software non banale – in modo da averlo pronto per l'implementazione su un vero processore. Di conseguenza, sono stati condotti test di simulazione sia con modelli semplificati per studi a livello di sistema (utilizzando il linguaggio Modelica) sia con un'accurata simulazione termica tridimensionale del chip (usando lo strumento3D-ICE, cui l'autotuner realizzato in C++ è stato accoppiato).

**Parole chiave:** autotuning; controllo termico; dark silicon; gestione potenza/prestazioni; microprocessori;

# Contents

# Introduction, Motivation and Contribution

This thesis is about the automatic tuning of power/performance/thermal controllers for modern microprocessors — a matter that emerged in quite recent years but is gaining so much importance to rank among the enablers for high-performance computing solutions, and often to be vital for the safe operation of the processors themselves.

Since their invention, microprocessors – hereafter $\mu$Ps for short – have always shown an increase in performance from one generation to the following one, and the rate of that performance is nowadays impressive. To give just a few numbers, the first processor ever invented – the Intel 4004, introduced in 1971 – had a 4-bit technology, a single core, 2300 transistors, a clock frequency of 740kHz and thus a capability of 60000 operations per second, while dissipating "only" 0.5W [8]. Today, a consumer $\mu$P such as the AMD Ryzen 7 5800H [2] has eight 64-bit cores, over 10 billion transistors, and a clock frequency of 3.2GHz, which makes it capable of 60 *million* operations per second and results in a Thermal Design Power (TDP) of 45W (not a stunning figure, incidentally, as other devices break the 100W barrier). Figure 1, taken from [8], illustrates the current trends in the evolution of $\mu$Ps.

Such exponential growth was made possible by the continuous progress of manufacturing, and the first to notice this trend was Gordon Moore, co-founder of Intel, who in 1965 as-

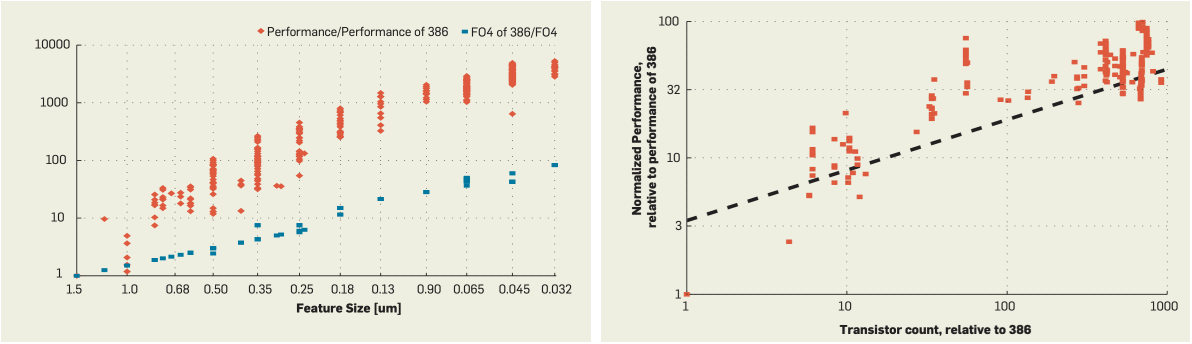



Figure 1: trends in the evolution of $\mu$Ps.

sumed that the number of processors would double every year. This observation was later revised to become the famous "Moore's law", which states that the number of transistors in $\mu$Ps doubles every 18 months and was taken as a yardstick and a target for the various manufacturers in the industry.

For some time, manufacturers could comply with this law by just shrinking the size of transistors (in the technical jargon, the "lithography") in each subsequent generation. This was made possible by another great intuition, known as the "Dennard's scalability", according to which it is possible to keep the base size and the power of a $\mu$P unchanged, from one generation to another, by adopting a shrinkage factor of $1/\sqrt{2}$, under the assumption that the number of transistors is doubled and the clock frequency is increased by 40%.

However, the said shrinkage was subject to physical limits, beyond which parasitic effects due to quantum physics would become relevant in the electronic circuits and deteriorate the performance. Consequently, as it was no longer possible to go deep, the only possible movement remaining horizontal, manufacturers moved from structures with only one computing unit, or "single core", to solutions with multiple computing elements called "multicore". This revolution reverberated throughout the computer world, opening the door to the possibility of executing code in parallel within a single machine, which previously was only possible via a network of computers.

The multicore era began with the first non-embedded multicore processor introduced in 2001 by IBM, under the name POWER4, and containing two cores. Since then, the number of cores continued to grow and, for example, today it is possible to find common commercial solutions with 6 or 8 cores, or industrial server solutions with 128 cores. Thanks to the advent of multicore, Moore's law could be extended up to the present day, and thanks to technological advances, clock frequencies could also increase year after year, guaranteeing ever higher computing speeds.

Nevertheless, the demise of Dennard's scalability, coupled with the increase in frequency but also the relevance of transistor parasitic effects, brought about the new problem of heat generation. This matter crept in slowly and was initially underestimated, but grew in relevance year by year to such an extent that it now threatens the future of multicore technology, and thus of the Moore's law in general. In fact, since power dissipation is tied to the efficiency of transistors and their switching frequency, things can only get worse if the number of transistors continues to increase.

As such, over the years, $\mu$P heat dissipation devices connected – once non-existent – began to grow in volume and power, also evolving to cope with the ever-increasing heat generated. Indeed, the power density of $\mu$Ps has already reached such levels that, considering

the technological limits of dissipation, it is impossible to operate all the elements of a $\mu$P at maximum power at the same time.

The world of processors has therefore entered the so-called "dark silicon era", in which the thermal power generated is such that certain areas of the active silicon, i.e. the portion of the silicon capable of performing operations, must remain unused to prevent the processor from overheating. In fact, the power is such that the processor would be destroyed if it were operated at full load without limitations. And to make the picture even more complicated, the heat generated is subject to enormous, sudden, and hardly predictable changes. The resulting challenge is so tough that some researchers even cast doubt on a prosperous future – if not on mere survival – for the multicore era [9].

When dissipation systems cannot handle the needed powers, the $\mu$P voltage, and frequency must be reduced in order to decrease the generated heat. Voltage has a quadratic relationship with the said heat, hence in modern electronics, it is often already minimised. On the other hand, frequency has a linear relationship with heat but allows more effective action by having larger operating ranges.

Obviously, however, limiting the clock frequency leads to a reduction in the $\mu$P performance. Thus, the problem of dark silicon introduced the question of the relationship between $\mu$P performance and temperature: said briefly, while thermal management is crucial, it should not be extremely limiting.

Historically, processor temperature management, or more generally thermal management, has always been split between the software and hardware side. While the former only lowered the frequency when the temperature exceeded particularly high thresholds, the latter merely cut the processor when safety limits were reached. This partitioning of roles was – and still is – inefficient, leading to sub-optimal solutions.

The relevance of this problem is reflected in the literature, where many solutions have been proposed addressing different aspects of the problem, from more "static" solutions such as "thermal-aware" processor design [11], to better management of the source of the problem such as "computational sprinting", which concentrates the most onerous operations in the initial instants of the execution of a task, to mitigate the effect on temperature by exploiting the transient [20].

However, design can only partially solve the problem, and optimised code scheduling may bring more harm than good. Hence, although such solutions may be valid and contribute positively to the thermal management of $\mu$Ps, they certainly cannot completely solve the problem and real-time temperature management systems should be introduced, alongside

them, in order to cover those scenarios where their effectiveness vanishes.

This is why in modern $\mu$Ps power, performance and temperature need managing by suitably designed feedback controls, to ensure effective and safe operation in the face of (i) uncertainties in the device installation conditions and (ii) completely unpredictable disturbances as the necessities of software threads spawned at virtually any time in a multitasking environment. This is therefore why the research to which this thesis belongs is necessary.

## Problem Statement

As nowadays the thermal power generated by active silicon is such that not all computing elements can operate at the same time, one of the greatest goals to be achieved is to minimise the area of dark silicon. Also, in a context where dissipation systems are not able to respect the TDP constraints, meaning that they overheat when the power exceeds the limit of the heat sink, it is necessary to integrate a solution that can effectively manage these transients, to make the system more reliable in the long term.

The TDP is a *de facto* standard design concept in the industry and represents a total power budget for the chip, such that cooling must respect it to ensure full system operability within safety limits. If the budget is not respected, the $\mu$P heats up more than the system is able to cool, leading to temperatures rising toward their safety limits.

Looking at a $\mu$P in more detail, it can be seen that only a small part of the silicon, called "active silicon", contributes to the generation of heat, while the bulk – i.e. the remaining part – only contributes to the thermal transmission of heat to the heat sink. The result is that heat passes through three distinct regions with different heat capacities and thermal resistances. In particular, the capacity of the active silicon is much smaller than that of the bulk, and it too is smaller than that of the heat sink. Consequently, the thermal dynamics relative to the processor has three main dynamics, starting with that of the active silicon, which is the fastest, to that of the heatsink, which is the slowest. However, the continuous downsising of the transistors due to the evolution of technology has greatly changed the relationship of the various dynamics, rendering once effective temperature management techniques nowadays inapplicable, so much so that today it is no longer possible to manage temperature transients by increasing the speed of the fan connected to the processor.

Looking into the past, it is possible to identify three different eras in the area of temperature management. Initially, processors dissipated so little power, and their area was so large, that not even the use of a fan for cooling was necessary. As a result, there was

no need to control the temperature, and performance-power management was reduced to just freezing the processor if there were no operations to be performed.

Subsequently, as computing power increased, processors began to require dissipation systems, thus defining the dawn of the new era. Although the size and power were such that all cores were fully operable, the problem of temperature management appeared, assuming a dignity of its own alongside performance management. Overheating was managed with the help of fans, due to the fact that thermal dynamics were very slow, while performance management was attributed to the operating system, where components ("governors") were introduced that acted on the DVFS depending on the perceived load. This period in history was the beginning of research in this area and the first papers on the subject appeared, although coming from computer engineering than from the control community.

With the advent of the dark silicon era, a gap appeared between the power output and the power input: the heat is so high that the active silicon part immediately overheats and the cooling system cannot keep up with the changes. For this reason, temperature control can no longer take place independently of performance/power management but must act on heat generation, reducing it temporarily, so that the heatsink can operate correctly and adapt.

As mentioned above, modern $\mu$Ps allow the voltage and/or frequency to be adjusted by means of *dynamics voltage and frequency scaling* (or DVFS) technology. The DVFS modulation technique is possible thanks to a controller allocated in the physical layer of the system and capable of managing the voltage supply and clock frequency. Although the voltage has a quadratic relationship with the power dissipation, it is constrained by the minimum threshold voltage for transistor switching. In fact, the transistor base voltage must be high enough to ensure proper switching, and the minimum threshold value increases with the switching speed of the transistors. Furthermore, intrinsic current losses only scale linearly with voltage, so that even though the heat emitted for switching decreases, the relative contribution of losses increases, making it even more complex to manage. Consequently, the only viable route is frequency modulation, assuming that the voltage is always kept at the lower limits to ensure minimum energy consumption at all times and that it follows the frequency demand when this changes.

At present, however, a divergence can be observed between the literature and the industry: although the former has analysed the problem by proposing increasingly complex techniques such as linear-quadratic controllers, model predictive control [13], convex optimisation, and so on, these solutions are tested on specific simulators and can hardly be ported to real hardware given the limitations of sensors and actuators. For example,

commercially available (and affordable) temperature sensors have a typical resolution of one degree and even higher noise. In addition, this kind of control is generally computationally heavy, making it ineffective in the context where the control needs to act in a very short time.

As a result, the industry is handling the problem differently, having left performance management to software and having assigned temperature control to simple controllers embedded in hardware. These controllers are designed to have fixed and fast timings, can take control over the DVFS, and act, for example, on the frequency by clock gating. The reasons for this are due to the need to ensure thermal *safety* rather than "control" in the strict sense of the term.

To give a more complete view, it must be said that there are proposals for handling the problem that do not rely on DVFS. Among the most important of these is certainly task migration, according to which the operating system should allocate new tasks to the colder cores, giving the hotter ones a chance to cool down. However, this proposal has a very variable impact and depends on many factors such as memory or cache management availability. Mostly, task migration involves very important elements of the operating system such as the task scheduler, which are already subject to several constraints that must be met. Consequently, such a solution may be difficult to integrate due to the increased maintenance costs – and difficulties at large – that the resulting, increasingly complex system would incur.

Finally, it is important to note that performance management, temperature control, and task scheduling are generally different disciplines, managed by different developers. For this reason, a solution that maintains this division while remaining as transparent as possible, and affecting other aspects of the system as little as possible, is certainly the way to go. This is the attitude taken in the research to which this thesis belongs.

## Motivations and contributions of the proposed solution

The thermal dynamics of active silicon is already incredibly fast, and today requires a control capable of reacting at a millisecond scale. Moreover, the situation will most likely worsen with the advent of three-dimensional architectures. For this reason, the use of fixed rate controls is no longer a viable option. Such fast, periodically computed controls require a non-negligible amount of computational power to be allocated, and it is not clear where to best integrate them: if they were allocated in software, they would burden the operating system by stealing computational power. If they were implemented in hardware, they would require silicon area and power. In contrast, event-based controls

have the characteristic of acting only when needed, mitigating the overall computation and power demand.

Also, the thermal dynamics of a single core is so rapid that, on the time horizon of interest for thermal control, the interaction with the other cores is negligible. This allows for the use of decentralised control techniques, which offer the great advantage that each element is controlled independently of the others, representing the interactions as disturbances; thus, the computational load is greatly lightened compared to centralised solutions. This choice is also the one adopted today by manufacturers in processors that allow DVFS modulation on individual cores.

Finally, nowadays, the same operating system may be found on a myriad of different products, which may differ in processor and cooling systems, and individual components may also differ in the same product family due to manufacturing variability. Therefore, the control should be able to adapt to ensure correct operation on the different systems, also removing the burden from manufacturers of having to design *ad hoc* solutions for their products. Ideally, next-generation controls should be able to perform self-calibrate, and which is more challenging, to do in a context where the inactivity of the other cores cannot be guaranteed.

As proven in the previous work [16], event-based control is able to effectively manage temperature transients in a multicore system, also demonstrating the viability of using decentralised control systems. The objective of this thesis is the development of a controller with a methodological approach to temperature management of individual cores in a multicore context, and more specifically, to endow such a controller with autotuning capabilities. A controller is proposed that is able to calibrate itself autonomously by performing an experiment on the controlled core, in such a way as to be resilient to disturbances from other cores and the environment, and rapidly enough to allow for periodic re-calibration, for example at boot time.

## Thesis organisation

The thesis is organised as follows:

- Chapter 1 offers an overview of the most recent (and worthy of mention) approaches to account for the problem faced in this thesis.
- Chapter 2 is dedicated to the theory in support of our proposal. It begins with an introduction to the concept of the describing function, how it is a powerful and effective tool for studying permanent oscillations of a nonlinear system, and

how it can be used to identify the properties of a dynamic system. Then, the theory supporting the technique of permanent excitation used for the self-calibration experiment is also discussed, introducing the describing function of the nonlinear system and the conditions for the existence of limit cycles. The chapter concludes by discussing the theory in support of the chosen self-tuning rules.

- Chapter 3 overviews the physics behind the process we aim to control, describing the thermal propagation phenomenon and how it can be modelled to obtain representations adequate for control purposes and simulations and the tools used to simulate the system according to the introduced models.
- Chapter 4 discusses the rationale behind the proposed autotuner, how it is composed, what techniques are used and why they were chosen. In particular, the autotuning regulator, the high and low levels of control, the management of operations, and the component of fundamental analysis for self-calibration are described.
- Chapter 5 is dedicated to the implementation process of the autotuner, as how it was integrated into Modelica, and the development of the corresponding library in the C language. The organisation of the Modelica components and their usage, the development of the library, how it was integrated into Modelica and 3D-ICE, and the development of the 3D-ICE client to test the controller on a more accurate simulator will be primarily described.
- Chapter **??** offers a series of experiments in both the Modelica and 3d-ice environments to demonstrate the effectiveness of this approach, with a greater emphasis on the resilience of the method to external disturbances and the self-regulating capabilities of the controller.

# 1 | Related Work

The gradual increase in microprocessor power density has led to the problem of dark silicon, such that it is no longer possible to operate all the computing elements of a processor without making it overheat to the point of damage. The result is a depletion of processor performance that can no longer support its own thermal heat production. As a result of its impact on performance, the problem has gained prominence in the literature, which has begun to grow owing to numerous works that aim to alleviate the negative effects as much as possible. Essentially, dark silicon has introduced two critical issues: i) microprocessors can exceed safe thermal limits so rapidly that the dissipation system cannot keep the pace and ii), in high-stress situations, to avoid overheating, it can be necessary to shut down certain areas of the processor. The literature, consequently, tries to propose solutions that can mitigate one or the other aspect, seeking optimalities that will maximise performance in this intricate context.

Putting the focus on the heat generation aspect of the processor, it is sensible to say that it is mainly related proportionally to the square of the voltage applied to the microprocessor and to the clock frequency. As introduced in the previous chapter, although the effects of a change to the voltage impact heat generation the most, the voltage is subject to minimum constraints that serve to ensure the proper operability of the processor. In fact, given a switching frequency, the voltage must be above a minimum value (proportional to the frequency) to ensure that transistors switch from a state of complete isolation to a fully conductive state, avoiding the resistive zone inherent in them. Since the voltage only acts on the generated power, with no further negative effects, there is no reason not to keep it at its minimum at all times, rather following the variations on the frequency command. In contrast, the switching frequency depends on both the utilisation factor of the individual core and the maximum achievable frequency. The former is related to the computational load required to the processor, while the latter is managed at the hardware level through the DVFS. The remaining contributions to the generated power, on the other hand, depend on the structure and design of the processor, which, of course, can only be credited at the design stage and, for that reason, can mitigate but not solve the problem in the transients of interest in the dark silicon case.

As a result, along with proposals for different processor designs that also consider the heat transfer occurring within the processor, almost all of the proposals resort to acting on the switching frequency through software, hardware, or both.

Among the various solutions that do not involve the use of DVFS, the most important is certainly the use of task migration [17] [18] [6] [10]. It consists of moving a task from a hot core to a cold core in such a way as to allow the hot core to cool down. This is possible by making the scheduler aware of the temperature of individual cores. However, this kind of operation is incredibly delicate because it requires changing the behaviour of the task scheduler, which is a very critical component of the operating system, since it must ensure the real-time execution of various operations. In addition, task migration takes a time that varies greatly depending on many factors among which cache and memory availability, and, in the absence of a proper architectural environment, could have detrimental effects.

A noteworthy citation is the recent work of Mohammed et al. [17]. Their proposal combines an advanced task migration method and the use of clock gating through DVFS. Specifically, cores are organised into clusters, whose components share the LLC (*last-level cache*). Migration occurs only from a hot core to an idle core, so overhead can be minimised because a unidirectional transfer is performed: the task passes from the hot core to the idle core, there is no task swapping between active cores. In addition, migration occurs between cores in a power-saving state that still maintains information in L1 or L2 caches. In this way, the previously-idle core can draw on the information inherent in the task just received directly from the hot core, reducing requests to the common LLC cache. In addition, migration occurs only between elements in the same cluster, allowing requests to memory due to cache misses to be minimised. These choices allow for minimising the "slowest" operations concerning memory access. In fact, accessing the first- or second-level cache of a processor is faster than accessing the LLC, and it is faster to access the LLC than RAM.

If there are no cold cores to use, DVFS is called as a last resort to decrease the temperature. However, the DVFS is called to iteratively decrease the frequency at each step until it returns to the "regime" values.

In any case, such a solution assumes that the active computational power is always less than 50%, resulting in a big waste of resources. Moreover, the frequency management is crude (to say the least).

On the other hand, it is possible to limit the maximum frequency of operations through the use of DVFS. In this case, the core is under full (computing)load but the maximum frequency at which it performs operations is reduced, and thus the power generated is

also reduced. However, the effect of such a control is time dilation: the same number of operations is performed at a lower rate, resulting in a longer execution time. In any case, control via DVFS is very effective and simple to implement, which is why to date it is the (only) solution used in the industry.

Over time, several proposals more tightly related to the explicit use of control were introduced. The advantage of true control lies in its ability to deal promptly with disturbances (a problem that also in this thesis has dictated several choices, as will be shown). Thus increasingly complex controls began to appear, such as linear-quadratic controls and predictive controls [25] [4].

In particular, the recent proposal by Wang et al. [25] consists of a hierarchical predictive control that contains both task migration and thermal management via DVFS. The model considered for predictive control is a linear system with $n$ states, that are the temperatures of the several regions of the core obtained through space discretisation, and such that the number of cores $l < n$. The boundary temperatures of individual cores are then defined, and the predictive control operates to maintain the temperatures below said limits. At each iteration, the predictive control calculates the power that allows the individual core to follow its temperature reference. This power reference is then compared with the powers currently generated by the individual cores. If there are powers that exceed the calculated value, task migration is performed. Since task migration among all the cores would be too onerous because it requires time $O(n^3)$) for its complete resolution, the cores are divided into clusters and migration is performed at two levels: migration between clusters and migration within the cluster. If cores with higher power than suggested remain, DVFS intervenes by reducing the frequency.

However, as they state, the control takes an average of 0.01s per iteration to resolve for a 100-core processor, casting doubt on its actual effectiveness. In addition, it is unclear how the information needed for modelling the thermal system can be derived and how this solution can be portable.

However, these kinds of proposals lack tests on the actual physical system or at least on some accurate simulation platform, rightly so given their complexity, but they also lack solutions for portability, which is very important if this particular proposal is to be used on a large scale. In fact, these solutions require a thermal model of the system, and the experiments required for modelling are not trivial and require the input of an experienced user.

Consequently, in this thesis, we decided to use a different approach, relying on event-based control for thermal management and a simple self-tuning technique (in this case, not event-

based) that make the controller capable of withstanding the disturbances present in the particular system at hand.

Event-based control is a particular control methodology that seeks to minimise communication between controller, actuators, and sensors. The sampling time (or even better, inter-action) is no longer constant and synchronised but is governed by an event generator. When properly calibrated, event-based control can perform almost identically to traditional control in the case of disturbance rejection, with the advantage of interposing control only when strictly necessary.

In particular, in this thesis, we rely on the results obtained by Leva et al. [16]. In the quoted work the event generator is a state machine implemented at the hardware level. Specifically, it fires an interrupt request if the temperature is close to the threshold limit, its variation is above a certain threshold, or a certain amount of time has passed since the last control computation. In contrast, the control was implemented software-side, and a PI was chosen. The motivation was to provide as much flexibility as possible to their solution, so that if they wanted to change the control technique, it would be easily replaceable. In addition, moving the control to the software side allows them to minimise the silicon area required for control.

In the paper just cited, a stability analysis was done that provided conditions for the PI controller parameters for the event-based control to be stable. The result is a control with an overhead of only 16 ns, bringing it among the fastest proposals.

Regarding control calibration or tuning, a peculiarity of this research is that tuning must be done in the presence of ubiquitous thermal disturbances from adjacent cores. In the literature, it is possible to find several proposals capable of rejection of slowly varying disturbances, the following article [14] provides an overview. However, the robust relay structure proposed by da Silva et al. was chosen as the exciter. [7]. The method achieves the same results with filters requiring less computational effort and is described in more detail in the associated section 2.4.

As calibration rules, several approaches have been proposed, including the method of contextual control calibration with internal model [15]. This consists of a repurposing of the classic technique useful in contexts where it is used to calibrate a controller after performing an exploratory experiment. In fact, it proposes to combine controller calibration and model identification, thus better results can be obtained by reducing the “waste” of information. This method is seen in more detail in the dedicated theory chapter.

# 2 | Theoretical background

## 2.1. Foreword

In many control applications, there is the need to control a system whose model is (partially) unknown due to several reasons such as an excessive modelling complexity for the purpose to attain, or even just the limited time available to set up the required control.

In such cases, a technique able to identify the control-relevant features of the plant effortlessly and rapidly, and consequently of synthesising a suitable controller, is the preferred way to go. A controller that is able to tune itself after a tuning request, according to a technique of the kind just envisaged, is called an *autotuning* controller.

In the literature, many solutions to the autotuning problem were proposed: some of these – the most relevant for us, and the only ones to which we refer in this work – are based on the execution of an experiment on the controlled process, to obtain information about its dynamics. In turn, and quite expectedly, a number of experiment types can be found in the vast autotuning literature [19]. For the purpose of this work, we limit the focus to two of these kinds, namely open-loop step tests and closed-loop relay feedback tests.

In principle, an open-loop step response record could lead to a complete knowledge of the system to control. In practice, however, this would require that the system be initially in an unperturbed condition, as otherwise the residual free motion would be interpreted – no matter how data is analysed – as an effect of the applied step, leading to erratic results. Relay feedback offers somehow symmetric possibilities: the yielded process information is "local in frequency", consisting in general of points of its frequency response, but as these are collected only when the induced oscillation is "permanent", a non-equilibrium initial condition has hardly any influence on the correctness of the results.

These simple considerations motivate the success of "relay autotuning", as witnessed e.g. by the comprehensive work [26]. The driving theory of the so-called "permanent oscillation" approach is the describing function method, introduced by Nikolay Mitrofanovich Krylov and Nikolay Bogoliubov in the 1930s, which will is described shortly. The ap-

proach just mentioned is selected in this thesis, because when tuning the controller for one core it is practically impossible to "freeze" all the previous software activity so as to have it consume a constant power and thus reach a thermal equilibrium.

Among the numerous relay-based techniques available, in addition, the selection is centred on those that are able to gather reliable frequency response information while at the same time rejecting static or slowly variable disturbances. Such a feature is fundamental in our *scenario* because any tuning experiment on one core must be handled while the other cores are working, hence producing thermal disturbances, however at a time scale that is "slow" with respect to the intra-core dynamics that the autotuning controller has to address.

In this chapter we set forth and motivate the methodological ideas and entities behind the proposed autotuner, thereby paving the way to the description of its design (Chapter 4) and realisation (Chapter 5).

## 2.2. Permanent Oscillations

We start from the stimulus applied to the dynamics under control, that as anticipated comes from a relay aimed to induce a permanent oscillation condition, as this is the preferred way to go when it is virtually impossible to ensure that prior to the stimulus the said dynamics was at rest.

Given a hard nonlinearity, which is a static nonlinear function such as relay, saturation, or dead zone, it is possible to form a loop with it and a linear system, so that the so-obtained compound system is driven into a permanent oscillation.

In a purely linear loop this would not be possible without leading the system to the stability limit, a condition that is actually not robust to parametric variations: changing the parameters of the linear part of the system could move its eigenvalues away from the imaginary axis and, subsequently, the oscillations would either diverge or be lost.

The analysis of permanent oscillations allows the properties identification of the related linear system, which is the aim of the relay feedback experiment. However, the computations for the existence and shape of permanent oscillations compatible with the linear system are a complex task, due to the nonlinear nature of the overall problem. Such complexity has given rise to heuristic solutions to estimate the required characteristics, a prominent approach being the describing function.

It is important to notice that as the describing function is based on heuristics, the yielded

results are not fully theoretically supported. In fact, there could be *scenarii* in which the describing function leads to erratic results. However, such results are rare and the describing function proves its validity most of the time. This is the reason why the great majority of relay-based autotuning is based on the describing function method.

## 2.3. The Describing Function method

Let us consider a nonlinear element, defined by a static hard nonlinearity with odd symmetry, fed with a sinusoidal input

$$\varepsilon(t) = E\cos(\omega t) \tag{2.1}$$

defined by the couple of real numbers $(E, \omega)$, and let us consider the periodic output with a corresponding period equal to $T = 2\pi/\omega$. if the nonlinear element can be represented in the form $\xi(t) = \varphi(\varepsilon(t))$, where $\varepsilon$ is the input and $\xi$ is the output of the hard nonlinearity, then the periodic output exists, unique and it is equal to $\xi(t) = \varphi(E\cos(\omega t))$. Otherwise, if multiple solutions could exist, still let us consider only one periodic solution exists, at least for some value of E. Under the assumption that the output of the hard nonlinearity admits a Fourier series, defining $\{\Xi_n\}$, being twice the spectrum of $\xi(t)$, it is possible to write:

$$\xi(t) = \sum_{\substack{n=1 \\ \text{n odd}}}^{+\infty} |\Xi_n(E)|\cos(n\omega t + \arg\Xi_n(E))$$

since $\xi(t + T/2) = -\xi(t)$ because the hard nonlinearity is odd and the input is in the form of (2.1). Furthermore, this relation shows that the Fourier coefficients depend only on the input's amplitude and not on the natural frequency $\omega$.

Given the previous results, the describing function is defined as

$$D(E) = \frac{\Xi_1(E)}{E} = \frac{|\Xi_1(E)|}{E} e^{j\arg\Xi_1(E)} \tag{2.2}$$

The aforementioned equation allows us to compute the describing function of any hard nonlinearity, however, it is not necessary to compute it every time since there exist tables that define the describing functions of the most common hard nonlinearities.

The concept of describing function can be extended by introducing a parallelism between the transfer function of a linear system and the describing function of a nonlinear one: as the transfer function represents the relationship between the input and output of a linear system, the describing function represents the relationship between the oscillating input

and the oscillating output of a nonlinear system. Such characteristics can be exploited, under certain assumptions, to infer some information regarding the attached system as described by the following method.

Let us consider a feedback system composed of a hard nonlinearity (named N) cascaded with a SISO system defined using its transfer function $\Gamma(s)$. Furthermore, let us consider the system input $\xi$ to be a periodic motion with angular frequency $\omega > 0$, or equivalently with period $T = 2\pi/\omega$, whose Fourier series is:

$$\xi(t) = \sum_{\substack{n=1 \\ \text{n odd}}}^{+\infty} |\Xi_n| \cos\left(n\omega t + \arg \Xi_n\right)$$

Assuming $\Gamma(s)$ doesn't have any null real part pole, the Fourier series of the periodic motion of the output $\chi$ is:

$$\chi(t) = \sum_{\substack{n=1 \\ \text{n odd}}}^{+\infty} |\Gamma(jn\omega)\, \Xi_n| \cos\left(n\omega t + \arg \Gamma(jn\omega) + \arg \Xi_n\right)$$

If the following equation holds

$$|\Gamma(jn\omega)\, \Xi_n| \ll |\Gamma(jn\omega)\, \Xi_1| \quad , \quad \text{n} = 3,\, 5,\, ... \tag{2.3}$$

one can say that the output $\chi$ is almost composed only of its first harmonic.

The equation (2.3) is also known as the "low-pass assumption" and it constitutes the foundation of the describing function method for the existence assessment and parameters computation of permanent oscillations in the feedback system.

Now, let us consider the ideal case where equation (2.3) holds and

$$\Gamma(jn\omega)\, \Xi_n \;=\; 0 \quad , \quad \text{n} = 3,\, 5,\, ...$$

the following equation is obtained

$$\chi(t) = |\Gamma(j\omega)\, \Xi_1| \cos\left(\omega t + \arg \Gamma(j\omega) + \arg \Xi_1\right)$$

followed by

$$\xi(t) = -\chi(t) = |\Gamma(j\omega)\Xi_1| \cos\left(\omega t + \arg\Gamma(j\omega) + \arg\Xi_1 + \pi\right)$$

Since it is possible to move the time origin without loose of generality, it could be set such that $\arg\Xi_1$ satisfy the equation

$$\arg\Gamma(j\omega) + \arg\Xi_1 + \pi = 0 \tag{2.4}$$

obtaining that

$$\varepsilon(t) = E\cos(\omega t) \quad , \quad E > 0$$

Therefore, the low-pass assumption implies that the input of the hard nonlinearity is a pure sinusoid, allowing to represent N with its describing function, at least to compute the first harmonic of $\xi(t)$. Proceeding with the analysis, for the oscillation to be compatible with the feedback system, it must match the conditions of all the elements involved and, subsequently, going backwards, the following equation is obtained

$$\begin{aligned}\xi(t) = |\Gamma(j\omega)\,\Xi_1| \cos\left(\omega t + \arg\Gamma(j\omega) + \arg\Xi_1 + \pi\right) = \\ |\Gamma(j\omega)\,D(E)| \cos\left(\omega t + \arg\,\Gamma(j\omega) + \arg D(E) + \pi\right)\end{aligned}$$

Knowing that (2.4) can be rewritten as

$$\arg\Gamma(j\omega) \,+\, \arg D(E) \,+\, \pi \,=\, 0$$

The following relation is obtained

$$|\Gamma(j\omega)\,D(E)| \,=\, 1$$

or, equivalently

$$1 \,+\, \Gamma(j\omega)\,D(E) \,=\, 0$$

In conclusion, the obtained results can be reformulated as a proposition.

**Proposition 2.1.** *If the transfer function* $\Gamma(s)$ *doesn't have imaginary poles, assuming the low-pass assumption* (2.3) *holds, the feedback system admits an oscillation in the form*

$$\varepsilon(t) \,=\, \bar{E}\,\cos(\bar{\omega}t) \tag{2.5}$$

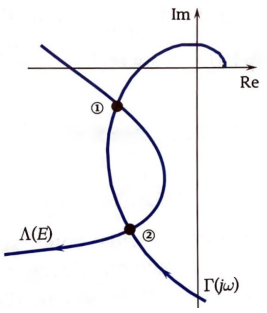



Figure 2.1: Graphical interpretation of the describing function method with two possible solutions. The figure is taken from [5].

*if the pair* $(\bar{E}, \bar{\omega})$ *is a solution of the equation*

$$1 + \Gamma(j\omega)\, D(E) = 0 \tag{2.6}$$

Proposition 2.1 can be seen as a kind of necessary condition for the existence of permanent oscillations in the feedback system, based – we stress – on a heuristic assumption. Additionally, the "much less than" operator allows a fairly free interpretation of the low pass assumption, offering a trade-off between accuracy and applicability. As a final consideration about equation (2.6), we observe that it is a vector equation that represents a system of two equations in two real variables. Since the equations are nonlinear, it could be difficult to set up closed-form conditions for the existence of the solution and, for this reason, numerical techniques are often applied.

The potential of the describing function method can be better appreciated by rearranging equation (2.6): defining

$$\Lambda(E) = -\frac{1}{D(E)}$$

and rearranging (2.6) the following equation is obtained:

$$\Gamma(j\omega) = \Lambda(E) \tag{2.7}$$

This equation allows for a graphical representation of the describing function method: a permanent oscillation of the feedback system exists only if the polar plot of the transfer function of the linear system, parametrised in $\omega$, and the complex plot of the describing function of the nonlinear element, parametrised in $E$, intersect one another. Additionally, the oscillation will be in the form $\varepsilon(t) = \bar{E}\cos(\bar{\omega}t)$ where $(\bar{E}\,, \bar{\omega})$ is an intersection point.

Lastly, not all the solutions of equation (2.6) result in a stable permanent oscillation. In

fact, some solutions could be unstable and a small perturbation could diverge the system into another state. Therefore, a definition for the asymptotic stability of permanent oscillations and a tool to address the problem are introduced.

To briefly explain, let us consider a permanent solution of the feedback system: it will be asymptotically stable if the motion obtained by a "small perturbation" slightly differs from the nominal one, tending to it asymptotically, eventually except for a phase shift. It is important to notice that the concept of asymptotic stability is slightly different from the asymptotic stability of equilibrium. Here, a phase shift between the nominal motion and the perturbed one is admitted, allowing the two motions to have the same shape even though they assume the same value at different time instants. In other words, the definition requires that all the trajectories of the system tend to a closed orbit, which is a limit cycle. Now let us define two vectors applied at the intersection point $(\bar{E}\,,\,\bar{\omega})$: vector $\vec{t}$ is the tangent vector of the plot of $\Lambda$ pointing towards increasing values of $E$ and $\vec{n}$ is the vector orthogonal to the plot of $\Gamma$ pointing to its right following increasing value of $\omega$. Then the following result holds.

**Proposition 2.2.** *The permanent oscillation in the form of equation* (2.5) *and represented by the pair* $(\bar{E}\,,\,\bar{\omega})$ *is asymptotically stable if and only if the scalar product between the vectors* $\vec{t}$ *and* $\vec{n}$ *is negative:*

$$\vec{t} \times \vec{n} < 0$$

Therefore, the asymptotic stability of the permanent oscillation depends on the angle of the two vectors, if the angle is bigger than 90° the oscillation is asymptotically stable, otherwise it isn't.

Equation (2.7) is the core of the describing function method and it is used during the relay experiment by the controller to extract the information about the system required to tune itself. As the equation shows, the obtained information is the complex point of the system frequency representation evaluated at the frequency of the permanent oscillation and computed solving $\Lambda(E)$ with the obtained oscillation amplitude $E$. As a final consideration, the method is justified heuristically and it is based on the assumption called the "low pass assumption" which means that only the fundamental frequency of the oscillation is accounted for. However, it is important to say that the results of the describing function method are consistent in practice and the computed results match the real ones with a good approximation. In conclusion, the describing function method represents an extremely powerful tool to both design the relay feedback system and to infer the system with good accuracy and low effort and it is the reason why it is widely used and one of the few widely applicable methods in the analysis and design of permanent oscillations of

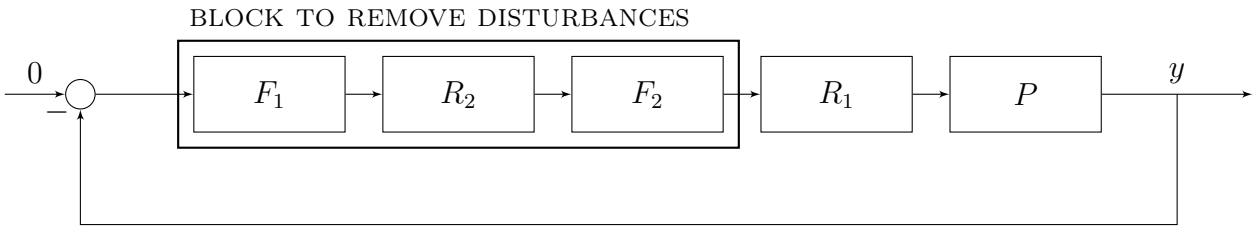



Figure 2.2: Robust relay feedback structure

nonlinear systems.

## 2.4. Robust Relay Feedback Structure

In general, disturbances greatly deteriorate the performance of relay-based identification procedures. While high-frequency noise makes it harder to correctly measure the amplitude of the oscillation, and can even provoke spurious relay toggles if not chattering, static and slowly variable disturbances deteriorate the experiment in that they make it hard to recognise a permanent oscillation; if large enough, such disturbances can even prevent relay control from operating correctly, getting the system to remain stuck in either of the two relay conditions.

Therefore, given the physical setting we address, a relay feedback structure (RFS in short) capable to reject slowly varying disturbances is mandatory in our control problem. To fulfil this need, we resort to the robust structure proposed by da Silva et al. [7] and shown in figure 2.2. The said structure is composed of a relay with a high- and a low-pass filter. The high-pass filter is an approximate derivative obtained by performing the difference between the current error value and the value of the previous time instant. The low pass filter, used to compensate for the dynamics of the high pass filter is instead an integrator.

### 2.4.1. Structure

Starting from the high-pass filter $F_1(S)$, da Silva et al. suggested the following one

$$F_1(s) = 1 - e^{-s\tau_f} \tag{2.8}$$

where $\tau_f$ is a time delay constant.

Using the first-order Taylor expansion of the chosen filter, it is possible to show that it is

an approximate derivative:

$$F_1(s) \approx 1 - (1 - s\tau_f) \approx s\tau_f$$

Finally, the frequency response of $F_1(s)$ is

$$|F_1(s)| = \left[(1 - cos(\omega\tau_f))^2 + (sin(\omega\tau_f))^2\right]^{\frac{1}{2}} \tag{2.9}$$

$$\angle F_1(s) = \tan^{-1}\left(\frac{sin(\omega\tau_f)}{1 - cos(\omega\tau_f)}\right) \tag{2.10}$$

This filter has the advantage to be easily implementable in a digital controller and requires low computational effort. Additionally, the choice of this particular filter is justified by its frequency response: with a small time delay $\tau_f$ and at low frequencies, $F_1(s)$ has a phase angle close to $+\pi/2$, meaning that it behaves as a derivative at low frequencies.

Consequently, the proposed low-pass filter is an integrator, i.e.,

$$F_2(s) = \frac{1}{s} \tag{2.11}$$

Its purpose is to compensate for the high filter dynamics and the effect of the $R_1$ relay because it doesn't contribute to a null error at steady state. Finally, as already said, the $R_2$ relay is used to separate the bandwidth of $F_1$ and $F_2$ while the $R_1$ relay is used as the standard relay to generate a stable oscillation in the system. It is noticeable to say that the proposed solution requires tuning a low number of parameters, actually the time delay of the approximate derivative and the amplitude and hysteresis of the two relays. As is shown shortly, tuning rules can be introduced to further reduce the number of parameters: the user needs to define just the amplitude of the two relays (which will be the same), tune the hysteresis of $R_2$ only in case of noisy measurements and choose an appropriate time delay.

### 2.4.2. Analysis of the Robust Relay Feedback Structure

In this section we analyse the proposed robust relay structure using the describing function method and the Poincaré map analysis, to introduce the theoretical tools that can be used as a guide during the tuning phase of the experiment parameters. In particular, a lemma for the definition of the describing function and two theorems for the conditions of the existence of unimodal and symmetrical limit cycles are stated.

## Describing Function

Under the usual assumptions of the describing function method, we have that

- the input reference signal of the relay feedback is r = 0;
- a sine wave is present at the input of the nonlinear element, and its output contains no zero frequency and no subharmonic terms;
- the linear component has low-pass filter characteristics.

The describing function can be obtained by computing the ratio of the Fourier coefficient of the first harmonic at the output signal to the input signal amplitude. Starting from the high-pass filter, and assuming that the input signal has the form

$$\varepsilon(t) \;=\; E\sin(\omega t) \tag{2.12}$$

the output is in the form

$$y_1(t) \;=\; E|F_1(j\omega)|\,\sin\left(\omega t + \arg F_1(j\omega)\right)$$

where $F_1(j\omega) = |F_1(j\omega)|\angle\theta_1(\omega) = |F_1(j\omega)|\,e^{j\theta_1(\omega)}$is the transfer function of the high pass filter and its frequency response is described by equation (2.9).

Under the assumption of symmetric and odd nonlinearities, which we assume fulfilled in this dissertation, the computation of the first harmonic of the output of the relay $R_2$ (where $\Xi_2$ is the amplitude and $\hat{E}_2$ is its hysteresis) is simplified and the output is

$$v(t) \;=\; \frac{4\Xi_2}{\pi}\,\sin\left(\omega t + \theta_1(\omega) + \theta_{R2}(E)\right)$$

where

$$\theta_{R2}(t) \;=\; -\arcsin\frac{\hat{E}_2}{E}$$

from which it is possible to compute the describing function of the relay $R_2$

$$N_1(E,\omega) \;=\; \frac{4\Xi_2}{\pi E|F_1(j\omega)|}e^{j\theta_{R2}(E)}$$

Subsequently, the output of the low pass filter $F_2$, where $F_2(j\omega) = |F_2(j\omega)|\angle F_2(j\omega) = |F_2(j\omega)|e^{-j\frac{\pi}{2}}$ is

$$w(t) \;=\; \frac{4\Xi_2}{\pi}|F_2(j\omega)|\,\sin\left(\omega t + \theta_1(\omega) - \frac{\pi}{2} + \theta_{R2}(E)\right)$$

Finally, under the same assumptions of relay $R_2$, the first harmonic of the output of $R_1$ is

$$z(t) \;=\; \frac{4\Xi_1}{\pi}\sin\Big(\omega t + \theta_1(\omega) - \frac{\pi}{2} + \theta_{R1}(\tilde{E}) + \theta_{R2}(E)\Big)$$

from which it is possible to compute the describing function of $R_1$

$$N_2(E,\omega) \;=\; \frac{\Xi_1}{\Xi_2|F_2(j\omega)|}e^{j\theta_{R1}(\tilde{E})}$$

Given all these information, it is possible to compute the describing function of the overall system

$$N(E,\omega) \;=\; F_1 \times N_2 \times F_2 \times N_1 \;=\; \frac{4\Xi_1}{\pi E}e^{j\left(\theta_1(\omega)-\frac{\pi}{2}+\theta_{R1}(\tilde{E})+\theta_{R2}(E)\right)} \tag{2.13}$$

The obtained result can be reformulated as a lemma:

**lemma** Consider the robust relay feedback structure. Assume that the transfer function of $F_1(s)$ is in the form described by (2.8), and $F_2(s)$ is an integrator (2.11) and assume the phase angle of the filter $F_1(s)$ is $\theta_1$ at the frequency $\omega$. Therefore, the describing function of the RRFS method is given by

$$N(E,\omega) \;=\; \frac{4\Xi_1}{\pi E}e^{j\left(\theta_1(\omega)-\frac{\pi}{2}+\theta_{R1}(\tilde{E})+\theta_{R2}(E)\right)} \tag{2.14}$$

where

$$\theta_{Ri}(E_i) \;=\; -\arcsin\frac{\hat{E}_i}{E_i} \quad \text{for i} = 1,2$$

$E_i$ is the relay input oscillation amplitude

$\Xi_i$ is the relay amplitude

$\hat{E}_i$ is the relay hysteresis

$$\tilde{E} \;=\; \frac{4\Xi_2}{\pi}|F_2(j\omega)|$$

As already said in section 2.3, the describing function is a very powerful tool to analyse the permanent oscillations of nonlinear systems. In the context of this thesis, the describing function is used to both provide easy-to-use rules to define the experiment required by the autotuner and by the controller to infer data about the system.

However, the main drawback of the describing function is that it is based on heuristics, meaning that the results obtained using it should be validated with another type of analysis. In this respect, one can discuss the existence and uniqueness of limit cycles with the aid of the Poincaré map analysis, developing two theorems to address the existence

and their stability.

## Poincaré Map analysis

Consider a single-input-single-output (SISO) LTI system satisfying the following linear dynamic equations

$$\begin{cases} \dot{x}_1 = Ax_1(t) + Bu_p(t) \\ y_p(t) = Cx_1(t) \end{cases} \tag{2.15}$$

where $x_1 \in \mathbb{R}^n$, $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times 1}$, $C \in \mathbb{R}^{1\times n}$ and A is Hurwitz. The transfer function of the related system results to be

$$G(s) = C(sI - A)^{-1}B$$

Consider now the feedback structure depicted in figure 2.2 and define $R_1$ the relay whose output is the input of the linear system and $R_2$ the relay used to separate the bandwidth of the two filters. The relay $R_1$ can be presented by the following control law

$$u_p(t) = \begin{cases} \{1\}, & \begin{array}{c}\text{if } w(t)>\varepsilon_1\text{,or } w(t)<\varepsilon_1 \\ \text{and } u_p(t\_)=1\end{array} \\ \{-1\}, & \begin{array}{c}\text{if } w(t)<\varepsilon_1\text{,or } w(t)>\varepsilon_1 \\ \text{and } u_p(t\_)=-1\end{array} \end{cases} \tag{2.16}$$

where $\varepsilon_1 > 0$ is the hysteresis parameter and $w(t\_)$ is the value of $w$ before time $t$.

Accordingly, the $R_2$ relay can be represented as

$$v(t) = \begin{cases} \{1\}, & \begin{array}{c}\text{if } y_1(t)>\varepsilon_2\text{,or } y_1(t)<\varepsilon_2 \\ \text{and } y(t\_)=1\end{array} \\ \{-1\}, & \begin{array}{c}\text{if } y_1(t)<\varepsilon_2\text{,or } \omega(t)>\varepsilon_2 \\ \text{and } y(t\_)=-1\end{array} \end{cases} \tag{2.17}$$

where $\varepsilon_2 > 0$ is the hysteresis parameter and $y_1(t\_)$ is the value of $y_1$ before time $t$.

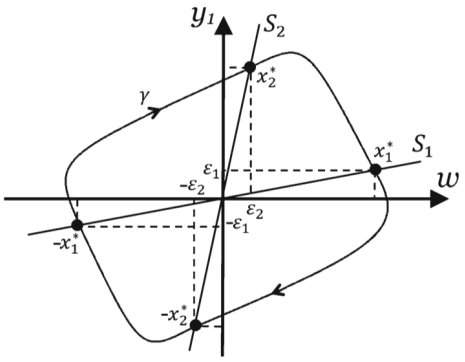



Figure 2.3: Trajectory on the switching surfaces for RRFS, taken from [12]

Additionally, the low pass filter has state space representation in the form

$$\begin{cases} \dot{x}_2 = v(t) \\ \omega(t) = x_2(t) \end{cases} \tag{2.18}$$

where $x_2 \in \mathbb{R}$.

Considering the high pass filter, the combined LTI system $(G(s)F_1(s))$is described as

$$\begin{cases} \dot{x}_1(t) = Ax_1(t) + Bu_p(t) \\ y_1(t) = C[x_1(t) - x_1(t - \tau_f)] \end{cases} \tag{2.19}$$

The switching surface, composed of a hyperplane of dimension $n-1$, for the relay $R_1$ is defined as

$$S_1 = \{x_1 \in \mathbb{R}^n : x_2 = \varepsilon_1\}$$

It is noticeable that the switching surface is a hyperplane that divides the state space into two distinct regions. Based on this consideration, it is possible to define $R_1^- = \{x_1 \in \mathbb{R}^n : x_2 < \varepsilon_1\}$ the region in which the system behaves as $\dot{x}_1 = Ax_1 - B$ and $R_1^+ = \{x_1 \in \mathbb{R}^n : x_2 > \varepsilon_1\}$ the region in which the system behaves as $\dot{x}_1 = Ax_1 + B$. Accordingly, the same considerations can be applied to relay $R_2$, defining its switching surface

$$S_1 = \{x_2 \in \mathbb{R}^n : C\left[x_1(t) - x_1(t - \tau_f)\right] = \varepsilon_2\}$$

And obtaining the following regions:

$$R_2^- = \{x_1 \in \mathbb{R}^n : C\left[x_1(t) - x_1(t - \tau_f)\right] < \varepsilon_2\}$$

and

$$R_2^+ = \{x_1 \in \mathbb{R}^n : C\left[x_1(t) - x_1(t - \tau_f)\right] > \varepsilon_2\}$$

Given these regions, it is possible to analyse the evolution of the switching system to identify the limit cycle with the properties introduced at the beginning of the section. In particular, consider a unimodal and symmetric limit cycle $\gamma$ with period $2t^*$ which is obtained from the initial condition $x^* \in S$ where S is defined as a switching surface.

Therefore, the closed orbit $\gamma$ crosses the switching surface S at $-x^* = x(t^*) \in S$. In other words, the limit cycle would cross the switching surface S at the state opposite to the initial one after the half period. Figure 2.3 (taken from[7]) shows the behaviour of a possible orbit along with the switching surfaces, in the plane of $w, y_1$. It is possible to see

that the switching surfaces divide the plane into four parts and in each one, one variable is growing while the other is reducing forcing the system into a permanent oscillation.

Finally, the theorems for existence and stability are introduced. They will not be proven here, as this would stray from the scope of this thesis.

**Theorem 1** Consider the linear system given by eq (2.18) and eq (2.19) connected in feedback with the relays. There exists a symmetrical and unimodal limit cycle with period $T = 2t^*$ if and only if the following conditions are satisfied:

1. $g_1(t) \triangleq C\left[-\left(I + e^{At^*}\right)^{-1}\left(e^{At^*} - I\right) - \left(e^{A\tau_f} + e^{At^*}\right)^{-1}\left(e^{At^*} - e^{A\tau_f}\right)\right]A^{-1}B = \varepsilon_2$,
2. $g_2(t^*) \triangleq \varepsilon_1 = 0$,
3. $y_1(t) = C\left[x_1(t) - x_1(t - \tau_f)\right] > \varepsilon_2, \forall t \in (0, t^*)$,
4. $\omega(t) = x_2(t) > \varepsilon_1, \forall t \in (0, t^*)$,

where

$$x_1^* = \left(I + e^{A*t^*}\right)^{-1}\left(e^{At^*} - I\right)A^{-1}B,$$
$$x_2^* = 0,$$

are the initial conditions $x_1(0) = x_1^*$ and $x_2(0) = x_2^*$ which leads to the periodic solution.

**Theorem 2** Consider the linear system given by equation (2.18) and eq (2.19) connected in feedback with the relays. Assume that there is a symmetric periodic solution with $t^* > \theta$, where $\theta$ represents the process time delay. The Jacobian of the Poincaré map is given by

$$W_i = \left(I - \frac{\omega_i C}{C\omega_i}\right)e^{At^*} \text{with i = 1, 2, 3} \tag{2.20}$$

where

$$\omega_1 = e^{At^*}\left(Ax_1^* - B\right) \tag{2.21}$$
$$\omega_2 = e^{A\left(t^* - \tau_f\right)}\left(Ax_\tau^* - B\right) \tag{2.22}$$
$$\omega_3 = 0 \tag{2.23}$$

The limit cycle is locally stable if and only if each $W_i$ has all its eigenvalues inside the unit disk. It will be unstable if $W_i$ has at least one eigenvalue outside the unit disk.

## FOPDT Model Identification

First Order Plus Dead Time (FOPDT in short) models are mathematical models used to describe the behaviour of dynamic systems in transfer function form. The FOPDT model describes the system defining a first-order dynamic equation with a time delay (also called dead time) and a time constant. Such models are often used to describe the behaviour of dynamical systems whose response to input variations is slow or damped. Alternatively, such a model can be used to "hide" higher order dynamics that few affect the response of the system, but still, such effects are not negligible.

The FOPDT model can be expressed as

$$G(j\omega) = \frac{K}{1 + j\omega T} e^{-j\omega\tau} \tag{2.24}$$

where $T$ is the time constant, $\tau$ is the dead time and $K$ is the gain.

Given the disturbance rejection properties of the RRFS method, da Silva et al. proposed a model identification solution that obtains more information by exploiting the low-frequency rejection of the relay structure. Initially, the system is excited with the RRFS control, and after a few oscillations, the frequency is obtained. After that, a step and a low-frequency square wave are added to the process input. The suggested frequency of the low-level signal is half the one obtained with the RRFS control. In this way, it is possible to obtain the static gain of the system plus two frequency points and, usually, the low-frequency point is the one of interest in the control application. With such information, a correction approach to increase the quality of a FOPDT identification is suggested by da Silva.

Proposition 1 Consider the transfer function G(s), given by Eq. (2.25). At frequency $\omega_i$, which is the oscillation frequency of the process, $G(j\omega_i)$ and $\phi(\omega_i)$ are the gain and the phase of the system, respectively.

$$G(s) = \frac{K}{1 + sT} e^{-sL} \tag{2.25}$$

Define the relative gain $\kappa(\omega_i) = \frac{|G(j\omega_i)|}{G(0)}$. According to Åström and Hägglund [12], $G(j\omega_i)$ is estimated by a relay feedback test and the parameters for the FOPDT model can be computed using the following equations

$$T(\omega_i) = \frac{1}{\omega_i}\sqrt{\kappa^{-2}(\omega_i) - 1}$$
$$L(\omega_i) = \frac{1}{\omega_i}\left(\phi(\omega_i) - \tan^{-1}\left(\sqrt{\kappa^{-2}(\omega_i) - 1}\right)\right)$$

Since two frequency points have been identified, they can be used to calculate two different FOPDT models. Using the method just introduced with the point obtained at high frequency $\omega_H$, it is possible to define the system

$$G_H(s) = \bar{G}_H(s)e^{-jL_H s}$$

where $L_H$ is the time delay of the high frequency model and $\bar{G}_H$ is the corresponding first order system part. The same operations can be done using the low frequency $\omega_L$ data point, obtaining the system

$$G_L(s) = \bar{G}_L(s)e^{-jL_L s}$$

Now, let us consider the initial model as

$$G_i(s) = \bar{G}_i(s)e^{-jL_i s}$$

and suppose that it is equal to $G_L$. Therefore, at frequency $\omega_H$ the phase of $G_i$ is

$$\phi(\omega_H) = \angle\bar{G}_i(\omega_H) + \angle e^{-jL_i\omega_H}$$

Now, let us assume that $|\bar{G}_i| = |\bar{G}_H|$. At frequency $\omega_H$, the phase difference between $G_H$ and $G_i$ is

$$\angle e^{-j\Delta L\omega_H} = \phi_H(\omega_H) - \phi_i(\omega_H)$$

Therefore

$$\Delta L = \frac{\omega_H - \omega_i}{\omega_H}$$

In conclusion, the corrected final model obtained by merging both models is

$$G_f(s) = \bar{G}_L(s)\, e^{-(L_i+\Delta L)s}$$

## 2.5. Internal model control

The Internal Model Control (IMC) is a principle and a control design technique that states (informally) that "a good controller incorporates a model of the dynamics that generate the signals which the control system is intended to track". Briefly explained,

the controller has a model of the "outside world" inside it and the control is intended to correct the divergences between the real process and the internal model.

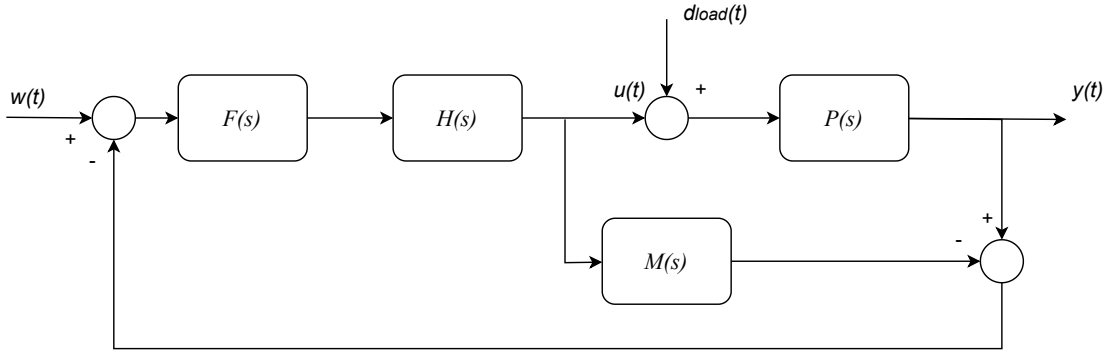



Figure 2.4: Typical scheme of the internal model control

Figure 2.4 shows the typical block scheme representation of the IMC layout. As one could notice, if the model is exact and there are no load disturbances, the closed-loop system is opened because the feedback is null. In other words, in nominal conditions ( $M = P$ and $d_{load} = 0$), the loop is open, hence

$$\frac{Y(s)}{W(s)} = P(s)H(s)F(s)$$

Furthermore, if one could set $H(s) = P(s)^{-1}$

$$\frac{Y(s)}{W(s)} = F(s)$$

In accordance, in absence of model errors or disturbances, the feedback opens spontaneously. Moreover, one can enforce the dynamics $w(t) \to y(t)$ with a relative degree at least equal to that of $P$ (for realisability reasons).

The system can be rearranged as shown in figure 2.5 and the control equation is obtained

$$C(s) = \frac{H(s)F(s)}{1 - H(s)F(s)M(s)}$$

The previous equation is useful to tune PI/PID controllers according to FOPDT models.

Let us assume

$$M(s) = \mu\frac{e^{-sD}}{1+sT}, H(s) = \frac{1+sT}{\mu}, F(s) = \frac{1}{1+s\lambda}$$

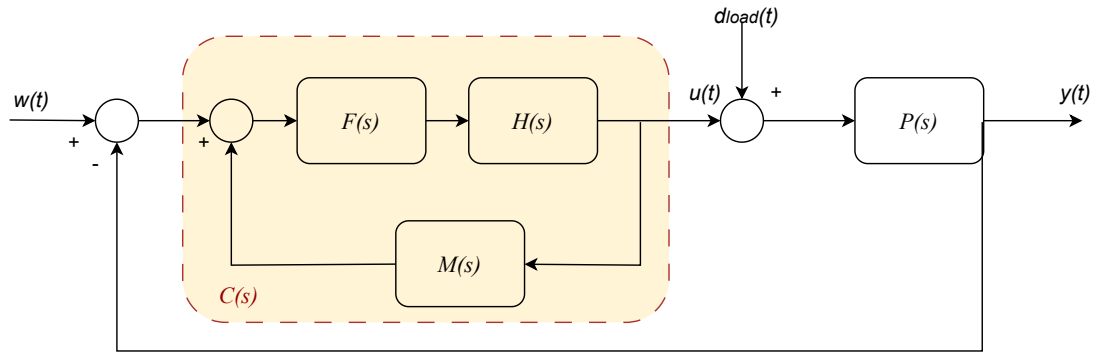



Figure 2.5: Rearrangment of the internal model control to highlight the controller

This corresponds to zero steady-state error for constant inputs, since $F(0) = 1$. Furthermore, the parameter $\lambda$ is easily interpreted as the desired closed-loop (dominant) time constant. Accordingly

$$C(s) = \frac{1}{\mu} \frac{1 + sT}{1 + s\lambda - e^{-sD}}$$

Using the (1,0) Padè approximation to substitute the time delay, the following controller is obtained

$$C(s) = \frac{1 + sT}{s\mu(\lambda + D)}$$

From which the tuning rule for a PI controller can be identified

$$\begin{cases} K = \frac{T}{\mu(\lambda+D)} \\ T_I = T \end{cases}$$

## 2.6. Contextual autotuning

The last – but very important – ingredient that we need to introduce in this chapter is the so called "contextual autotuning" technique, introduced in [15], that we shall now summarise and motivate in the context of the intended application

In its most general form, the contextual autotuning technique refers to a controller structure, expressed in transfer function form as

$$R(s, \theta_R) \qquad \theta_R \in \Re^{n_R} \tag{2.26}$$

as well as to a process model structure

$$M(s, \theta_M) \qquad \theta_M \in \Re^{n_M} \tag{2.27}$$

where $\theta_R$ and $\theta_M$ are parameter vectors.

The technique further requires to specify a tuning rule to determine $\theta_R$ based on $\theta_M$ and on a further vector $\theta_D \in \Re^{n_D}$ of design variables by means of $n_T$ *tuning equations*. In a case like the one we address, where fast tuning is mandatory, it is natural to limit the scope to *explicit* tuning equation or "rules", i.e., formulæ that compute the controller parameters without iterative methods, optimisations and the like. As such, obviously, $n_T = n_R$.

Finally, to use the technique, $n_P$ points of the process Nyquist curve $P(j\omega_i)$, $i = 1 \dots n_P$ are required, which are very naturally determined with relay experiment(s) as just discussed.

The above said, the problem of jointly parametrising the model $M$ and tuning the controller $R$ has $n_R + n_M + n_D$ variables, and here the main difference between contextual and non-contextual autotuning emerges. In the non-contextual case, the above equations are solved in two separate sets:

- first the $n_M$ ones relative to $\theta_M$ are considered, carrying out the model parametrisation with any method of choice — and in doing so following a criterion that is not related to the subsequent tuning, or said otherwise, evaluating the quality of a model parametrisation based just on its "adherence to the data" whatever this means;
- then the $n_R$ ones relative to $\theta_R$ are addressed, which corresponds in fact to applying the selected tuning rule and viewing the $n_D$ design variables as quantities to select (simplifying a bit for brevity) based on the tuning requirements, and possibly also on the parameters of the model (think for example of the various suggested formulæ to select the IMC-PID required time constant based on the observed normalise delay of the model [19]).

In the contextual case, on the contrary, the same equations are solved all together as a unique system, with the motivations and advantages that will now emerge while explaining how the said union is realised.

The first step is to write that "the model is exact at the known points of the process

frequency response", which means

$$M(j\omega_i, \theta_M) = P(j\omega_i) \qquad \forall\, i = 1 \dots n_P, \tag{2.28}$$

and provides $2n_P$ (real) equations. Most important, doing so implies that any conclusion drawn using the model in the frequency band comprising those points will carry over "reasonably well" to the control system with the real process. Note that in the non-contextual case this in general is not guaranteed.

Then second step is to take the chosen tuning rule and express the (nominal) closed-loop cutoff frequency $\omega_{cn}$ – the way this is done depends on the rule, we omit inessential details – and so obtain one further equation saying that $\omega_{cn}$ equals one of the frequencies $\omega_i$ of the known process frequency response points; the way the point is chosen gives rise to several variants, here too we do not enter a discussion that is not relevant for our purposes.

At this point, the overall problem contains $n_T + 2n_P + 1$ equations, hence it suffices to add $n_f$ real equations, where

$$n_R + n_M + n_D = n_T + 2n_P + 1 + n_f. \tag{2.29}$$

These $n_f$ equations can prescribe the value of design variables in the tuning rule, but also of regulator parameters, model parameters or any combination of choice thereof, or they might even just prescribe relationships between the quantities just mentioned.

In synthesis, under the sole constraint that the obtained system of equations be mathematically tractable, there is neither distinction nor any kind of hierarchy among the different sets (model, controller, design) of variables in the overall tuning problem: model parametrisation and controller tuning are treated jointly, whence the name "contextual" chosen for the technique.

The main advantage achieved is that the model used for the tuning is by construction "exact at the cutoff frequency", and this has an important consequence: one can use the closed-loop model formed with the parametrised $M$ and the tuned $R$ to predict the behaviour of the controlled variable reliably enough in the face of any input that enters the control system in a position such that its influence on the said variable depends only on the loop transfer function, provided that the tuning is made in such a way that at low frequency (with respect to the cutoff) the loop frequency response magnitude is "very large" — as incidentally any controller with integral action inherently guarantees. This is because a contextually parametrised model is (almost) exact at the cutoff frequency,

and as a consequence most likely incorrect at low frequency – for example, based on a set of frequency response points there is evidently no guarantee to match the process gain – where however a "very large" loop frequency response magnitude greatly quenches, and in practice eliminates, such a discrepancy.

Unfortunately, coming to the case at hand, this does not apply to matched (input) disturbances, as the nominal transfer function from these to the controlled variable is $P/(1 + RP)$. At present we are thus exploited contextual tuning as an effective policy, but not (yet) as a means to pre-evaluate a tuning beyond the possibilities of literature rules. If the problem just mentioned were solved, one could for example take records of load/power taken from previous system operation, feed them to the nominal model obtained after a tuning, and see how this would react, hence rapidly qualifying and assessing the achieved tuning quality in a manner strictly tied to the particular system to control. A frontier for this research is therefore to find some way to improve a contextually obtained model (for example by coupling it to a "low frequency" one) in such a way as to allow for a reliable estimation of the matched (input) disturbance response. This thesis does not address the matter, but no doubt provides a foundation for doing so in future works.

# 3 | Physics, models and tools

In this chapter, before entering the description of the proposed autotuner, we introduce the physics involved in the addressed control problem — i.e., we provide a control-targeted overview of heat generation and propagation in a microprocessor, also indicating how we addressed the relative thermal modelling.

In particular, we introduce two modelling approaches, conceived to address the two main phases of our control implementation: the control design and the performance analysis (through simulation). Additionally, we spend some words on how the thermal behaviour of a single-core device is usually modelled in the literature for control design, as well as on how such a modelling approach was modified to consider the thermal interaction in multicore systems. Moreover, we shall here talk about the fine-grain models that can be obtained through the finite volume method, and why these are useful to simulate the behaviour of distributed-parameters systems.

Finally, the chapter says some words about the tools used to model the addressed system: Modelica, which is a modelling language suitable for multi-domain systems, as well as 3D-ICE, a thermal simulator written in C for the fine-grain representation of thermal phenomena in CPUs.

## 3.1. Overview

Microprocessors are part of the family of monolithic integrated circuits, more commonly known as integrated circuits. An integrated circuit is a set of electronic devices on one small flat piece (a.k.a "chip") of semiconductor material, usually silicon. Many miniaturised transistors and other electronic components are integrated into a chip, resulting in circuits that are orders of magnitude smaller, faster, and cheaper than other solutions constructed with discrete components.

The production of integrated circuits is a highly complex manufacturing task, where a single block of silicon undergoes an articulated lithographic process to create the myriad of transistors composing the computational units, in a way that the electronic circuit is

indivisible from the rest of the material. The obtained result is a block of silicon with regions intertwined by very complex thermal interactions. In fact, on the active silicon layer, it is possible to find confined areas representing (in the case of microprocessors) the so-called "cores", along with cache, system on a chip (SoC), peripheral interfaces, and so on, which have different dimensions and thermal generations. Additionally, another layer of thermally conductive material is added on top of the silicon to protect it and enhance thermal exchange with the cooling system.

The final result of so articulated a manufacturing, viewed from the control stand point, is a thermal system comprising several dynamics. Looking closely at the silicon one more time, it is in fact possible to distinguish two regions, namely the active silicon and the bulk. The active silicon is the part involved in the computations, therefore the part generating the heat to dissipate, and the temperature profile of this layer greatly depends on its floorplan, which is the layout of the electronic circuit, nowadays – owing to the already mentioned dark silicon problem – designed according to the thermal interactions between cores. The rest of the silicon (the "bulk") is not involved in the computations, meaning that it does not contribute to heat generation, and its role is to behave as a thermal medium between the active silicon (which it mechanically comprehends and sustains) and the heat dissipation system. In a *scenario* where the need exists for thermal-aware management, it is important to analyse the thermal interaction between the individual cores in a view to devising policies and routines that best fit with the thermal management, or, equivalently, the minimisation of the dark silicon area, i.e. of the units that cannot be fully utilised at any given time.

Assuming the possibility to control the generated heat of a single core, which (as will be shown) is reasonable in modern microprocessors, the thermal interaction between cores needs considering for the control choices to be made: if the thermal interaction is small (i.e. the cores are weakly coupled), at least at the time scale of the transients to manage, then a decentralised control can be chosen, leading to lightweight, less computationally expensive and more easily implementable solutions. Otherwise, centralised controls must be employed, with all the downsides of such solutions. For this reason, a proper modelling of the thermal system considered, as well as a proper identification of its parameters, was a crucial step in the design of the control and all the results depend on it.

## 3.2. The physics to consider

The problem we are facing is related to the heat propagation within a solid object. The behaviour of this phenomenon can be modelled through the heat equation

$$\frac{\partial u}{\partial t} = \alpha \left( \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} \right) \tag{3.1}$$

where $u$ is the temperature of any point of the medium in any time instant, i.e. a function of space and time $u(x, y, z, t)$, $\alpha$ is the thermal diffusivity, $t$ is the time, and $x, y, z$ are the spatial coordinates in the chosen reference frame.

Equation (3.1) refers to a homogeneous and isotropic medium, i.e., a medium with constant properties throughout its entirety and that expands uniformly in the 3D space (mathematically speaking, the heat equation can be expanded to any arbitrary dimension). It is important to notice that, if the medium is not homogeneous and isotropic as it is in our case, $\alpha$ depends on the space coordinates and the equation is slightly more complicated. However, a full description of the diffusion phenomenon is beyond the scope of this dissertation.

The right-hand side of the heat equation is equivalent to the expansion of the so-called Laplacian operator, therefore the equation can be rewritten in the form

$$\dot{u} = \Delta u \tag{3.2}$$

where $\dot{u}$ is the time derivative of $u$, and $\Delta u$ its Laplacian. Equation (3.2) shows no explicit dependence on spatial coordinates because the Laplacian operator is traslationally and rotationally invariant, which represents an important property of homogeneous and isotropic media.

Loosely speaking, the Laplacian operator returns the difference between the average value of a neighborhood of a point and its value. Thus, since $u$ is here the temperature, the Laplacian returns the difference between how much warmer or cooler the material around the point is on average with respect to the point and the temperature of the point itself.

According to the second law of thermodynamics, heat flows from hotter regions to colder ones and the rate of exchange is proportional to temperature difference and thermal conductivity. Moreover, the temperature increase (or decrease) of an object is proportional to the entering (or exiting) heat and to the inverse of the mass, multiplied by the thermal specific heat capacity of the object.

Combining both considerations, the temperature variation rate of a point, denoted by $\dot{u}$, is proportional to how much warmer (or cooler) the surrounding material is. The diffusion coefficient $\alpha$ considers the thermal conductivity, the heat capacity, and the density of the material.

The heat equation implies mitigation over time of the maxima and minima of the solution, meaning that over time the maxima will be eroded and the minima will be filled in. This is because a point in the solution is stable only if it is equal to the average of its surrounding. Accordingly, if there is no thermal generation, the temperature at each point of the medium will eventually reach the same average value.

The mathematical representation of the heat propagation in a medium is an important means to understand the physical principles of the problem. The mathematical model may (and should) be expanded to account for the thermal generation and anisotropic and inhomogeneous media. However, this representation of the problem is already too complex for the control design and the control properties analysis, and for accurate simulations in general. In accordance, some simplifications must be introduced, as we are going to describe in the following section, to account for the control design and validation of the control strategy, prior to hardware implementation.

## 3.3. Purposed modelling

In control applications, the model of a physical process is fundamental for the design and analysis of control laws. Whenever there is the need to control a physical process, a set of inputs and outputs must be defined, where inputs are the variables to be controlled to enforce the output to behave as desired. Accordingly, during the control design, a control engineer has to manipulate the physical representation of the system to obtain another mathematical representation, the so-called dynamic model, which shows the input-output relationship of said system and how the system evolves in time. In mathematics, a dynamic system is a system in which a function represents the time dependence of a point in space, usually, using differential equations. In particular, the variables composing a dynamic model are:

- The state variables: those physical variables that encode the current state of the system.
- the outputs: those physical variables of interest that are measured and should be controlled.
- the inputs: the physical variables that are external to the system and can influence

its behaviour over time. The inputs can be further divided into controllable inputs: inputs that can be controlled using actuators, i.e. devices that actuate the control action, and exogenous inputs (or disturbances): inputs that cannot be controlled.

Then, the dynamic model can be represented as

$$\begin{cases} \dot{x} = f(x, u, d) \\ y = g(x, u, d) \end{cases} \tag{3.3}$$

where $x \in \mathbb{R}^n$ is the state, $u \in \mathbb{R}^p$ is the input, $d \in \mathbb{R}^r$ is the disturbance and $y \in \mathbb{R}^m$ is the output. The former equation represents how the system evolves in time, while the latter equation shows how the output is related to the state.

Another important aspect in control applications is the level of detail of a model, which changes greatly according to the modelling purposes. During the control design, the model should be "lightweight", in some sense, in order to keep the control problem addressable. Consequently, one should follow control design-oriented approaches that are able to provide simplified models that capture the key behaviour of the system. For example, during the development of a control design-oriented model, non-linear functions are commonly linearised around the equilibrium of interest, i.e. the state around which we want the process to remain, so as to resort to linear systems. The classical control theory is plenty of techniques to handle such systems, and usually, the effectiveness of such techniques justifies the approximations made.

On the other hand, the model should be as accurate as possible during the analysis of the system to capture all the characteristics ignored during the control design, and consequently, validate the approximations. Typically, the control performances (such as the phase margin and the control bandwidth) are addressed during the design, then, a simulation of the overall system can be used to validate the control strategy in a scenario much more complex than the one in which it was developed. Instabilities or more general undesired effects may happen that cannot be seen from the simplified model, but only through the analysis of the system according to accurate models.

In our control *scenario*, the controller tunes itself after an experiment; consequently, the "simplified" model is used to address the characteristics of the autotuned control, e.g. the phase margin and control bandwidth, rather than to design the control itself. In contrast, a detailed model can be used to validate the control performances in complex systems. Anticipating, the above motivates the choice made in this work, to carry out experimentations both with system-level models (in Modelica) and with fine-grain ones (with the 3D-ICE chip simulator).

### 3.3.1. Detailed modelling

During the performance analysis of the controller (usually performed through simulation), the model should be as accurate as possible to reproduce the behaviour of the system up to a precision level that is generally not required during the control design. However, the behaviour of the problem addressed in this thesis belongs to the class of distributed-parameter systems, i.e. physical systems whose state space is infinite-dimensional. Usually, such kind of systems is described by partial differential equations or delay differential equations. The nature of these systems is such that it is impossible to derive a dynamic model capable to simulate and predict the system without discretisation. In particular, in our control problem, the infinite dimensionality arises from the heat propagation which depends on time and spatial coordinates (as shown previously in equation (3.1)). Accordingly, the discretisation breaks down the partial differential equations arising from the spatial dependence.

Several methods exist to account for distributed systems, one of which is the Finite Volume (FV) method. The FV method divides the volume of the system into several small interconnected elements, then, the mathematical representation of the interconnection is defined as well as the dynamic equations arising from the discretisation. Accordingly, the number of states of the overall system is proportional to the number of elements and their properties, and subsequently, the discretisation choices greatly impact the degree of detail of the obtained model and its performance. Moreover, the discretisation permits accounting for the evolution of the state variables according to the space coordinates too, resulting in a "map" of the variable which evolves in time. Due to the high level of detail that this technique can reach, the FVM permits the so-called "fine-grained" simulations.

In this particular *scenario*, one can use the FVM to divide the system into a set of thermal nodes, which represent discrete regions where heat is generated or transferred. These nodes are connected by thermal resistances and capacitances, which represent the heat transfer and thermal storage capacities of the system. Such discretisation permits better representation capability of the system, at the cost of a huge increase in the number of dynamic equations. The nodes are usually boxes, and therefore, they have 6 thermal resistances (one for each face), and a thermal capacitance as shown in figure 3.1.

Once the nodes and thermal connections have been defined, a set of differential equations is developed to describe the thermal behaviour of each node over time. These equations are then solved numerically using software tools, allowing the temperature response of the system to be predicted under different operating conditions.

The simplified model of the system (obtained through FVM), which captures the key ther-

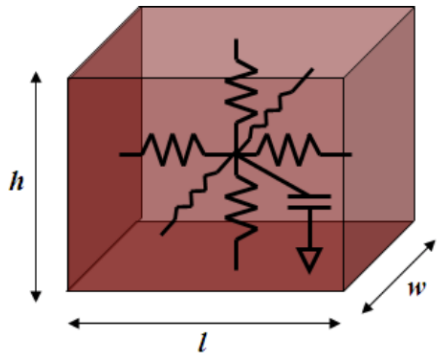



Figure 3.1: Figure representing a typical thermal node.

mal properties, allows accurate simulations of the thermal behaviour in a much shorter time than using detailed numerical simulations. Additionally, FVM permits better thermal insight into the system than control design-oriented modelling, which is able to capture only the main dynamics of the system. Some of the advantages of FVM are:

- speed: FVM allows faster simulations compared to detailed numerical simulations. This is particularly useful when testing multiple control solutions to address their trade-offs or when simulating large electronic systems.
- Accuracy: Even though the FVM is based on a simplified model, FVM can provide accurate simulations or predictions of the temperature transients of the regions inside the system. This is because the model captures the main features of the thermal behaviour of the system, such as thermal conductivity, heat capacity, and thermal resistance.

As a final consideration, FVM represents a valuable simulation environment for the accurate simulation of the distributed parameter system we aim to control, providing insights into the microprocessor thermal behaviour that cannot be addressed using just control-oriented modelling.

### 3.3.2. Control design-oriented modelling

Control design-oriented modelling is an approach to devising models of complex systems that are focused specifically on designing and implementing control strategies. As already stated, design-oriented models are usually simplified representations of the underlying physical system and these models are designed to capture the key dynamic behaviour relevant to the control problem. Accordingly, the models devised for control design are usually much simpler than those used in the analysis (and simulation) of the control

performances.

Design-oriented modelling aims to develop models that can be used to design and develop control strategies for the control problem. These control strategies usually consist of tracking a desired reference trajectory or rejecting the disturbances that can perturb the system.

Usually, the model contains differential equations that encode the response of the system to certain inputs or disturbances. These equations are obtained through knowledge of the physical principle of the system, from data, or a combination of both (in engineering jargon, the approaches are called "white-box", "black-box" or "gray-box" modelling respectively). Design-oriented modelling may also contain simplifying assumptions, such as neglecting nonlinearities, to maintain the system linear and, therefore, more suitable for the control design. Once the model is complete, it can be used to devise control strategies such as classical control, linear-quadratic control, or model predictive control. Typically, design-oriented modelling aims to derive linear models from which the transfer function is extrapolated. The transfer function is meaningful only for linear systems and encodes the relationship between the input and output spectra. It represents a powerful tool for the analysis and design of control for linear systems and most of the classical control theory is based on this mathematical tool. The transfer function contains the poles and zeros of the system, which are the time constants that the process takes to react to certain *stimuli*.

As said previously, the accounted physical process belongs to the class of distributed-parameter systems and discretisation must be performed to model it. Accordingly, a model feasible for the control design can be obtained by lowering the degree of detail of the finite volume method, up to a small number of volumes, (for example, three volumes as described in section 3.3.2).

Furthermore, in our control scenario the controller tunes itself according to the data obtained from an experiment, therefore there is no design phase asking for a design-oriented model. However, the simulation with a model with well-known dynamics (which is the case of design-oriented modelling) permits to address of the quality of the autotune operation: the analysis returns the information of the system in the form of frequency data point (see sections 2.3 2.4.2 for more details), if the transfer function of the system is known, the exact data point of the system is also known, therefore the estimation can be validated. On the same basis, one could set up other types of experiments to validate other control properties, such as the settling time, addressed by looking at the step response of the controller.

## The 3-capacities single-core model

The three-capacity thermal model is a commonly used model for analysing the thermal behaviour of a single-core microprocessor. It assumes that the thermal behaviour of a microprocessor can be represented with a series of thermal capacitances and resistors, where the capacitances represent the thermal energy stored in several elements of the microprocessor.

In particular, the three-capacity model considers the energy stored in the core, the bulk, and the spreader, represented by the corresponding thermal capacitances. On the other hand, the resistances represent the rate of exchange of thermal power between the elements. The resistance between the core and the bulk represents the rate at which the thermal energy is transferred from the core to the bulk and, accordingly, the resistance between the bulk and the spreader represents the rate at which thermal energy is transferred from the package to the surrounding environment.

Thanks to the duality between thermal and electrical systems, the obtained model can be translated into its equivalent electrical one and, therefore, the obtained model can be analysed according to the usual electrical analysis tools. In particular, assuming the interaction between the spreader and the environment constant -not a stunning limitation because the dynamics of the heat-sink is much slower than the thermal dynamics of the microprocessor, at the considered bandwidth- the resulting dynamic behaviour of the core consists of three dynamics with different time scales dictated by the thermal elements. Figure 3.2 shows the electrical equivalent of the three-capacity thermal model. Accordingly, one can use the electrical equivalent and the theoretical tools of circuit analysis to extrapolate the frequency response of the system, by setting as input the thermal generation (represented by the ideal current generator), and the temperature of the core (represented by Ta, the voltage stored by the first capacitor).

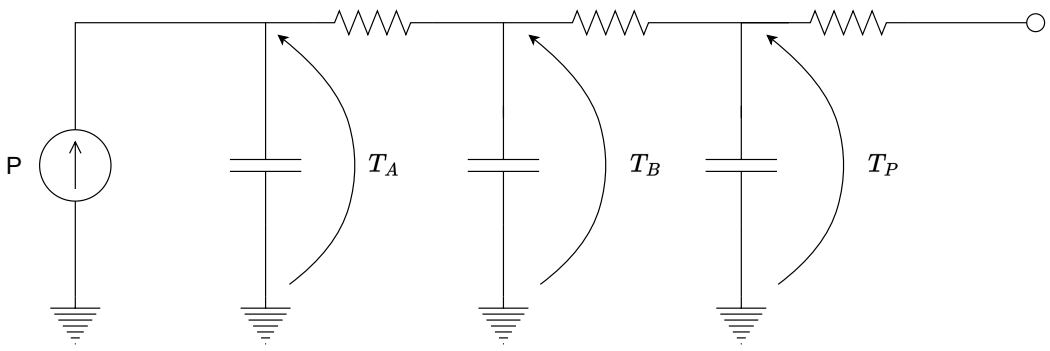



Figure 3.2: Electrical equivalent of the 3-capacities model of a single-core

However, the model is capable to represent just the thermal interaction of a single core interacting with the environment (which is the missing part to attach to the pin). Accordingly, the model should be expanded to account for the thermal interaction between cores, as shown in the following subsection.

### A system-level model for a multicore device

The three-capacity model is a valuable tool for capturing the key dynamics of a single core. However, it is not directly applicable to multicore simulations without some adjustments to account for the interaction between cores. To address this, a modified version of the model has been developed, which shares the slower dynamics associated with the spreader among the different cores (as shown in figure 3.3). This leaves the stressed core still undergoing the third-order dynamics but, now, the modified model can also capture the fifth-order dynamics of the cores' interaction. By introducing these changes, the modified model maintains the transient behaviour of the stressed core but also enables the modelling of core interactions.

## 3.4. Tools

In this section, we introduce the tools used during the development of the thesis to simulate the process for the control analysis. In particular, Modelica was used for the development of the three-capacity model and the initial testing of the autotuner prototype. Even though Modelica permits the modelling setup required by the fine-grained simulation, the work and time expenditure to do so would have been too high. Therefore, we chose 3D-ICE, a C library specifically designed to perform fine-grained thermal simulations of microprocessors.

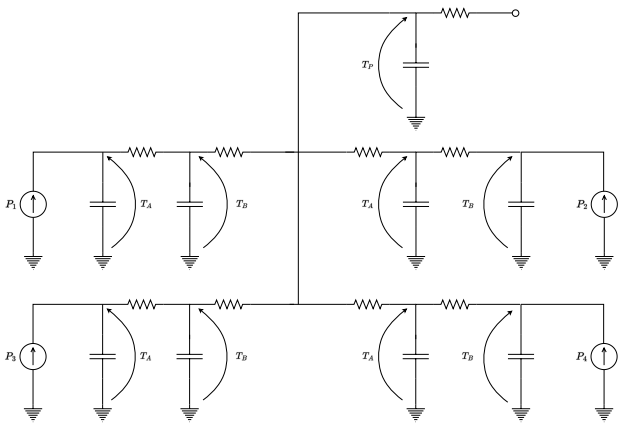



Figure 3.3: adaptation of the 3-capacities model for a four cores multicore

### 3.4.1. Modelica

Modelica is an object-oriented, declarative modelling language suitable for modelling multi-domain systems, i.e., systems in which different physical domains such as mechanical, electrical, and fluid systems interact with each other. The free language is developed by the non-profit Modelica Association [1] which also develops the Modelica Standard Library that contains 1400 generic components and 1200 functions in several domains.

Even though the Modelica syntax somehow resembles object-oriented programming languages such as C++, the concept of classes and inheritance greatly differs. Since Modelica is a modelling language, the classes are not compiled in the usual sense, but they are translated into objects which are then exercised by a simulation engine. Furthermore, the primary content of an object is a set of equations rather than instructions, as normally happens in programming languages. Equations represent equalities between different physical variables, meaning that equations don't have a predefined causality as assignment instructions do. The said equalities must be fulfilled at every time instant, otherwise the simulation is incorrect. Thus the simulation engine may, and usually it does, manipulate the equations symbolically to determine their order of execution and which components are inputs and which others are outputs. Once the model is defined, the simulation engine generates a set of internal binaries which are compiled and then used to simulate the overall system.

One key feature of Modelica is the possibility to model the systems both through block-modelling and acausal-modelling. The former is the typical representation of control theory, where the modelled elements are represented by blocks that accept some inputs and have some outputs. The block-modelling technique enforces an input-output relation, and one should spend some time and effort to obtain the model from the physical representation of the process, where, usually, the input-output relation is hard to see. This type of representation allows seeing the flow of information in the system, thanks to the input-output identification of the modelled elements. On the other hand, the representation is fixed and a modification of the underlying physics (for example by considering the nonlinear interaction rather than its linearisation) usually leads to a complete rewrite of the model.

Alternatively, one could use the a-causal modelling approach in which the behaviour of the element is defined, rather than the input-output relation. The result is a set of symbolic equations that the solver has to deal with to extrapolate the evolving behaviour of the physical system. The advantage of this representation is that the model maintains a physical interpretation, at the cost of a less clear flow of information.

### 3.4.2. 3D-ICE

The 3D Interlayer Cooling Emulator, 3D-ICE in short, represents the *compact transient thermal model* (CTTM in short) tool chosen for the simulation and testing of the proposed control [21].

The compact transient thermal modelling is an industrial application for the finite volume analysis of the transient temperatures in electrical circuits, such as microprocessors. It is a Thermal Emulator Library written in C and designed for Linux, and is capable of conducting transient thermal analyses on vertically stacked 3D integrated circuits featuring inter-tier Microchannel Liquid Cooling, using CTTM to model both solids and liquids. 3D-ICE aims to provide an environment for fine-grained simulation of integrated circuits with 3D architectures, i.e. integrated circuits which have multiple active silicon layers and, eventually, microchannels for internal liquid cooling.

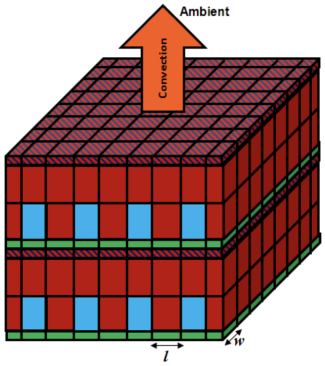



Figure 3.4: Representation of the discretisation performed by 3D ICE and how the heat traverses the stack. The stack is composed of two dies, each one containing a source layer (green) and liquid cavities for cooling (blue). The figure is taken from the 3D-ICE user manual

3D-ICE provides two simulation modes:

- batch simulation, in which the power profile generated by the active elements is known *a priori* and therefore, all the power-related data are passed to 3D-ICE in the form of a configuration file at the beginning of the simulation. Then 3D-ICE simulates the system and returns a file containing the evolution of the thermal map.
- interactive simulation, in which the power profile is not known at the beginning of the simulation because, as in our case, it may depend on a thermal control policy. In our scenario, the controller manages the frequency through the DVFS module, consequently, the power profile depends on the evolution of the core temperature,

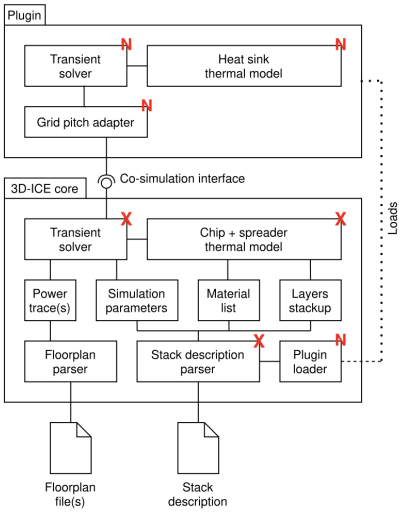



Figure 3.5: uml representation of the simulator modules and the co-simulation FMI interface

which cannot be known at the beginning of the simulation. Consequently, the power profile is provided to the simulator in real time, thanks to the co-simulation offered by 3D-ICE. In particular, 3D-ICE offers a TCP/IP socket interface for client-server communication to implement the interactive simulation. The policy, or the thermal profile generator, is encoded in the client, which sends the profile to the server through the socket and asks the server to take a step forward in the simulation. On the other hand, the server encapsulates the 3D-ICE simulator and waits for the insertion of the power profile and the request to simulate a step. Once the simulation step is done, the server returns the cores temperature profile to the client, which can repeat the steps.

Additionally, the library provides support for FMI integration for the co-simulation of externally-modelled heat sinks. In fact, heat sinks can be plugged through the use of dynamic libraries, and FMI-compatible loaders are provided for this purpose. The user could define custom heat sinks in languages such as Python or C++, or model them in Modelica and export the model through the FMI exporter provided by Modelica itself.

For the fine-grained simulation, the user is required to provide the emulator with two files:

- the stack description file: in this file, the user describes the 3D IC thermal problem to be solved. This contains information about the structure, the physical properties of the materials the layers are made of, the discretisation parameters, the analysis parameters, and the instruction for outputting the desired information. The information about the structure regards the width and length of the layers, their number

and height, and the location of the active layers, i.e. those generating power, and the heat sinks. Additionally, each layer can be made of a different material, as long as the material is defined and its heat capacity and thermal resistance are provided. Possibly, a layer could contain the cavity for liquid cooling. The user could require steady-state or transient simulations. In contrast to steady-state simulation, transient simulation enables the study of the evolution of the system over time, allowing the analysis of the dynamics of the system. in accordance with what has been said, the latter is the one of interest for our simulation purposes.

- the floorplan file: the floorplan is the layout of the active silicon layer, which defines the regions occupied by the cores, the caches and other elements of the microprocessor. 3D-ICE requires only the definition of the dimensions and location of the regions generating power. In the case of batch simulation, the floorplan contains also the power profiles of the active elements.

Figure 3.4 shows a possible 3D architecture use-case, corresponding to a *scenario* in which the processor is composed of two active silicon layers (green layers), red blocks represent the discretisation of the bulk, while the blue blocks represent the cavities for in-chip cooling (a feature that 3D-ICE offers and we mention for completeness, although not relevant for this thesis) and the dark red blocks represent the spreader layers. It is possible to notice also the heat flow direction considering that 3D-ICE assumes that all the walls are adiabatic except for the spreader layer.

# 4 | The proposed autotuner

In this chapter we discuss the functionalities implemented into the controller, and how these were turned into the autotuner software application. We start with some generalities and the addressed control structure, then move to the features to realise, and finally to the organisation of the autotuner into modules and to the communication among these.

## 4.1. Foreword

In the case where a controller is connected to a system with several inputs and several outputs, and the objective of the control is to modulate all outputs (i.e., each output has its own reference), then the number of inputs must be at least equal to the number of outputs. Also, the control should take this connection into account, and there are two ways to consider it depending on the degree of interaction between inputs and outputs:

- Decentralised control method: the controller consists of a series of controllers that modulate a single process output by acting on a single input. This is possible because the physical system has a preponderant one-to-one correspondence for each input-output pair, such that, each input largely influences only one output and weakly the others. In this case, it is possible to treat the MIMO (Multiple-Input, Multiple-Output) system as a collection of SISO (Single-Input, Single-Output) systems, and each controller is associated with one SISO system. A decentralised structure allows for a much simpler control law, because the various interconnections are not considered, and easier implementation. However, in the general case it does not guarantee any property – staility included – if not properly applied, or in other words, there must be solid motivations to support its applicability.
- Centralised control method: the controller simultaneously manages all control variables to modulate all outputs simultaneously. Usually, the centralised method is chosen whenever the MIMO system has too strong interconnections, making decentralised control inappropriate.

Given a MIMO system with strong interconnections, the complexity of the autoregulator's

experiment is also complicated because the autoregulator must test all possible input-output combinations for the experiment to be successful. In contrast, if decentralised control is possible, an autotuner only needs to investigate the interaction between the designated input-output pairs, greatly reducing the number of experiments.

For this reason, we performed an analysis on the strength of interconnection between the input and output elements of the process, in order to validate the hypothesis that the system is weakly coupled, allowing the use of decentralised control, with the resulting benefits. The next section introduces the physical system and the experiments conducted, showing the data obtained and how they demonstrate that a decentralised solution is viable. Based on these results, we decided to use decentralised control, and the next sections explain what design choices resulted from this initial one.

## 4.2. The addressed control structure

We start our treatise by discussing the control structure to which the presented autotuner needs to be applied. This is based on a model devised in light of physics and of data collected by a previous study [16].

In the said study, experiments were carried out on an Intel i5-6600K processor running Linux. The considered microprocessor has four cores, labeled 0-3, and during the experiment all the thermal controls were disabled, leaving in place only the hardware thermal protection (which, however, never intervened). Four tests were performed, in any of which just one core was stressed with different types of excitations, while the others were left without any load but the inevitable operating system tasks. In the first experiment, core 0 was excited with a maximum load step at time instant 0.1s using the cpuburn thermal

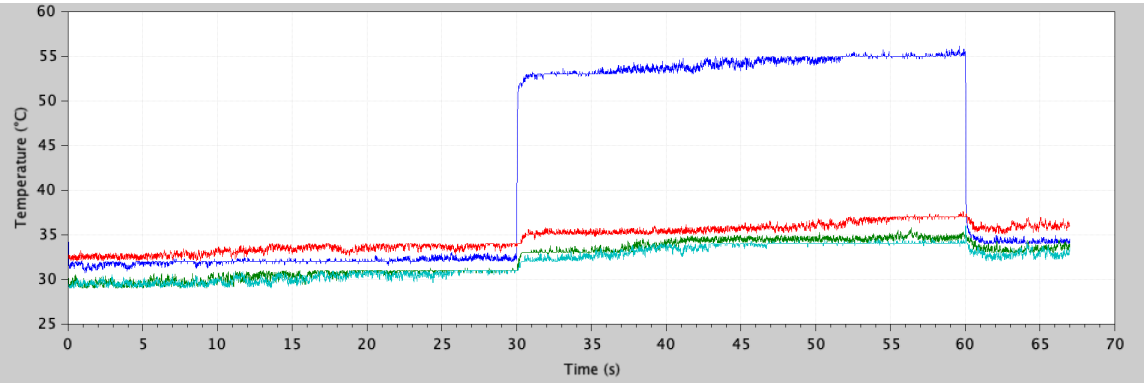



Figure 4.1: Raw data obtained from the study [16] showing the cores temperatures after a step change in the computational power request

stresser, which is a routine used in benchmarks to study the microprocessors' behaviour, stressing the cores by flooding them with a sequence of mathematical computations to reach the maximum utilisation of the core. In this way, it is possible to analyse the effect of the maximum computational power from the thermal point of view. In the second experiment, core 0 was kept excited with the cpuburn command, but the clock frequency of the cores was kept at the minimum value and then, at time 0.1s, the clock frequency was set to the maximum admissible value. The third and fourth experiments were performed in the same way with a different software load, a routine called cache-miss where the core was forced to allocate and deallocate some memory. The purpose of the cachemiss is to keep the core at 100% loaded(i.e. always active), but only to manage memory, that is, with small power consumption compared to math operations.

The raw data are depicted in picture 4.1. The difference between the thermal transients undergone by the stimulated core and the others is noticeable. In particular, data related to the stressed core dynamics show a qualitative behaviour similar to a first-order response while the thermal transients that arise from the cores' interaction exhibit a second-order (at least) response. Based on these considerations, an analysis was carried out in Matlab using the tfest function of the system identification toolbox assuming a fast pole plus a quasi-canceled slow pole for the single core dynamics, and two poles and a quasi-canceled slow pole for the core interaction.

The results showed a clear decoupling in the control bandwidth, validating the weak-coupling hypothesis and leading to the decentralised approach.

## 4.3. Implemented features

Given the result of the previous section, we chose to develop the controller according to the decentralised approach. Consequently, all considerations made from here on out relate to application to a SISO system, knowing that multiple controllers can be aggregated to address cases with multiple inputs.

As anticipated, the autoregulator performs an experiment to identify information about the system to which it is connected. For the experiment to be successful, the autotuner needs methods for system excitation, system response analysis, and controller self-tuning.

This section introduces the methods used for the success of the experiment, and specifically, the focus is on the rationale behind their choice. The elements are then taken up in the next section in which they are discussed in more detail.

Starting from the excitation methods, the autotuner provides three different exciters: the

relay, the relay with integrator, and the robust relay. Moreover, the proposed analysis methods are the describing function method and Fourier analysis. Finally, the calibration methods are closed-loop adjustment, IMC, and contextual IMC. All the elements introduced are then taken up later and elaborated upon during the description of the proposed autotuner.

- **Process *stimuli*: relay, relay plus integrator and robust relay**

  In our control scenario, the analysis experiment takes place under a condition such that the process cannot be guaranteed to be unperturbed by free motions of previous inputs (i.e., residual motions of actions that are no longer present). Consequently, the experiment performed must be meaningful even in the case of residual free motions. As described in 2.2, a common practice is to bring the physical process into a condition of forced permanent oscillation. Since free motion decays over time, after a few oscillations, it can be considered that the only perceptible motion is forced motion.

  To bring a linear system into permanent oscillation, it is sufficient to excite it with a static nonlinear system (i.e. a dynamic system without an equation of state). For this reason, we decided to use the relay and its integrated version. However, the experiment on a single core occurs while the other cores are busy doing the inevitable operating system operations, resulting in disturbances. Consequently, the exciter must be able to perform its action by rejecting the slow disturbances arising from the interaction with the other cores. For this reason, the robust method covered in the 2.4 section was implemented: a relay structure with filters to be able to filter away the low-frequency effects of the interaction between cores.

- **Response processing: describing function and Fourier analysis**

  Once a permanent oscillation has been obtained, information about the system can be extrapolated through the use of the describing function method or of Fourier analysis. The describing function method (as extensively described in the 2.3 section) represents a quick and easy approach for analysing the system, which can identify the frequency response of the system to the pulsation of the obtained permanent oscillation. On the other hand, Fourier analysis represents a more powerful tool, capable of simultaneously identifying multiple points in the frequency response of the system (as described later) but requires additional time and a greater computational load than the describing function.

  Both methods show merits and demerits and were implemented in order to verify

their performance in simulation.

- **Control parameters computation: IMC and contextual IMC** In order to explain the rationale behind the choice of these tuning methods, it is necessary to introduce the concepts of closed-loop system, bandwidth, and phase margin of the same. The closed-loop system is the system that is obtained by connecting the controller to the process, and its bandwidth represents the maximum frequency of the reference signal or disturbances that the controller is capable of tracking or rejecting, respectively. The phase margin represents the degree of robustness of the closed-loop system with respect to modelling errors: the higher it is in value, the greater the degree of modelling error the controller is able to tolerate before its action becomes ineffective and deleterious.

  The closed-loop adjustment allows the controller to be tuned so that it guarantees a certain phase margin by also forcing the closed-loop system to have the band obtained during the experiment. Having the same bandwidth allows the system to be treated at the point where the information obtained is most accurate, resulting in more effective control. On the other hand, the IMC method requires the development of a process model and allows the controller to be tuned so that the closed-loop system has a user-defined frequency. This method allows greater freedom than the previous method, however moving away from the point identified by the experiment results in the model being less representative. Finally, the contextual IMC method represents a repurposed version of the IMC in which the identification and self-tuning steps occur simultaneously. Specifically, the closed-loop bandwidth is placed equal to the permanent oscillation obtained during the experiment. This allows the system to be controlled at the point of maximum representativeness. In addition, the contextual IMC method also returns a sufficiently accurate model of the process (which, however, is not used in this discussion) to be able to predict the behaviour of the process itself.

Having come to the end of this review and exposed the reasons for the choices made, we can introduce the structure of the proposed autotuner, highlighting how these decisions were implemented on the practical side.

## 4.4. The resulting application

The overall autotuner application as stemming from the above features plays a particularly complex role, as it has to manage the different phases of stimulation, data processing, and controller calibration. Consequently, we decided to organise the application architecture

into the following modules:

- High-level controller: the high-level module that handles the various low-level elements, listens to the request for self-tuning, and performs the verification operations of the data obtained.
- Low-level controller: it is the element that modulates the process according to the control law.
- Exciter: The element that deals with the excitation of the process during the experiment.
- Analyser: The element that deals with the analysis of the response of the system during the experiment.

of which only the first mentioned element belongs to the higher hierarchical level, while the remaining ones belong to the lower one.

According to this separation of roles, the elements of the structure of the autotuner are presented in detail below.

### 4.4.1. The low-level controller

The results obtained from the model analysis showed that the single core dynamics have a prevalent first-order response. For this reason, a PI controller is more than enough to fulfill the control requirements.

The PI controller is very popular in industrial applications due to its simplicity and wide applicability. A PI controller continuously computes the error value $e(t)$ as the difference between a setpoint and the measurement of a process variable to be controlled. The output, used to correct the error, is proportional to the error, and its integral, both multiplied by convenient gains. Therefore, the effect of the PI controller can be tuned by changing the value of the three gains. The mathematical representation of the PI controller is

$$u(t) = K_P\, e(t) + K_I \int_0^t e(\tau)\, d\tau$$

However, the standard form, where $K_I$ is substituted with $K_P/T_I$, is more common, and reads

$$u(t) = K_P \left( e(t) + \frac{1}{T_I} \int_0^t e(\tau)\, d\tau \right)$$

The role of the proportional action is straightforward: the modulating action is proportional to the detected error, allowing the control action to follow it. However, the sole

use of a proportional gain is not enough in normal applications, and this is the reason why t the integral action is introduced. The integral action has a "delayed" contribution but guarantees zero error at the steady-state. A typical application of a PI controller to regulate a first-order system is to tune the integral time such that the associated zero cancels out the pole of the system. The obtained loop transfer function is a pure integrator, and the resulting control bandwidth can be tuned by setting the gain of the feedback system equal to the desired bandwidth. The result is a closed loop system with first-order dynamics and a phase margin equal to 90°, which is why the PI controller is "enough" to control first-order dynamics.

A common issue of PI controllers is the integrator's so-called "windup" problem when the control action is saturated. Real actuators have a range of possible values beyond which their action stops. What happens, however, is that, even when the such range is exceeded, the integrator keeps integrating the error requiring increasing action, which cannot be applied because the actuator is saturated. This result is a "block" of the control signal (hence of the controlled variable) even when the error eventually changes sign, as the integrator must be "discharged" and this takes time.

Several anti-windup solutions can be found in the literature to solve the issue. Since the controller is implemented in a digital environment, the chosen anti-windup is the "clamping" method which stops the integration whenever the required action would be outside the feasible range of the actuator.

Additionally, the low-level controller should not interfere with the action of the governor of the operating system. The governor is the part of the operating system in charge of managing the frequency (request to the DVFS) of the cores depending on the type of computational load. In practice, the governor manages the power/performances tradeoff and tunes the frequency request according to the power consumption profile chosen by the user or the operating system, e.g. the governor reduces the clock frequency if the user requires low-effort application, such as browsing or video streaming, to limit the power consumption.

Subsequently, the control action of the PI controller should always be lower than the governor requests. Therefore, the control action of the controller, before the saturation block, must be compared with the governor request, and the controller must choose the minimum between its control action and the governor request.

Figure 4.2 shows the block diagram of the final low-level controller.

Finally, the PI controller must have a routine that guarantees a smooth insertion of the

control after the autotuning experiment (that, we anticipate, requires to open the loop): when the experiment is over and the controller takes back the control, the transition should be without jumps. To match this requirement, the controller has an initialisation routine called "bumpless", in this routine, the controller sets the integration error in such a way that it covers the mismatch between the action of the exciter and the effect of the proportional gain of the PI. In this way, a transitional period is obtained between the exciter control and the modulating action of the PI controller, during which the integrated error is discharged.

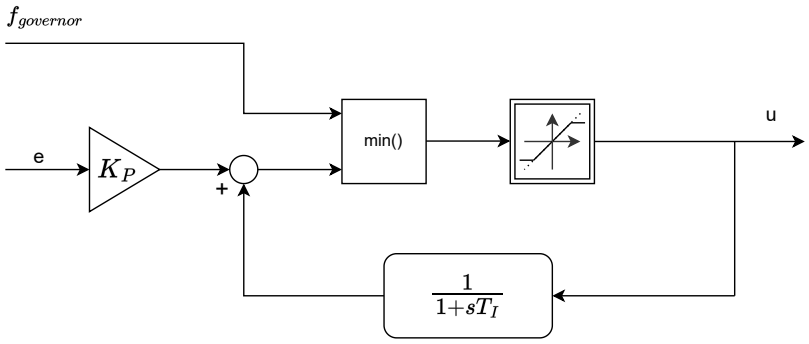



Figure 4.2: PI controller with clamping anti-windup solution and the min block to compare the action with the governor request

### 4.4.2. The experiment

The experiment is carried out by the joint action of the exciter and analyser whenever the autotuner receives the autotuning request. Accordingly, the high-level controller requires the exciter to perform the control action and pauses the low-level control. During the experiment, the analyser component has the role of analysing the system and tuning the controller accordingly.

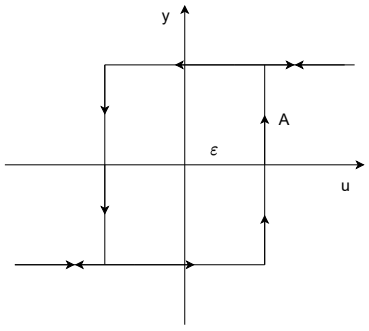



Figure 4.3: Graphical representation of the behaviour of a relay with hysteresis

As mentioned earlier, the autotuner offers three different exciters:

- relay: it is the simplest exciter structure possible. The relay could be ideal, and, therefore, the output depends only on the sign of the error, or it could have a hysteresis. Figure 4.3 shows the behaviour of a relay with hysteresis and it can be represented mathematically as

$$y(t) = \begin{cases} \{A\}, & \begin{array}{c}\text{if } u(t)>\varepsilon\text{,or } u(t)<\varepsilon \\ \text{and } y(t\_)=A\end{array} \\ \{-A\}, & \begin{array}{c}\text{if } u(t)<\varepsilon\text{,or } u(t)>\varepsilon \\ \text{and } y(t\_)=-A\end{array} \end{cases}$$

- relay plus integrator: a relay in which the output is integrated. The reason behind this choice is that the relay excites the process with step changes. However, a step change could overstress the system and strain the actuators too much. Integrating the output of the relay, the process is excited with ramps, instead of steps, resulting in a smoother action that does not overstress the actuators.
- robust relay: the robust relay is a particular structure composed of a relay plus filters to cancel out slow dynamics (it is deeply described in section 2.4). This feature can be used to reject the slow disturbances coming from the cores' interaction.

During the experiment realisation, the exciters are centered on the last output of the low-level controller. In such a way, the transition from the modulating action of the low-level controller to the experiment is smoother.

### 4.4.3. The analyser

The analyser is the low-level element in charge of examining the process and tuning the controller. This is implemented with two different methods to infer the frequency point, that is the describing function method and the Fourier analysis of the signals.

#### Describing function method

The describing function method (as discussed in section 2.3) allows associating to any nonlinear dynamical system a function in complex numbers, known as describing function, which represents the input-output relationship of permanent oscillations passing through said system. In particular, the describing function is defined as the ratio between the permanent oscillation at the output of the process and the one at the input. Given as input the input oscillation of the nonlinear system (sometimes it requires also the frequency of the oscillation), the describing function returns the output oscillation amplitude and the phase shift.

It is possible (and was shown in section 2.3) to use the describing function to compute the frequency data point of the linear process attached to the nonlinear system at the frequency of the oscillation. In fact, it is possible to show that the reciprocal opposite of the describing function is precisely the Laplace representation evaluated at the oscillation frequency. According to these considerations, the describing function method represents a simple yet effective method for the identification of a single point of the Laplace transform of the process.

The describing function method requires the knowledge of the amplitude of the oscillation, which value can be obtained by computing the distance between the maximum and minimum recorded value. To obtain such values, at each time step, the analyser evaluates the current value of the process variable with respect to the recorded highest and smallest terms. If the current value is bigger or smaller respectively, the corresponding variable is updated. Furthermore, the maximum and minimum values are reset at each period because the system could have a "slow" drift with respect to the oscillation period due to external disturbances, and the recorded minimum could be smaller than the new one, resulting in bigger measured oscillations than the real ones. Additionally, the describing function method requires the frequency of the oscillation. Assuming that the exciter is not subjected to chattering, the value of the period can be obtained by looking at two distinctive time instants in which the relay toggles up. Whenever the assumption doesn't hold, the time interval should be computed by looking at two consecutive maxima or minima. However, this solution is less robust because the detection of maxima and minima is subjected to noise and this could deteriorate the tuning quality.

## Fourier analysis

Alternatively, the information can be obtained through Fourier analysis. Since it is possible to assume that the input and output of the plant are periodic signals, the computation of a single data point $G(j\omega)$, which normally would require the integration along the infinite time horizon, can be computed integrating along a time interval equal to the period of the oscillation, provided that $\omega$ is a multiple of $\omega_r = 2\pi/P_r$ where $P_r$ is the period of the permanent oscillation, as demonstrated by [22]:

$$G(j\omega) \equiv \frac{\int_0^\infty y(t)\, e^{-j\omega t}\, dt}{\int_0^\infty u(t)\, e^{-j\omega t}\, dt} = \frac{\int_t^{t+P_r} y(t)\, e^{-j\omega t}\, dt}{\int_t^{t+P_r} u(t)\, e^{-j\omega t}\, dt} \tag{4.1}$$

The same is valid to compute the static gain:

$$G(0) \equiv \frac{\int_0^\infty y(t)\,dt}{\int_0^\infty u(t)\,dt} = \frac{\int_t^{t+P_r} y(t)\,dt}{\int_t^{t+P_r} u(t)\,dt}$$

It is important to notice that the microprocessor behaves as a heat generator, therefore, the effect of the generated power is an increment in temperature with respect to the ambient one. Accordingly, the process output (which is the temperature of the core) must be corrected by subtracting the ambient temperature in order to obtain a correct estimation of the system behaviour. Said so, the ambient temperature is not a stunning request since many modern motherboards detect the case temperature and it can be supposed constant during the experiment, removing the need for real-time measurement of the said variable.

Therefore, if the Fourier analysis is chosen and the minimum number of oscillations is reached, the analyser first infers the period of the oscillation, and then computes the integrals during the next period, at the end of which the experiment is considered over. It should be noted that the integrals introduced in equation (4.1) are complex and their results are complex numbers. Since the autotuner is implemented digitally, the integrals were discretised and their values are computed as the previous values plus the new ones times the integration step. Clearly, the cartesian representation is more suitable for the computations, knowing that

$$\mathfrak{Re}\left(\int x(t)\,e^{-j\omega t}\,dt\right) = \int \mathfrak{Re}\left(x(t)\,e^{-j\omega t}\right)\,dt = \int \left(\mathfrak{Re}\left(x(t)\right)\cos\left(-\omega t\right)\right)\,dt$$

The same results hold for the imaginary part

$$\mathfrak{Im}\left(\int x(t)\,e^{-j\omega t}\,dt\right) = \int \mathfrak{Im}\left(x(t)\,e^{-j\omega t}\right)\,dt = \int \left(\mathfrak{Im}\left(x(t)\right)\sin\left(-\omega t\right)\right)\,dt$$

discretising the results, the following relationships are obtained:

$$\mathfrak{Re}\left(\int x(t)\,e^{-j\omega t}\,dt\right) \approx \sum_0^\infty T_s \mathfrak{Re}\left(x(t)\right)\,\cos\left(-\omega t\right)$$
$$\mathfrak{Im}\left(\int x(t)\,e^{-j\omega t}\,dt\right) \approx \sum_0^\infty T_s \mathfrak{Im}\left(x(t)\right)\,\sin\left(-\omega t\right)$$

Finally, with a slight notation, defining $int\_u(t) = \int u(t)dt$ as the integral of the input, and $int\_y = \int y(t)dt$ as the integral of the (corrected) output, the data point can be

computed as

$$|G(j\omega)| = \frac{\sqrt{\mathfrak{Re}int\_y^2 + \mathfrak{Im}int\_y^2}}{\sqrt{\mathfrak{Re}int\_u^2 + \mathfrak{Im}int\_u^2}}$$
$$\angle G(j\omega) = \tan^{-1}\left(\frac{\mathfrak{Im}int\_y}{\mathfrak{Re}int\_y}\right) - \tan^{-1}\left(\frac{\mathfrak{Im}int\_u}{\mathfrak{Re}int\_u}\right)$$

The Fourier analysis allows the estimation of multiple data points, as long as they are evaluated at frequencies multiple of the natural frequency of the permanent oscillation and the system is excited with said oscillations.

Finally, the analyser is in charge of tuning the controller according to the computed data point(s). The controller can be tuned according to three different tuning rules: closed loop adjustment, internal model control (IMC) and its contextual version.

## Closed-loop adjustment

In the closed-loop adjustment method, just one frequency point is used. The idea is to tune the controller in such a way that the closed-loop system has a bandwidth equal to the frequency of the induced oscillation and the phase margin required by the user.

Assuming that the identified point is in the form $G(j\bar{\omega}) = A\,e^{j\phi_p}$, where $\bar{\omega}$ is the permanent oscillation frequency, and assuming that a PI controller is tuned, in order to have the closed-loop system with the desired phase margin $\phi_m$, the following equation must hold

$$R(j\bar{\omega})G(j\bar{\omega}) = e^{\phi_m - \pi}$$

Knowing that $R(j\omega) = K\frac{1+\omega T_I}{j\omega T_I}$, the equation can be rewrite as

$$K\frac{1+j\omega T_I}{j\omega T_I}\,Ae^{\phi_p} = e^{\phi_m - \pi}$$

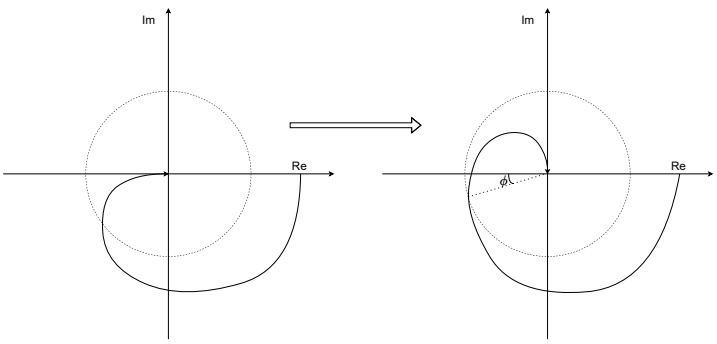



Figure 4.4: Graphical representation of the rationale besides the closed-loop adjustment

Computing the magnitude and the angle of both hand sides, the following relationship are obtained

$$\begin{cases} K\,\frac{\sqrt{1+\bar{\omega}^2{T_I}^2}}{\bar{\omega}^2{T_I}^2}\,A \;=\; 1 \\ -\frac{\pi}{2}+\tan^{-1}\bar{\omega}T_I+\phi_p \;=\; \phi_m-\pi \end{cases}$$

from which it is possible to define the parameters tuning rules

$$\begin{cases} T_I \;=\; \frac{1}{\bar{\omega}}\tan\left(\phi_m-\phi_p-\frac{\pi}{2}\right) \\ K \;=\; \frac{\bar{\omega}T_I}{A\sqrt{1+\bar{\omega}^2T_I^2}} \end{cases}$$

## Internal Model Control

The internal model control (IMC in short) is a tuning method that enforces the process (through the control action) to behave as a virtual process defined by the user.

The IMC method requires two data points, namely the static gain and a frequency data point. A first order plus time delay (FOPDT) model is obtained, which has form as shown in equation (4.2) after that the controller is tuned subsequently.

$$G(s) \;=\; \frac{\mu}{1+sT}e^{-sL} \tag{4.2}$$

The rule to obtain the FOPDT model are those defined by Åström and Hägglund [12]

$$\begin{aligned} T(\omega_i) &= \frac{1}{\omega_i}\sqrt{\kappa^{-2}(\omega_i)-1} \\ L(\omega_i) &= \frac{1}{\omega_i}\left(\phi(\omega_i)-\tan^{-1}\left(\sqrt{\kappa^{-2}(\omega_i)-1}\right)\right) \\ \mu &= G(0) \end{aligned}$$

where $\kappa(\omega_i)=\frac{|G(j\omega_i)|}{G(0)}$ is the relative gain, $\phi(\omega_i)=\angle G(j\omega_i)$ and $\omega_i$ is frequency of the permanent oscillation. Defining $\lambda$ to be the desired closed-loop time constant, (according to the considerations introduced in section 2.5) the PI parameters are tuned in the following way

$$\begin{aligned} K &= \frac{T}{\mu\,(\lambda+L)} \\ T_I &= T \end{aligned}$$

However, which this approach there is no way to guarantee a desired phase margin since this is related to the desired closed-loop time constant. Furthermore, in case the user has set the analyser to perform IMC tuning and has chosen the rrfs approach plus auxiliary

excitation, the analyser uses the FOPDT approach with phase adjustment proposed by da Silva et al. and discussed in section 2.4.2.

### Contextual IMC

The last implemented method to tune the controller is a variant of the internal model control introduced by Leva et al. [15] The main concept of this method is that when the recorded process information is used to obtain a model and the controller is tuned accordingly, a waste of information happens. However, if the information is used to obtain the model along with tuning the controller, whence the word "contextual", less information is wasted. Assuming that the identified data point of the system is $G(j\bar{\omega}) = A_P e^{j\varphi_P}$ with frequency $\bar{\omega}$, the proposed tuning rule, which allows obtaining both the model and the controller parameters, is obtained solving the following system of equations

$$\begin{cases} T_I &= T \\ K &= \frac{T}{\mu(\lambda+L)} \\ A_P &= \frac{\mu}{\sqrt{1+(\bar{\omega}T)^2}} \\ \varphi_P &= -\tan^{-1}(\bar{\omega}T) - \bar{\omega}L \\ \bar{\omega} &= \frac{1}{L+\lambda} \end{cases} \tag{4.3}$$

where, by fixing $\lambda$ with the same interpretation as above, the following result is obtained

$$\begin{cases} L &= \frac{1}{\bar{\omega}} - \lambda \\ T &= -\frac{1}{\bar{\omega}}\tan(\bar{\omega}L + \varphi_P) \\ \mu &= A_P\sqrt{1+(\bar{\omega}T)^2} \\ T_I &= T \\ K &= \frac{T}{\mu(L+\lambda)} \end{cases} \tag{4.4}$$

It is important to note that the contextual internal model control, as the closed-loop phase adjustment, requires one data point only because it forces the closed-loop control bandwidth to equal the identified natural frequency, which is where the identified model is most accurate.

## 4.4.4. The high-level controller

The high-level control is responsible for managing the various operations described above. It is responsible for correctly calling low-level components only when necessary, handling the various events that may occur such as timeout or negative check of the analysis.

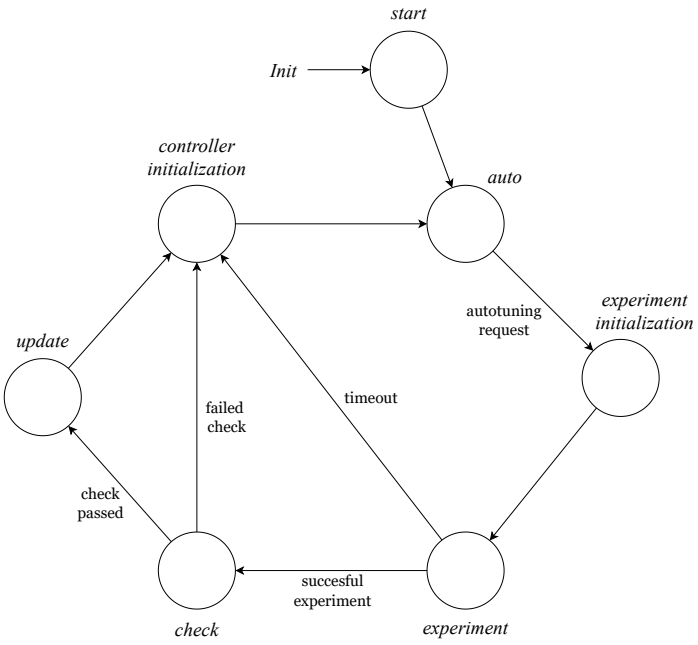



Figure 4.5: High-level automata

As a form of precaution, the controller requires a range of values within which the user expects that the identified point of the process can be found. Accordingly, the high-level controller verifies that the point obtained through the experiment is within this range, and proceeds with calibration only if the verification is fulfilled.

A state machine was developed to handle the different scenarios and the autotuning request event. Seven states were defined, namely start, auto, con_initialisation, exp_initialisation, experimenting, check, and update as shown in figure 4.5 which represents the so-realised finite state machine (or automaton).

- start state. This state is visited one time only during the operation of the controller (at the beginning) and the main purpose is to initialise the autotuner. After one iteration, the controller enters the following state.
- auto state. It is the main state and the automaton stays in this state most of the time. The auto state is associated with the modulating action of the low-level PI controller.
- exp_initialisation state. Whenever the autotuning request is received in the auto mode, the exp_initialisation is reached, where the experiment is set up and the exciter and analyser are initialised. After one time step, the automaton exits this state and reaches the following one.

- experimenting state. While the high-level controller is in this state, the experiment is performed. The automaton would move from this state only if the analysis is completed or the timeout occurs. In the former case, the automaton moves into the check state, otherwise, it enters the con_initialisation state.
- check state. the controller reaches this state only if the experiment is successful. In the check state, the high-level controller checks that the identified data point(s) is(are) in the user-defined admissible range. If the outcome is positive, the update state is reached, otherwise, the automaton moves into the con_initialisation.
- update state. The update state calls the tuning function of the analyser to tune the parameters of the controller. Another check is done to see that the parameters are allowable and do not make the controller unstable. After the update, the automaton reaches the con_initialisation state.
- con_initialisation state. In this state, the controller bumpless method is called. This is a "one step" state, meaning that the automaton would stay inside it for just one time step.

## 4.5. Inter-communication among sub-modules

During the execution of the tasks, the modules exchange information. Depending on the current state of the finite state machine defined previously, the information exchanged is different, as shown afterwards.

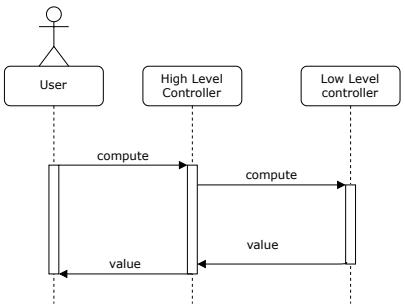



Figure 4.6: the autotuner calls the low-level controller compute method

When the automaton is in the auto state, the autotuner performs its computations.

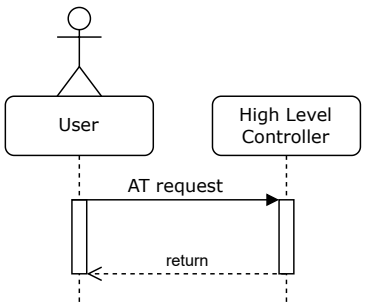



Figure 4.7: the autotuner records the autotuning request

If the autotune operation is requested, the high-level controller records the request.

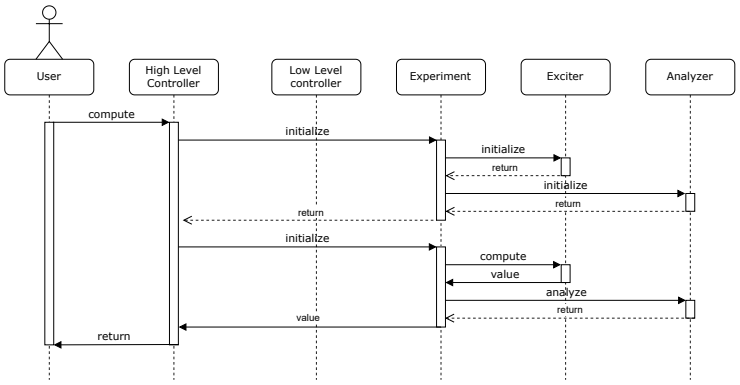



Figure 4.8: the autotuner initialises the experiment

The automaton enters the experiment initialisation state, therefore it calls the initialisation function of the experiment, which calls the initialisation functions of the experiment and the analyser. Once the initialisation is done, the autotuner requests the computation of the experiment. The computation is divided into two parts:

- the exciter computes its control action
- the analyser receives the current control action and the current value of the process variable.

The analysis requires the knowledge of the toggle of the exciter, therefore the analyser asks the exciter if the toggle happened. After the analysis step, the experiment returns the control action to the autotuner.

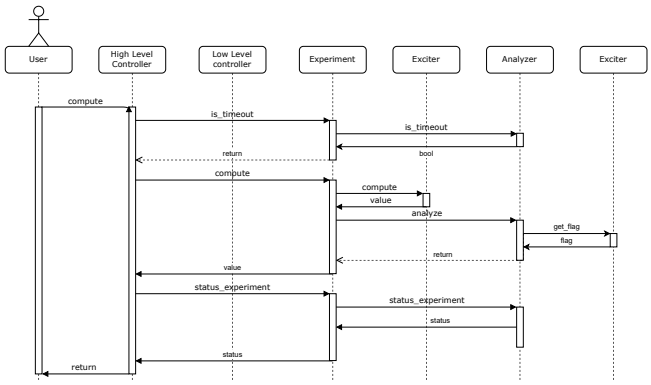



Figure 4.9: experiment step execution

Each time the autotuner is in the experimenting state, ti asks the experiment if the timeout is reached, in such case, the autotuner aborts the experiment and returns to the auto state, considering the experiment failed. Otherwise, the computations are the same as described earlier. Finally, the autotuner asks the experiment module if the experiment is complete, in such case the state machine advances to the sanity check state.

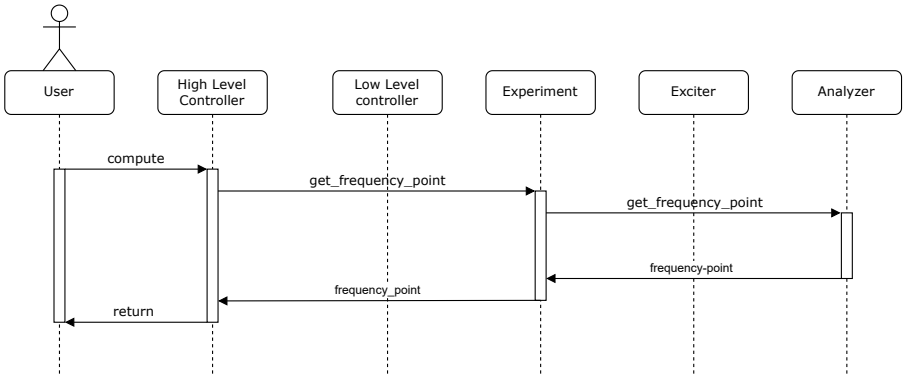



Figure 4.10: the autotuner requests the frequency point to perform the check

In the sanity check state, the autotuner asks the experiment the computed data point and checks that it is inside the acceptable range defined by the user. In case the check is affirmative, the autotuner enters the update state.

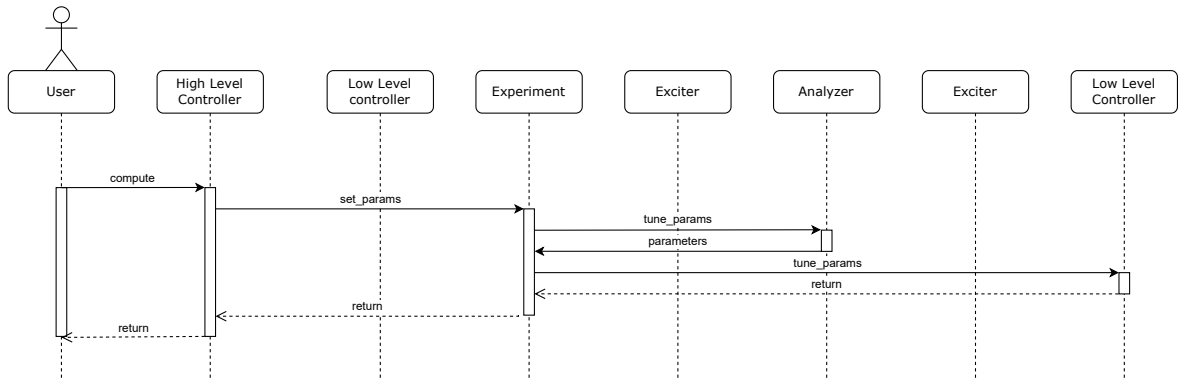



Figure 4.11: the autotuner tunes the low-level controller according to the obtained data point

Finally, the autotuner requires the experiment to tune the parameters of the controller.

# 5 | Software Implementation

In this chapter we discuss the implementation of the proposed autotuner, starting from the Modelica environment, talking about the related C language library, and concluding with the 3D-ICE implementation. The first section discusses the development of the Modelica library used for control-oriented modelling. The atomic elements identified, their operation, and their interaction are described as well. The models used for the DVFS core and controller and why these model choices were made are discussed. Next, the implementation of the CPU control-oriented model, introduced in section 3.3.2, is discussed. Finally, regarding Modelica, the controller implementation is discussed, both the monolithic code written in Modelica and the interface created for the use of the control written in C language.

The autoregulator, both in Modelica language and the C language, allows the use of 3 different exciters: the relay, its version with integrated output, and the robust method introduced in section 2.4(the secondary excitation is introduced in the C library). In addition, the system can be identified by the describing function method or by Fourier analysis. The calibration methods implemented in Modelica are closed-loop adjustment and IMC. It is important to note that the autoregulator written in Modelica was a prototype, and not all the features that can be found in the C library are also found in the autoregulator written in Modelica.

Next, the C (or wrapped C++) library is introduced, which was written both to be able to use the co-simulation offered by 3D-ICE, which offers simulations with greater detail, as described in section 3.4.2, and to be able to test the control in a real system. The library brings back all the features of Modelica's autotuner and adds the possibility of contextual IMC self-tuning, the identification method proposed by Da silva et al. described in section 2.4.2 and the robust variant with secondary excitation.

Next, the client-server communication offered by 3D-ICE is described, and how the client has been modified to be able to use the C library in "fine-grained" simulations.

## 5.1. Modelica implementation

Modelica supports the hierarchical representation of code through the use of the package class. This allows the code to be organised hierarchically, thus enabling easier organisation and navigation. In this section, we describe the package structure of the Modellica library created within this thesis

### 5.1.1. The package structure

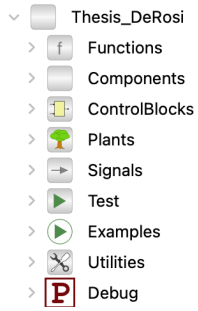



Figure 5.1: Modelica library

The package structure is depicted in picture 5.1 and following the order of the picture, a brief explanation of all the packages is provided.

- Functions package: this package contains all the functions used by the autotuner during the tuning phase. In particular, this contains the functions for the describing function method for the relay and relay plus integrator, the FOPDT estimation and the IMC tuning.
- Components package: this package contains the models of the smallest elements involved in the thermal interaction. In particular, this contains the model of the core and the DVFS module.
- ControlBlocks package: this package contains all the controllers created during the development of this work. The degree of complexity increased at each iteration because we started from the simple PI controller with governor override, then the autotuner written in Modelica, and, finally, the wrapper of the autotuner written in C. Each controller type has also a quadrupled version (i.e. the decentralised controller composed of four of them).
- Plants package: this package contains all the compound elements. In particular, this contains the model of a series layout multicore with or without the DVFS model.

- Test package: this package used to be a workbench for all the elements introduced during the development of the Modelica library.
- Example package: this package contains some examples of the autotuner performances according to several combinations of the exciters and analyser.
- Utilities package: this package contains the wrappers of the autotuner written in C.

In the following sections, we review in detail the main packages and the elements they contain.

### 5.1.2. The Components Blocks

The Component blocks are the atomic physical elements required to represent the thermal interaction of the processor. In particular, in our control scenario, the atomic elements that participate in the thermal generation are the core and its "actuator": the DVFS. Therefore, the package contains the thermal models of the core described in sections 3.3.2 and 3.3.2 and an extended model of the DVFS.

The model of the DVFS accounts for the actuator dynamics and the power generated by the core, which is then inputted in the thermal model of the core, as depicted in figure 5.2, where the block diagram of the DVFS is shown.

Considering the power generated by a single core, it is possible to compute it according to the equation

$$Q = \alpha \times l \times f \times v^2 \times C \tag{5.1}$$

where $v$ is the voltage of the core, $f$ is its clock frequency, $l$ is the load of the core, $\alpha$ is its utilisation and C is the total collective capacitance of the transistors. The load, which is a dimensionless coefficient between 0 and 1, represents the level of core utilisation on a time basis. So, given a time interval, the load coefficient represents the fraction of the interval in which the core is not idle. On the other hand, the utilisation $\alpha$, another adimensional coefficient comprised in the range $[0, 1]$, represents the software load of the core. Depending on the code instructions, the produced heat could be very different (as shown in [16]).

Furthermore, the voltage and frequency signals pass through two different dynamics which mimic the behaviour of the DVFS controller. In particular, the request of the frequency is evaluated to compute the minimum voltage necessary for the core, then the voltage signal passes through first-order dynamics to represent the response of the voltage controller.

Similarly, the frequency signal passes through second-order dynamics to simulate the frequency controller. Finally, the power is computed according to equation (5.1) and it is returned to the core thermal model.

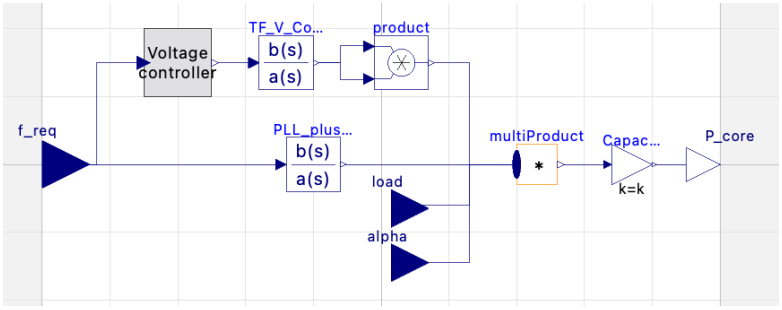



Figure 5.2: block diagram of the implemented DVFS module

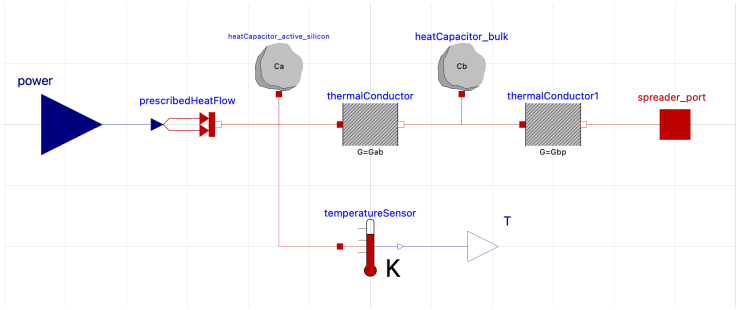



Figure 5.3: Modelica diagram of the two-capacity model

### 5.1.3. The Plant Blocks

The plant package groups the compound elements, consisting of the previously described components. Specifically, the internal thermal interaction of the microprocessor, and the microprocessor as a whole, interacting with the controller's frequency commands, external load and utilisation disturbances, and having the temperature sensors necessary for the controller to perpetrate its control action.

In addition, the microprocessor has been modelled with 4 cores; as a result, the thermal interactions within the processor, which properly represent the thermal aspect starting from power, are represented by 4 thermal models of cores, connected to each other. On the other hand, the microprocessor models comprise the thermal interaction and the power generation, along with some wiring to make it more appealing, to present it as a standalone object, and to interact with the controller, as depicted in figure 5.4.

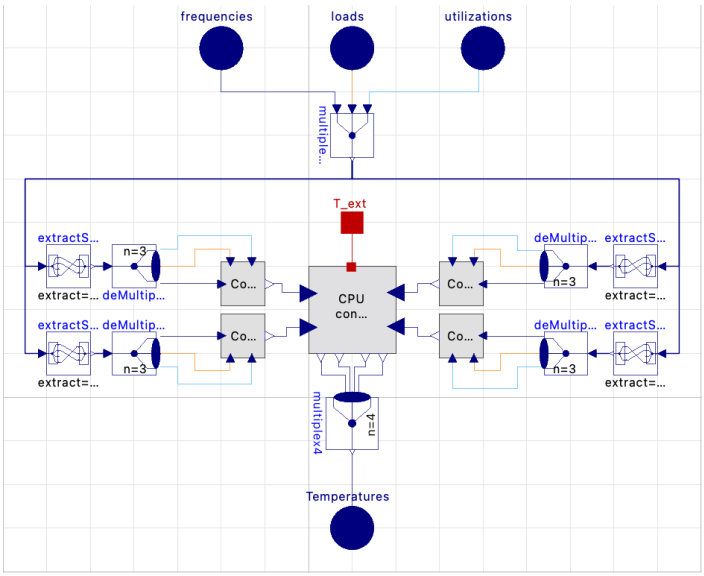



Figure 5.4: CPU model

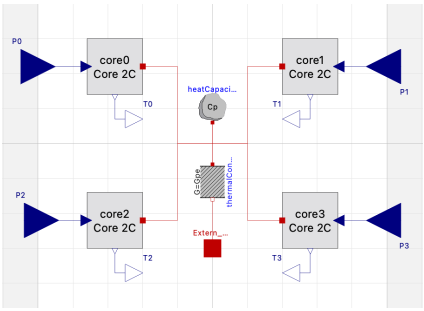



Figure 5.5: Modelica diagram of the multicore adaptation of three capacity model

### 5.1.4. The Control Blocks

The control blocks package contains all the different types of controllers implemented during the development of the thesis. Initially, a simplified controller, and its quadrupled version, were implemented to test the ideas set forth in 4. Figure 5.6 represent the block diagrams of the chosen controllers. The chosen controller is a continuous-time PI controller with an anti-windup. Additionally, the controller has a minimum override (see section 4.4.1) because it should always take the minimum between the modulating action and the frequency required by the governor. Thereafter, the autotuner was implemented inside Modelica. The Modelica language allows one to write algorithmic code, which resembles other high-level programming languages, to represent digital systems thanks to the when sample statement. The when statement evaluates the code only at the time instant when the associated condition becomes true, and the sample statement evaluates true when the specified time interval has elapsed since the last activation. Since the code was expected

to be tested on a physical device, as soon as the code proved successful, it was rewritten in C++, and because of the Modelica language's ability to be able to call external functions written in C or Fortran, the last control block represents the interface to the code written in C++.

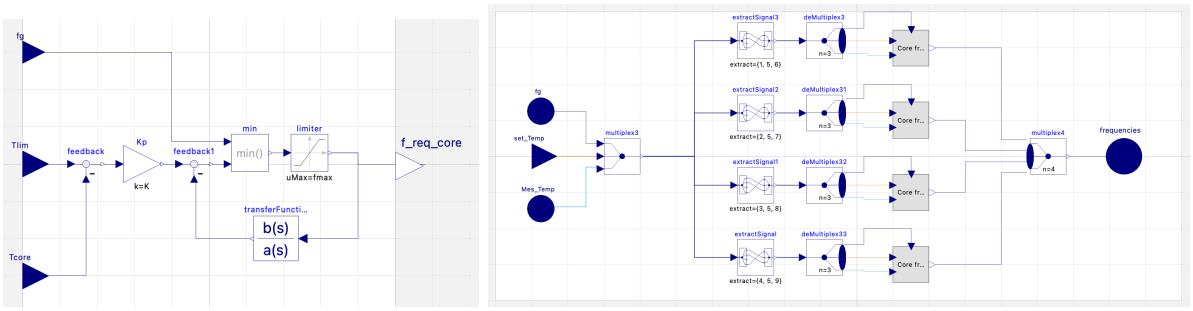



Figure 5.6: block schemes of the initial controller implemented in Modelica and its quadrupled version

### The Modelica autotuner

The autotuner was initially implemented using the Modelica algorithm language because the high-level paradigm allows faster implementation and testing of the code. However, the written code was monolithic, handling the different scenarios with several if statements during the execution of the code rather than addressing the variability at the beginning (as done, we anticipate, in the C library) because the Modelica algorithm language is less powerful than a pure programming language and offers fewer abstractions.

## 5.1.5. Tests and Examples

Finally, the main package is provided with a suite of testing and examples. The test package provides the user with a set of predefined tests to examine the behaviour of individual components and also their interaction. In particular, one can find tests of the thermal model of the core, the microprocessor, tests of the modulation of the simplified controller, the autoregulator, with or without disturbances, and finally, the same tests of the autoregulator written in C++. On the other hand, the package of examples is intended to show the behaviour of various controls under appropriately calibrated conditions. These represent the behaviour of the controller under ad-hoc conditions and the behaviour that should be expected under optimal conditions.

## 5.2. C library

The development of the code was done by looking at the possibility of being able to test it on a real system. For this reason, as soon as the code proved itself on Modelica, we decided to transpose it into the C++ language so that, one day, its executable could be used in processor thermal management. Even though the choice of C++ could seem inconvenient, because Modelica is not able to handle such code, the C++ language provides the ability to disguise its functions to look like C ones via the extern "C" keyword. Accordingly, we developed a C interface for using the code on Modelica and other software such as 3D-ICE.

Furthermore, the code, as described in section 4.4, well fit with the class paradigm offered by the C++ language, allowing the partitioning of the system which enhances the maintainability of the code. The aforementioned representation naturally identifies the atomic elements to be transposed into classes.

Furthermore, the autotuner provides different functionalities that are coded into different classes, but the possibility to choose which function to use should be available at runtime in order to work properly in Modelica. To match this requirement, we used the run time polymorphism paradigm during the development of the library: given a class inheritance tree, thanks to the run time polymorphism, it is possible to call the method of a child class even if it is referred to as its parent, as long as the involved function exists in both classes and it is labeled with the virtual keyword. In other words, virtual methods allow us to interact with groups of related objects uniformly: it is possible to create a child class and point to it using the parent pointer. If the called method has the virtual tag, at runtime the pointed object is checked and the method of the child class is called, rather than the method of the parent class.

Thus, one generic class per type was created, providing all the virtual functions required by the problem, and later on the generic classes were specialised to represent the particular types. At the end of the chapter is reported the UML of the entire library, which shows the complexity behind the proposed autotuner. Then, the composition of the autotuner with the desired functionalities is addressed at the construction of the object, since the builder function accepts as input several enumeration variables to build the correct “modules”. In addition, this provides the ability to expand the library whenever one wanted to try new types of elements without having to worry about general code modification, as long as the newly introduced classes inherit from the generalised classes.

In accordance with the above, the implementation of the sub-modules of the autotuner is

discussed, following the structure described in section 4.4.

### 5.2.1. Low-level controller

The superclass of the low-level controller has three main virtual functions that must be implemented while extending it:

- the compute function for the controllers: it accepts as input the setpoint, the process variable, the frequency of the governor, the autotuning request, and the time.
- the compute function for the exciters: it accepts as input the error and the time.
- the bumpless method: the method used during the transition from the experiment to the controller modulation.

The low-level controller implemented was a PI controller with clamping anti-windup solution. Based on the assumption that the governor requires a frequency equal to or lower than the maximum frequency achievable by the DVFS controller, and the governor action always overrides the action of the controller, there is no need to compare the modulating action to both the frequency of the governor and the maximum frequency. Instead, the controller compares its modulating action with respect to the lower limit, and the governor action is seen as the upper limit of the saturation. Algorithm 5.1 shows the behaviour of the action of the controller.

Algorithm 5.1 Digital implementation of a PI with anti-windup clamping method

1: $output \leftarrow 0$
2: $e \leftarrow K_P \times (SP - PV)$
3: $u_{par} \leftarrow ui + \frac{T_s}{T_I} e + e$
4: **if** $u_{par} > governor\ frequency$ **then**
5: $\quad output \leftarrow governor\ frequency$
6: **else if** $u_{par} < lowerlimit$ **then**
7: $\quad output \leftarrow lowerlimit$
8: **else**
9: $\quad u_i \leftarrow u_{par} - e$
10: $\quad output \leftarrow u_{par}$
11: **end if**

The clamping method is one of the most suitable anti-windup solutions in digital systems due to the elseif statement. In fact, many other solutions exist to account for the anti-windup problem and they sometimes act better than the clamping method. How-

ever, such solutions require additional computations and some also require additional parameters that should also be correctly tuned to avoid undesired effects. For example, back-propagation is a famous anti-windup solution alternative to clamping: the difference between the input and the output of the saturation is "back-propagated" multiplied by a gain. The back-propagated value is then subtracted from the integral action discharging the integral. One could demonstrate that the back-propagation method performs better with respect to the clamping method if correctly tuned. However, along with a higher number of computations (the multiplication and the subtraction of the backpropagation), no widely-accepted solutions exist to correctly tune the back-propagation gain, and a wrongly tuned gain would deteriorate completely the performance of the controller.

On the other hand, the clamping method does not require any additional parameters and requires fewer computations: in the anti-windup clamping method, the control action is computed at each time instant and, if the value does not exceed the user-defined limits, the action is accepted and the integral is updated, otherwise, the crossed limit is returned. Therefore, the clamping method was the most suitable in our control scenario.

### 5.2.2. The exciter

During the experiment, the analyser must know when the exciter relay toggles to perform its computations. In particular, it has to know the switching time to compute the period of the oscillation. For this reason, the exciter must record the switching event, and it must have methods to let the analyser know that the switch has happened.

Therefore, the exciter superclass extends the controller general one, adding the virtual methods needed to handle the switching events. In particular, the analyser calls the "get_flag" method to know if the switch event has happened and it calls the "reset_flag" method to flush the record.

Additionally, since the describing function depends on the nature of the nonlinear block, each exciter has a method that solves the describing function problem, accepting as inputs the amplitude and natural frequency of the permanent oscillation obtained during the experiment, and returning the estimated data point.

Moreover, a method is introduced to handle the initialisation of the exciter. For example, the center value of the relay must be set on the previous action of the low-level controller and the correct output should be chosen according to the position along the hysteresis.

Finally, four different types of exciter were implemented, namely:

- the relay, which can be ideal or with hysteresis

- the relay plus integrator, which is an ideal or with hysteresis relay whose output is integrated
- the rrfs introduced in section 2.4
- and the rrfs with the additional excitation (which was discussed in section 2.4)

## The relay

The mechanism of the relay is simple: the input of the relay is evaluated with respect to the hysteresis (if the value is zero, then the relay is ideal), if it is bigger, then set the output equals to the maximum, if the error is smaller than the opposite of the hysteresis, then set the output equal to the minimum, otherwise returns the previous value.

Additionally, the relay must record the switching event, and, without loss of generality, the rising event was chosen.

Algorithm 5.2 Digital implementation of a relay

1: **if** $e > hysteresis$ **then**
2: $output \leftarrow u_{cen} + amplitude$
3: $flag \leftarrow true$
4: **else if** $e < -hysteresis$ **then**
5: $output \leftarrow u_{cen} - amplitude$
6: **else**
7: $output \leftarrow output_{old}$
8: **end if**

## The relay plus integrator

As the name suggests, the relay plus integrator is composed of a relay whose output is integrated. The implementation extends the functionality of the relay with an additional variable to store the integration and all the methods are the same except for the "initialisation" and "compute" methods. In particular, during the initialisation, the center value of the inner relay is set to zero, while the integration value is initialised as the center value.

Additionally, the compute method is overridden and it calls the compute method of the inner relay, then, the returned value is integrated as shown in the algorithm 5.3

Algorithm 5.3 Digital implementation of a relay plus integrator

1: $ui \leftarrow ui + T_s \times relay :: compute(e)$
2: $output \leftarrow ui$

## The robust relay

The robust relay contains one relay object, named r1, one relay plus integrator object, called r2 (with the same roles as depicted in picture 2.2), and a vector used to store the records of the previous error values. During the initialisation, the "initialisation" methods of both relays are called. In particular, r1 is centered on the previous value of the control while r2 is set to zero. Additionally, each entry of the vector of records is set to the current error value.

Instead, the "compute" method implements the block diagram in picture 2.2: the method computes the difference between the current error value and its record, and inputs it in the r2 compute method. Then, the vector is updated, and the output of r2 is fed to the compute method of r1. Finally, the output of r1 is returned.

Algorithm 5.4 Digital implementation of the robust relay

1: $r2.compute(e - e_{record}[n_e])$
2: **for** $index \in [2, n_e]$, going backwards **do**
3: $\quad e_{record}[index] \leftarrow e_{record}[index - 1]$
4: **end for**
5: $e_{record}[1] \leftarrow e$
6: $output \leftarrow r1.compute(r2.get\_integral())$

The robust relay has a variant to implement the da Silva et al. FODPT identification (section 2.4.2). This variant extends the rrfs class overriding the compute function to add the square wave.

Algorithm 5.5 Digital implementation of the robust relay with additional square wave

1: $output \leftarrow 0$
2: **if** half_period is NaN **then**
3: $output \leftarrow RRFS :: compute(e)$
4: **else if** the time elapsed until the last switch is less than half the period **then**
5: $output \leftarrow amplitude_{squarewave} + RRFS :: compute(e)$
6: **else**
7: **if** $amplitude_{squarewave} > 0$ **then**
8: $amplitude_{squarewave} \leftarrow 0$
9: **else**
10: $amplitude_{squarewave} \leftarrow A$
11: **end if**
12: $output \leftarrow r1.compute(r2.get\_integral())$
13: **end if**

where $A$ is a user-defined parameter, namely the amplitude of the square wave.

Lines 7 to 10 are used to toggle the low-frequency action whenever the elapsed time is bigger than half a period.

### 5.2.3. Analyser

The analyser is the module that performs the analysis of the system, identifies the frequency data point and tunes the controller. The analysis is carried out by the analyse method, which takes as input the current control action, the current process variable and the current time. The class has also other routines to handle the initialisation, the check and the update phases.

Three different analysers were implemented, namely the describing function method and two versions of Fourier analysis, with one or two frequencies (the latter is necessary to use the improved model identification proposed by Da Silva and discussed in section 2.4.2).

#### The analyse method

The method is responsible for the analysis of the system. It waits for the user-defined number of oscillations before performing the estimation. While the analyser waits, it records the period and the amplitude -if the describing function analyser is chosen, otherwise just the period is tracked- each time the switching event occurs.

When the minimum number of oscillations has occurred, the analysis is performed. Depending on the type of analyser, the analysis is carried out differently: the describing function method does not require any additional time, since the last records of the period and the oscillation amplitude are sufficient. On the other hand, the natural frequency is required to perform the Fourier analysis, and the integration is done over a time period. Therefore another period is necessary to carry out the analysis.

Two pseudo algorithms are introduced to show the behaviour of both the analyser. The *describing function method* corresponds to the describing function method implemented in the exciter. In particular, the describing function method of the robust relay structure was introduced in section 2.4.2.

It is important to notice that the variant of the robust relay structure -with the additional low-frequency excitation- is only suitable for the Fourier analysis since the describing function analyser is not able to handle the low-frequency excitation.

Algorithm 5.6 Digital implementation of the describing function method

1: **if** $exciter \rightarrow get\_flag()$ **then**
2: $exciter \rightarrow reset\_flag()$
3: $period \leftarrow my\_time - previous\_switch$
4: $previous\_switch \leftarrow my\_time$
5: $osc\_count++$
6: **if** the number of oscillations is less than the maximum value **then**
7: $(max, min) \leftarrow reset()$
8: **end if**
9: **end if**
10: $(max, min) \leftarrow update(PV)$
11: **if** $osc\_count = max\_oscillation$ **then**
12: $\omega_{osc} \leftarrow 2\pi/period$
13: $amplitude \leftarrow \frac{max-min}{2}$
14: $G(j\omega_{osc}) \leftarrow exciter\, describing\, function\, method(amplitude, \omega_{osc})$
15: **end experiment**
16: **end if**

We could say that, at the beginning of the experiment, while the analyser is counting the number of oscillations passed, the recorded period and the maximum and minimum values are not trustworthy, because they could be subjected to the transient. For this reason, at each iteration, their value is reset.

In the Fourier analysis analyser, the *integrate* and *compute data points* functions implement the considerations introduces in section 4.4.3

Algorithm 5.7 Digital implementation of the Fourier analysis method

1: **if** $exciter \rightarrow get\_flag()$ **then**
2: $exciter \rightarrow reset\_flag()$
3: $period \leftarrow my\_time - previous\_switch$
4: $previous\_switch \leftarrow my\_time$
5: $osc\_count++$
6: **if** the number of oscillations is more than the maximum value **then**
7: $osc\_count \leftarrow 0$
8: $ok\_oscillation \leftarrow true$
9: $\omega_{osc} \leftarrow 2\pi/period$
10: **end if**
11: **end if**
12: **if** ok_oscillation and the experiment is not over **then**
13: $integrate(PV - PV\_offset, CS)$
14: **if** $osc\_count > 0$ **then**
15: $(G_0, G(j\omega_{osc})) \leftarrow compute\,data\,points()$
16: **end experiment**
17: **end if**
18: **end if**

### Tuning method

As already discussed in section 4.4.3, the analyser is able to tune the PI controller according to three different methods, namely: the closed-loop approach, the IMC and the IMC contextual methods. Said methods are coded into functions of the superclass which accept as input the pointers of the variables containing the control parameters. The analyser has also a function called tune_params accepting as inputs the pointers of the variables containing the control parameters, depending on the desired tuning method (which is stored in the form of an enumerated variable), the analyser calls the correct method.

## 5.2.4. High-level controller

During the execution of the code, whenever the autotuner is invoked, just the constructor of the high-level controller is called. In fact, the high-level controller takes care of the construction and deconstruction of the low-level "modules".

In order to exploit the runtime polymorphism, the modules are not stored inside the high-level controller class, but they are different elements that are allocated by the high-level controller constructor during its initialisation. Therefore, the controller object holds only the pointers to the module. This feature could be exploited in the future to invoke the autotuner only when the autotune is requested, attaching it to the preexisting low-level controller, and freeing memory when the autotune is not required.

The method responsible to encode the behaviour of the high-level control is called "compute". The compute method is responsible for the interaction between the high-level controller and the low-level elements and for the execution of the finite state machine of section 4.4.4, which was implemented using a series of if statements. An enumerated variable is used to represent the state of the automata, and depending on its value, the respective methods are called.

Additionally, the controller has a "get status" method, which returns the value of the inner variables. This method aims to provide insight into the controller, which is useful during debugging or simulations.

Finally, the autotuning request is handled in two different ways, depending on the environment: in Modelica, there are signals, which are inputs or outputs of functions, and therefore the autotuner should look at the value of the signal and, if it is toggled, it should record the request. On the other hand, in a pure C (or wrapped C++) environment, it is possible to call a function whenever the autotune request is formulated, removing unnecessary computations. Additionally, both operations are successful only if the finite state machine is in the "auto" state when receiving the request. To do so, the state variable is checked and it is set to "experiment initialisation" if it was "auto".

### Controller variants

While porting the C++ code in Modelica, we noticed an unwanted behaviour of the controller: the external function was called several times each time step, breaking its behaviour. It turned out that Modelica expects stateless code because the solver may not accept the step and redo it.

To match the requirement, a subclass of the original controller was implemented, called the "cached" version, which overrides the "compute" function. The new function checks that the received time is bigger than the recorded previous one, in such case it performs the computation, otherwise, it returns the previously computed value.

### 5.2.5. Additional structure and exceptions

The C++ code is enhanced with a set of customised exceptions that are used to detect possible failure on the memory allocation (e.g. return a null pointer after an allocation command), or unacceptable parameters of the system.

Additionally, a structure was introduced that contains all the parameters needed by the autotuner.

### 5.2.6. C interface

Once the C++ library was completed, a C interface was necessary to be called by the Modelica environment. Modelica has the capability to handle objects written in C or Fortran using the relative class called "external object", which requires a constructor and a destructor. The constructor is a function that returns a void pointer to the "external" structure, while the destructor is the function that encodes the routine to destroy the object.

Therefore, four different external C functions were implemented to port the constructor of the controller, the "compute" function, the "get status" function and the destructor of the cached variant of the controller.

The C function of the constructor returns the void casted pointer of the object while the destructor receives a void pointer which is cast into the cached variant and calls the respective destructor.

On the other hand, the compute and get status functions accept the same inputs as the method of the class plus the class pointer. Then the pointer is cast and the method of the pointed object is called.

## 5.3. 3D-ICE Implementation

Once the library was completed, and the autotuner was tested in Modelica, the controller was ported into 3D-ICE. As already stated in section 3.4.2, 3D-ICE is a C library for the fine-grained simulation of microprocessors. Said library offers the possibility of co-simulation based on TCP/IP socket connection between a server, which is the thermal emulator, and a client, which is the power profile generator, and offers templates for servers and clients. Accordingly, we modified the client template provided by 3D-ICE to host the autotuner written in C. As a final ingredient of the 3D-ICE implementation, we developed a YAML parser for a better user experience: condensing all the autotuner

parameters in a file outside the binaries, the user does not have to rebuild the client every time it has to change said parameters.

### 5.3.1. 3D-ICE Integration

In order to interact with 3D-ICE, we modified the client to accept our control action. As depicted in figure 5.7, which shows the flow chart of the client behaviour, the modified client has an initialisation stage, where the client creates the socket connection with the server, and then retrieves the number of active elements in the source layer. After that, the client spawns as many controllers as the number of active elements. Subsequently, the client enters the "operating" stage, where it computes the power to insert according to the controllers, sends the computed powers to the server, requires the server to simulate a time slot, and, finally, requests the current temperature of the cores. Additionally, the client visits the operating stage as many times as the number of slots required by the user.

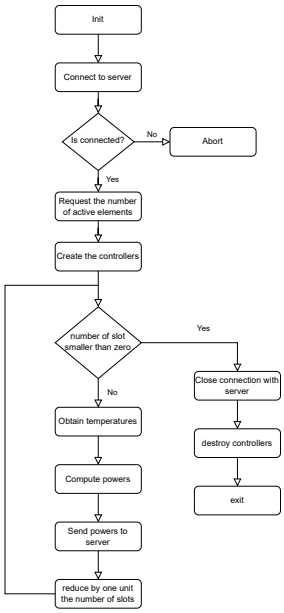



Figure 5.7: Flow chart diagram of the client

Since the controller returns a value comprised between 0 and 1, (which could be seen as the percentage of the maximum power), the power is computed as the product of this value and the maximum power that the core could produce.

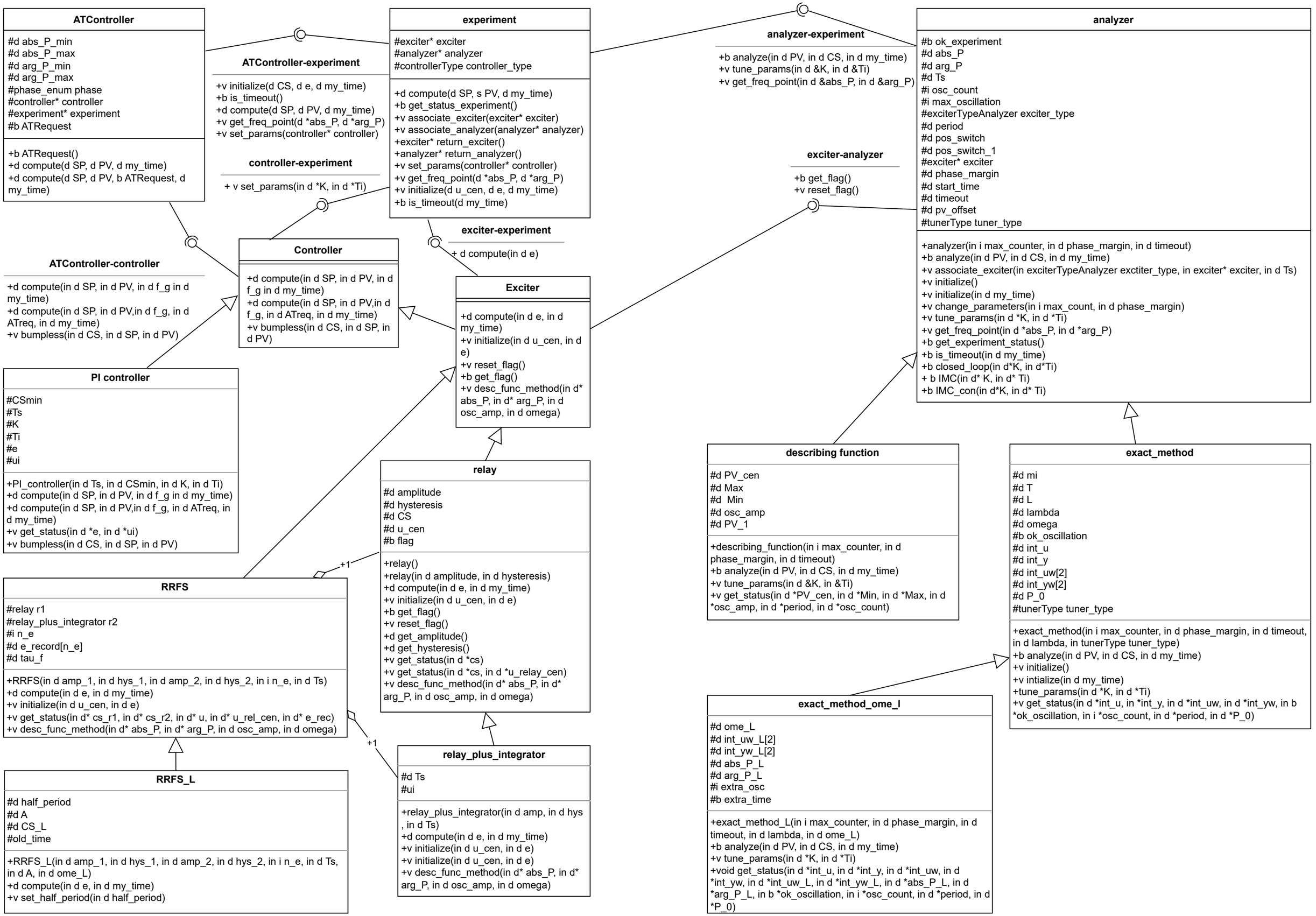
Additionally, the autotuner needs the current temperature of the core. Therefore, the stack description file must contain the instructions to let the server track the temperature of the cores. In particular, 3D-ICE allows outputting the temperature of "blocks" at

specific coordinates of the stack. Subsequently, we designed the stack description file to require the server to track the temperature of the "block" in the centre location of each core, resembling the behaviour of the thermal sensors of the processor.

### 5.3.2. YAML parser

To enhance the quality of service of the library, a free-source library to parse yaml files in C++ was added. Therefore, a simple parser was developed to read a yaml file with the autotuner parameters, allowing the user to not recompile the client any time a parameter of the autotuner is changed.

The parsing library is called "yaml-cpp" [3], and provides routines to read and parse yaml documents. In particular, the parser introduced requires the path to the YAML file and searches a dictionary called "Controller" which contains all the parameters of the controller. The output of the parser is the structure introduced in section 5.2.5


ATController
#d abs_P_min
#d abs_P_max
#d arg_P_min
#d arg_P_max
#phase_enum phase
#controller* controller
#experiment* experiment
#b ATRequest
+b ATRequest()
+d compute(d SP, d PV, d my_time)
+d compute(d SP, d PV, b ATRequest, d my_time)
ATController-experiment
+v initialize(d CS, d e, d my_time)
+b is_timeout()
+d compute(d SP, d PV, d my_time)
+v get_freq_point(d *abs_P, d *arg_P)
+v set_params(controller* controller)
experiment
#exciter* exciter
#analyzer* analyzer
#controllerType controller_type
+d compute(d SP, s PV, d my_time)
+b get_status_experiment()
+v associate_exciter(exciter* exciter)
+v associate_analyzer(analyzer* analyzer)
+exciter* return_exciter()
+analyzer* return_analyzer()
+v set_params(controller* controller)
+v get_freq_point(d *abs_P, d *arg_P)
+v initialize(d u_cen, d e, d my_time)
+b is_timeout(d my_time)
analyzer-experiment
+b analyze(in d PV, in d CS, in d my_time)
+v tune_params(in d &K, in d &Ti)
+v get_freq_point(in d &abs_P, in d &arg_P)
analyzer
#b ok_experiment
#d abs_P
#d arg_P
#d Ts
#i osc_count
#i max_oscillation
#exciterTypeAnalyzer exciter_type
#d period
#d pos_switch
#d pos_switch_1
#exciter* exciter
#d phase_margin
#d start_time
#d timeout
#d pv_offset
#tunerType tuner_type
+analyzer(in i max_counter, in d phase_margin, in d timeout)
+b analyze(in d PV, in d CS, in d my_time)
+v associate_exciter(in exciterTypeAnalyzer exctiter_type, in exciter* exciter, in d Ts)
+v initialize()
+v initialize(in d my_time)
+v change_parameters(in i max_count, in d phase_margin)
+v tune_params(in d *K, in d *Ti)
+v get_freq_point(in d *abs_P, in d *arg_P)
+b get_experiment_status()
+b is_timeout(in d my_time)
+b closed_loop(in d*K, in d*Ti)
+ b IMC(in d* K, in d* Ti)
+b IMC_con(in d*K, in d* Ti)
controller-experiment
+ v set_params(in d *K, in d *Ti)
exciter-experiment
+ d compute(in d e)
exciter-analyzer
+b get_flag()
+v reset_flag()
Controller
+d compute(in d SP, in d PV, in d f_g in d my_time)
+d compute(in d SP, in d PV,in d f_g, in d ATreq, in d my_time)
+v bumpless(in d CS, in d SP, in d PV)
ATController-controller
+d compute(in d SP, in d PV, in d f_g in d my_time)
+d compute(in d SP, in d PV,in d f_g, in d ATreq, in d my_time)
+v bumpless(in d CS, in d SP, in d PV)
Exciter
+d compute(in d e, in d my_time)
+v initialize(in d u_cen, in d e)
+v reset_flag()
+b get_flag()
+v desc_func_method(in d* abs_P, in d* arg_P, in d osc_amp, in d omega)
PI controller
#CSmin
#Ts
#K
#Ti
#e
#ui
+PI_controller(in d Ts, in d CSmin, in d K, in d Ti)
+d compute(in d SP, in d PV, in d f_g in d my_time)
+d compute(in d SP, in d PV,in d f_g, in d ATreq, in d my_time)
+v get_status(in d *e, in d *ui)
+v bumpless(in d CS, in d SP, in d PV)
relay
#d amplitude
#d hysteresis
#d CS
#d u_cen
#b flag
+relay()
+relay(in d amplitude, in d hysteresis)
+d compute(in d e, in d my_time)
+v initialize(in d u_cen, in d e)
+b get_flag()
+v reset_flag()
+d get_amplitude()
+d get_hysteresis()
+v get_status(in d *cs)
+v get_status(in d *cs, in d *u_relay_cen)
+v desc_func_method(in d* abs_P, in d* arg_P, in d osc_amp, in d omega)
+1
RRFS
#relay r1
#relay_plus_integrator r2
#i n_e
#d e_record[n_e]
#d tau_f
+RRFS(in d amp_1, in d hys_1, in d amp_2, in d hys_2, in i n_e, in d Ts)
+d compute(in d e, in d my_time)
+v initialize(in d u_cen, in d e)
+v get_status(in d* cs_r1, in d* cs_r2, in d* u, in d* u_rel_cen, in d* e_rec)
+v desc_func_method(in d* abs_P, in d* arg_P, in d osc_amp, in d omega)
+1
relay_plus_integrator
#d Ts
#ui
+relay_plus_integrator(in d amp, in d hys , in d Ts)
+d compute(in d e, in d my_time)
+v initialize(in d u_cen, in d e)
+v initialize(in d u_cen, in d e)
+v desc_func_method(in d* abs_P, in d* arg_P, in d osc_amp, in d omega)
RRFS_L
#d half_period
#d A
#d CS_L
#old_time
+RRFS_L(in d amp_1, in d hys_1, in d amp_2, in d hys_2, in i n_e, in d Ts, in d A, in d ome_L)
+d compute(in d e, in d my_time)
+v set_half_period(in d half_period)
describing function
#d PV_cen
#d Max
#d Min
#d osc_amp
#d PV_1
+describing_function(in i max_counter, in d phase_margin, in d timeout)
+b analyze(in d PV, in d CS, in d my_time)
+v tune_params(in d &K, in &Ti)
+v get_status(in d *PV_cen, in d *Min, in d *Max, in d *osc_amp, in d *period, in d *osc_count)
exact_method
#d mi
#d T
#d L
#d lambda
#d omega
#b ok_oscillation
#d int_u
#d int_y
#d int_uw[2]
#d int_yw[2]
#d P_0
#tunerType tuner_type
+exact_method(in i max_counter, in d phase_margin, in d timeout, in d lambda, in tunerType tuner_type)
+b analyze(in d PV, in d CS, in d my_time)
+v initialize()
+v intialize(in d my_time)
+tune_params(in d *K, in d *Ti)
+v get_status(in d *int_u, in *int_y, in d *int_uw, in d *int_yw, in b *ok_oscillation, in i *osc_count, in d *period, in d *P_0)
exact_method_ome_l
#d ome_L
#d int_uw_L[2]
#d int_yw_L[2]
#d abs_P_L
#d arg_P_L
#i extra_osc
#b extra_time
+exact_method_L(in i max_counter, in d phase_margin, in d timeout, in d lambda, in d ome_L)
+b analyze(in d PV, in d CS, in d my_time)
+v tune_params(in d *K, in d *Ti)
+void get_status(in d *int_u, in d *int_y, in d *int_uw, in d *int_yw, in d *int_uw_L, in d *int_yw_L, in d *abs_P_L, in d *arg_P_L, in b *ok_oscillation, in i *osc_count, in d *period, in d *P_0)

# 6 | Testing

In this section, we describe some of the simulation experiments we performed to validate the control approach proposed in the previous chapters, and sketch out the conclusions that we could draw based on them. Experiments were performed both in the Modelica environment (introduced in section 3.4.1) and in co-simulation with 3D-ICE (introduced in section 3.4.2).

## 6.1. Modelica experiments

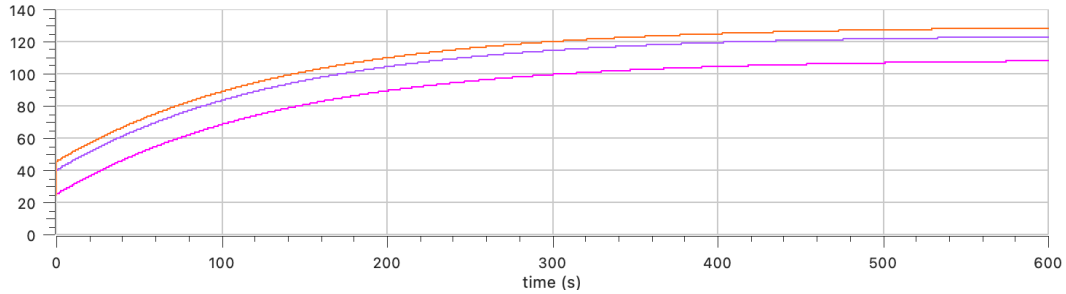
The purpose of these experiments is to verify the correct operation of the proposed auto-tuner with a model of the processor conceived for system-level studies.

As such, we used a multicore model built according to the 3-capacity approach (Section 3.3.2, the Modelica implementation of which was described in section 5.1.3). To briefly recap, the cores are represented by two thermal capacitances and two thermal resistances, which are then connected to the spreader, described by another thermal capacitance. Finally, the spreader is connected to the external environment by a thermal resistor, which represents the heat sink. For simplicity, this thermal resistance is constant. Assuming that all the cores can be represented by the same capacitances and resistances, the following parameters were chosen:

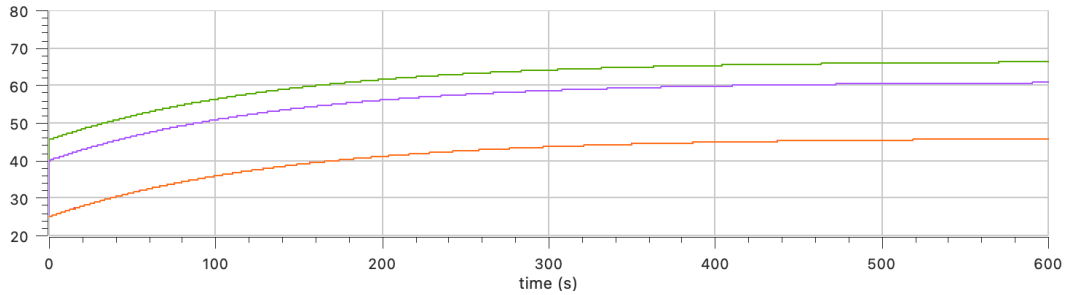
- active silicon capacity $Ca = 0.00012123\,J/K$
- bulk capacity $Cb = 0.01818\,J/K$
- spreader capacity $Cp = 130.93\,J/K$
- active silicon to bulk conductance $Gab = 3.636\,W/K$
- bulk to spreader conductance $Gbp = 1.334\,W/K$
- spreader to external environment $G_{ext} = 0.9522\,W/K$

The values above represent a general 80W microprocessor (hence, each core has 20W maximum power) and were selected so as to recreate the temperature profile obtained in

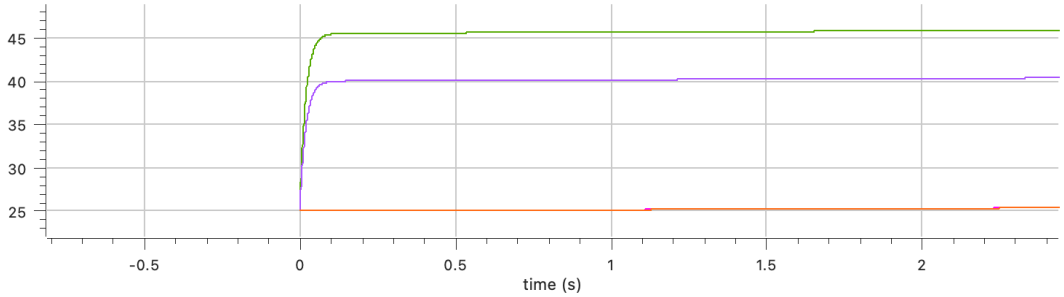
Figure 6.1: Open-loop experiments of the model




full-power excitation of the microprocessor




full-power excitation of a single core




highlighting of the fast dynamics of the single core excitation

the previous work [16] and depicted in figure 4.1.

Figure 6.1 top shows the effect of a full-power excitation of all cores. The orange line is the temperature of the active silicon, the purple line is the temperature of the bulk and the magenta line is the spreader temperature.

Figure 6.1 middle shows conversely the effect of a one-core full-power excitation, while Figure 6.1 bottom depicts a zoom of the above so as to highlight the fast dynamics. The green line is the active silicon temperature, the purple line is the bulk temperature and the orange line is the spreader temperature. The one-core excitation sets one core to the maximum power while the others are turned off and, therefore, their temperature profiles

are equivalent to the one of the spreader. The time constants of the power to active silicon temperature transfer function are

- $\mu = 2.0749$
- $\tau_1 = 0.0036581$
- $\tau_2 = 32.726$
- $T_1 = 3.312\,10^{-5}$
- $T_2 = 0.013717$
- $T_3 = 137.52$

### 6.1.1. Experiments with constant disturbances

The first set of Modelica experiments was performed to investigate the performances of the different tuning and analysis rules, using the robust relay method (Section 2.4) in the presence of constant disturbances. As such only one core is controlled, while the others are subjected to 5W constant power.

The autotuning is requested at 10s, when the controlled core has reached the target temperature of 45 °C. The autotuner high-pass filter was selected according to (2.8), the relay amplitude to 4W, and the sampling time to 5ms. Two different experiments were carried out using the describing function (DF) coupled as tuning policy to the (one-point) closed loop adjustment (CLA) and its contextual version (IMC con). Table 6.1 reports the results.

| | K | Ti | $\|P\|$ | $\mu$ | T | L |
|---|---|---|---|---|---|---|
| DF CLA | 1.2566 | 0.004 | 0.230 | | | |
| DF IMC con | 2.350 | 0.004 | 0.230 | 0.273 | 0.003 | 0 |

Table 6.1: Parameters, data point and model of the system

The obtained oscillation had a period $P = 0.03$. Computing the magnitude of the transfer function evaluated at that oscillation, we obtained that $|G(j0.03)| = 0.2045$. It is important to notice that the describing function identifies the frequency data point around the negative real semiaxis, resulting in not very accurate an identification, as the obtained response is not well represented as a single sinusoid (which the approximation requires).

Nevertheless, the autotuner was still able to operate satisfactorily, enforcing a 45° phase margin.

The autotuning experiment was repeated using the Fourier analysis, but for a correct operation we had to reduce the sampling time to 1 ms and set the hysteresis of the R1 relay equal to $16Ts$. In this way, the relay was forced to wait at least four sampling times before switching, because the integrator had to build up. Indeed, the experiment campaign convinced us that a shorter sampling time (with respect to the describing function case) is a compulsory choice to analyse the transient sufficiently well. The new results – with Fourier analysis – are reported in the following table.

| | K | Ti | $P(0)$ | $\|P\|$ | $\mu$ | T | L |
|---|---|---|---|---|---|---|---|
| FA IMC | 0.2437 | 0.0130 | 1.47051 | 0.5068 | 1.47051 | 0.0130 | 0.0262 |
| FA IMC con | 3.2984 | 0.0331 | 1.5502 | 0.295579 | 1.3282 | 0.0036 | 0 |

Table 6.2: Parameters, data point and model of the system

This time the oscillation period was $T = 0.052$, resulting in the true data point $|G(j0.052)| = 0.2790$. The results clearly show that the Fourier analysis returns better estimation of the data point with respect to the describing function method — however, as noticed, at the cost of a shorter sampling time. It is worth however recalling that this problem is confined to the tuning phase, when for simplicity the event-based approach of the controller to tune – see [16] – is abandoned. After te tuning phase the controller turns back to be event.based, which mitigates the computationally detrimental effects of a "small" sampling time (that becomes the periodic event quantum)

To appreciate the operation of the autotuner also visually, we end this section by reporting some time-domain transients taken from an experiment that was carried out using the Fourier analysis and the IMC contextual tuning rule, with a sampling time of 1 ms. In detail, Figure 6.2 shows the behaviour of the controlled temperature and the control signal during and after a tuning operation (at 10s); after the tuning, a step set point modification (at 12s) and a step disturbance (at 20s) were applied, so as to show the behaviour of the obtained controller.

The figure shows in red the temperature of the controlled core, in blue the temperature of the second and third cores (in fact identical owing to the symmetry of the square 2×2 chip floorplan) and in green the temperature of the fourth core. Regarding the reference step, we chose such a low value (55 °C) to highlight the timeliness of the control (a higher

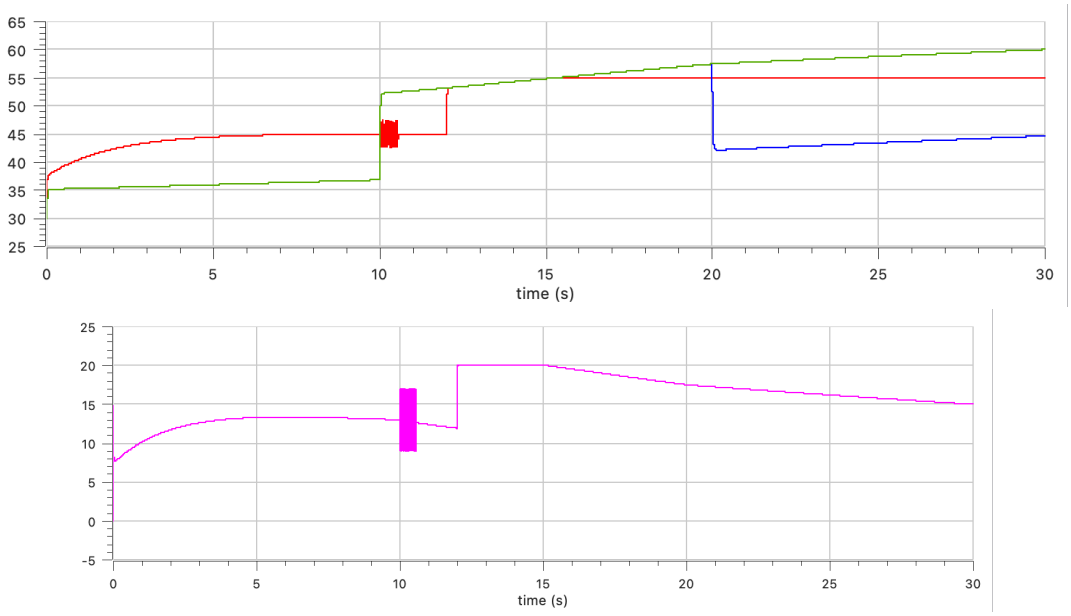



Figure 6.2: Top figure: temperature (°C) of the controlled core in red, the temperature of the second and third core in blue and the fourth core in green. Bottom figure: power profile of the controlled core in magenta.

value would have required waiting for the slow transient of the spreader dynamics) even though in a real control application higher temperature thresholds (70°C or higher) should be set to not throttle the processor. The disturbances consisted of a step change (up to the maximum power) of all the non-controlled cores at the beginning of the experiment, and then of a power drop of the same, except for the fourth one, down to 5 W per core. The former enlightens the disturbance rejection capabilities of the robust relay structure, while the latter shows the readiness of the self-tuned control to disturbances. The effects of the disturbance (at 20s) on the control can be appreciated by looking at the slope of the control variable, which slows down as the total generated power decreases.

As a final consideration of the proposed result, the underlying (much) slower dynamics of the spreader is clearly visible for the entirety of the experiment: since the initial settlement, the controller had to continuously reduce the control action in order to accommodate the set point because the spreader had not yet reached the steady-state temperature.

Based on the obtained results (and others not reported for brevity) we conclude that the describing function and the Fourier analysis are both viable analysis approaches, that the former is less accurate as expected, but also that the latter provides its better accuracy at the cost of an increased number of required points — i.e., of a smaller sampling time (which is computationally detrimental, but only in the tuning phase). We also conclude that none of the proposed tuning policies significantly outperforms the others, hence CLA

can be used if maximum simplicity is a must, while the contextual IMC is preferable in a view of the future extensions envisaged, thanks in particular to its capabilities of providing reliable forecasts of the controlled variable.

### 6.1.2. Experiments with variable disturbances

The second set of Modelica experiments aims to assess the proposed autotuner in a more realistic setting, i.e., when a tuning operation is carried out on a core while the neighbouring ones are dissipating a time-varying power, which is the normal case when they are executing their software tasks.

For a good degree of realism, the power dissipated by the "disturbing" cores is taken from a recorded power track, in turn coming from the execution of a real application. In detail, he chosen power profile was the measured electrical power consumed by a microprocessor running the Cloverleaf mini-application, from the UK Mini-App Consortium, which employs an explicit second-order method for the resolution of compressible Euler equations, a representative application from the HPC domain.

The said power profile was transformed into a load power profile to address both the disturbances rejection capability of the exciter and of the autotuned control. Accordingly, the power signals fed to the microprocessor represent the maximum virtual power of each core, i.e. the power of the core if it were fully utilised, resembling the effects of the modulation by the DVFS, while the effective power generated by the core is the virtual one corrected by the load coefficient (which is always between 0 and 1). Hence, one core only was controlled, using the Fourier analysis and the IMC contextual methods, and a 1ms sampling time, while the others were fed by the maximum virtual power of 20 W. Once again, we subjected the autotuned control to a step change in the reference (time 13.5 s) and a disturbance generated by the other cores (time 20 s).

Figure 6.3 shows the obtained results. During the initial settlement and during the experiment, the load trace of the controlled core was overridden such that it was equal to 1 (it was obtained artificially by enforcing the trace to 1 until the second 11.5 was reached). The maximum (or at least constant) load is a fundamental requirement for the successful execution of the experiment because the said disturbances could nullify the exciter action and the analyser needs to identify the data point in the absence of multiplicative disturbances. Moreover, the constant load is not a compelling requirement and it can be enforced by the use of appropriate routines, such as the `cpuburn` program (a "thermal stresser" designed to just have a CPU consume as much power as possible).

Once the tuning was completed, the setpoint was set to 50W at the second 13.5 to reflect

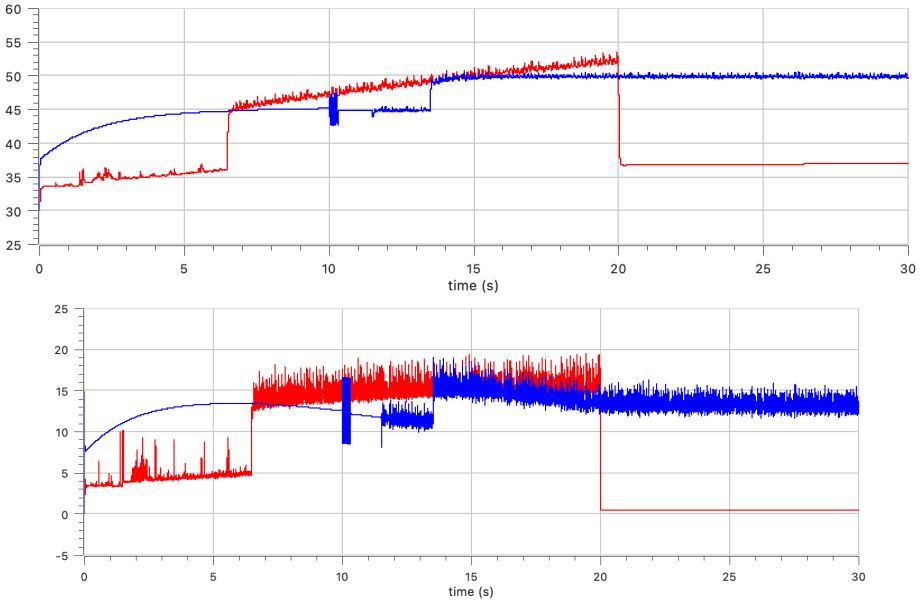



Figure 6.3: Top figure: the temperature profile of the controlled core in blue and the one of the others core in red. Bottom figure: the effective power delivered to the core with the same colour code as just said.

the results obtained in the previous experiment. On the other hand, the disturbance on the control action was obtained by overriding the load profile such that the non-controlled cores had a 0.5 W effective power generation.

In the light of the results obtained, of which only a few were shown, we can say that the autotuning controller is able to handle multiplicative-like load disturbances in a timely fashion and fulfilling the control requirements, also when confronted with power profiles coming from a real application.

## 6.2. A 3D-ICE co-simulation experiment

The aim of these experiments is twofold. On the one hand, we want to verify that the behaviour of the C++ autotuner implementation reasonably matches that of the Modelica one – of course from a qualitative standpoint, as the processor model used in 3D-ICE is different from the 3-capacities Modelica one, and far more accurate as it also includes a detailed representation of the heat sink. The second purpose is to check the co-simulation capabilities of the proposed C++ realisation, so as to assess its viability for the intended integrated studies aimed at a joint verification of heat dissipation equipment and on-chip thermal policies, as envisaged e.g. in [24].

The 3D-ICE library provides examples of the so-called “pluggable heat sink” capabil-

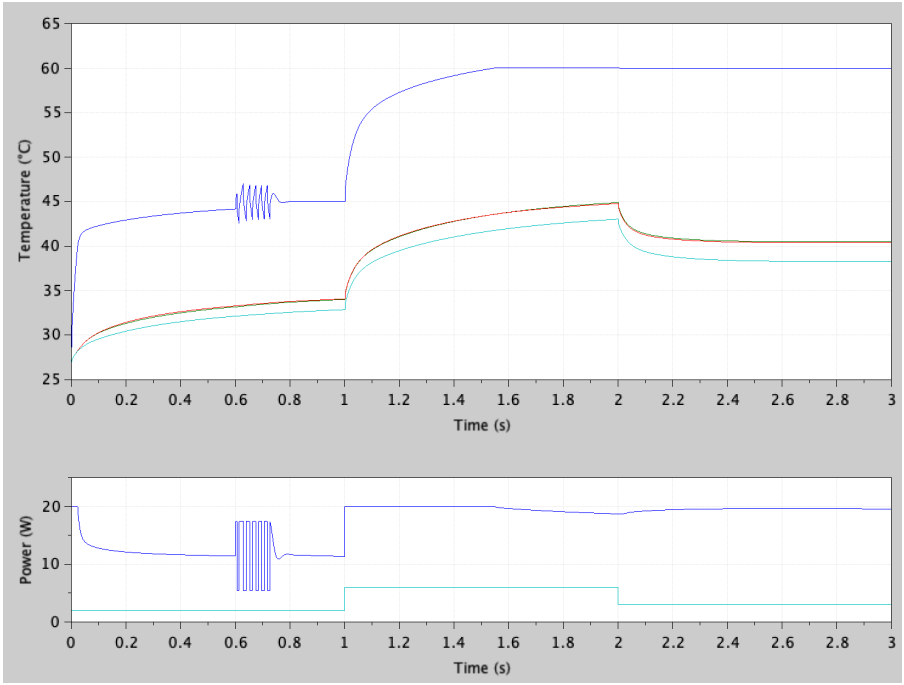



Figure 6.4: Top figure: temperature profile of the core; blu for the controlled (first) core, orange for the second core, green (hidden by the orange line) for the third core and light blue for the fourth core. Bottom figure: power profile of the cores; the non-controlled cores were subjected to the same power profile

ity [23], one of which is the Cuplex water cooling system. We report for compactness only one test, in which an autotuning operation was performed with the Fourier analysis and the IMC contextual method. During this experiment, we had to reduce the sampling time to 100 $\mu s$ because the fast dynamics in 3D-ICE proved to be faster than that obtained in Modelica experiments, owing to the fact that 3D-ICE represents small-scale phenomena that the inherently coarse spatial discretisation of the system-level Modelica chip model smooths out. However, it is important to note that the reduced sampling time is only needed during the experiment in order to correctly estimate the data point; then it can be set to higher values such as 1ms or so — and still, after the tuning, the controller turns back to be event-based. For example, in our test, the autotuner chose an integral time equal to 0.00264s allowing a sampling time smaller than 1.3 ms.

Figure 6.4 shows the obtained results, depicting the controlled variable and the control signal throughout an autotuning operation. A set point step change was applied at 1s and a disturbance was applied at 2s.

The obtained results clearly highlight the disturbance rejection capability of the exciter, showing that the experiment could be performed even during the slower transients of the

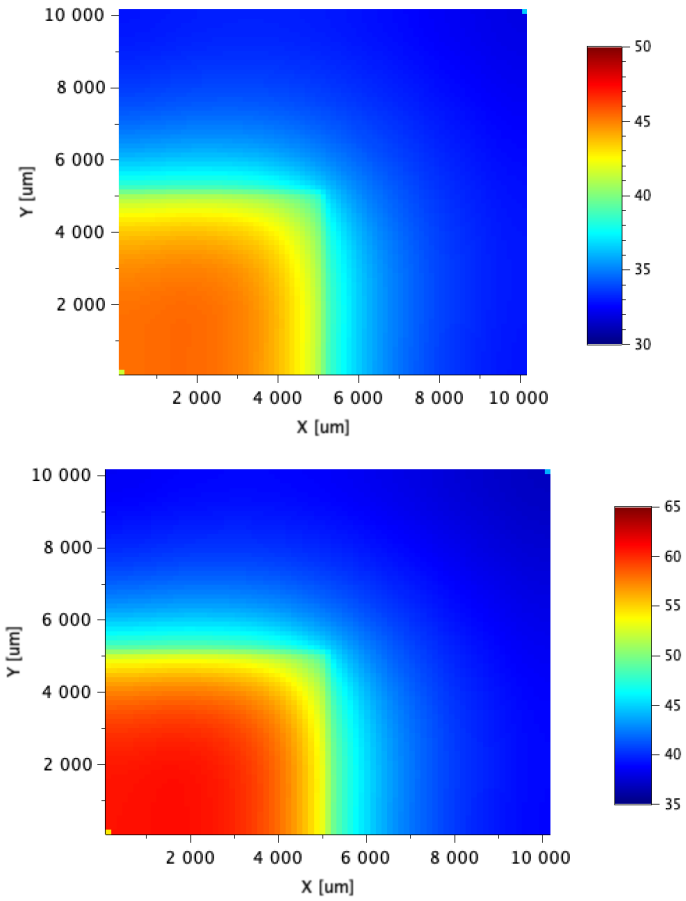



Figure 6.5: Top figure: thermal map of the active silicon before the step change in the setpoint. Bottom figure: thermal map of the core at 3s time instant.

thermal dynamics of the microprocessor.

After the experiment, the setpoint was set to 60 °C (at 1 s) to resemble real applications; in doing so, the slower dynamics can be clearly seen in the temperature response. It is important to notice that the non-controlled cores' power was initially set to 2W each, then it was set to 6W each when the step variation was applied, to facilitate the control action (in this way the slow dynamics effects were partially reduced) and finally, at second 2, the power was set to 3W each to represent the disturbance already discussed in the other experiments.

In addition, to show the added value of co-simulating with 3D-ICE, we show some thermal maps of the chip at various instants during and after the tuning operation

Based on the obtained results, we can conclude that the autotuner is capable to face the control problem it was designed for and, in particular, we can say the following.

- The experiment could be performed in non-steady-state situations, as highlighted

in the last test where the experiment was performed while the controlled core temperature was still arising due to the effects of the other cores.

- The analyser is capable to identify the process data point; it requires a small sampling time, but this requirement is dictated by the nature of the problem (the very small time constant of the fast dynamics), rather than a deficiency of the analysis.
- The autotuner is able to tune itself in a proficient way, devising proper control parameters to correctly face the disturbances to which the controller is subject, as shown in all experiments that were conducted.

Finally, the correspondence of the Modelica and the C++ implementation – in the qualitative sense anticipated, given the inevitably different boundary conditions provided by the two simulation settings – was verified.

# 7 | Conclusions and future work

We presented an autotuning controller conceived for joint power/performance/thermal control in modern microprocessors. The necessity of such controls is nowadays testified by the increasing importance of the "dark silicon" problem, and as quenching thermal stress inherently comes at the cost of reducing performance, the said controls must be capable of limiting that detriment to the minimum required. In turn, given the various installation settings and ambient conditions that a microprocessor can experience, the need for so effective control performances calls for adaptation capabilities. As a result, for the purpose just sketched, autotuning controllers are highly desired.

However, designing such controllers is not an easy task, for several reasons:

- the usage has to be very simple, easy to understand and possibly parameter-free, so that computer (not control) personnel can operate the controller;
- the action on the controlled processor must not be too invasive, so as to not upset the operation of the overall system where the autotuning controller resides;
- the operation must be robust in the face of disturbances from neighboring parts of that system such as adjacent cores, as this is inevitable in any real-world application;
- the resulting algorithm has to be computationally light, so as to make it possible to invoke it e.g. at every system startup.

In this thesis we analysed the problem, also in the light of relevant literature and previous works by the research group where the work was carried out. As a result we came to propose a solution combining several ingredients, namely

- relay-based process stimulation of various types, also including a technique (based on results from the literature) to effectively reject disturbances even during the experiment phase;
- relay data analysis based on the describing function approximation and via Fourier analysis;
- tuning policies based on assigning one point of the open-loop Nyquist curve as

well as on the Internal Model Control principle, including its "contextual" version to allow for reliably forecasting at tuning time the resulting behaviour of the controlled variable.

The proposed solution, thanks to the combination of purpose-specific stimulation, analysis and tuning policies, fulfills all the needs just set forth. After the theoretical motivation and the discussion sketched above in retrospect, it was realised in two forms:

- a Modelica library, targeted to system-level testing,
- and a C++ application, for experimentation with fine-grain chip simulators like 3D-ICE and to run in conjunction with other on-chip policies such as load-based frequency/voltage governors, so as to be assessed as ready for porting on a real device.

Simulation experiments were carried out and analysed using both realisations, to assess their correctness and mutual consistency – reasoning on a qualitative basis, given the different nature of the used processor models, but nonetheless obtaining good correspondence – as well as the capability of the C++ code to integrate with domain-specific accurate Simulation tools such as 3D-ICE.

The autotuner realisation on a physical processor is the first activity planned for the future, together with refinements of the tuning procedure and further assessment in simulation, for example within studies aimed at a joint design, based on virtual prototyping, of heat dissipation equipment and on-chip power/performance/thermal policies.

# Bibliography

[1] URL https://modelica.org.

[2] URL https://www.techpowerup.com/cpu-specs/ryzen-7-5800h.c2368.

[3] URL https://github.com/jbeder/yaml-cpp.

[4] A. Bartolini, M. Cacciari, A. Tilli, and L. Benini. Thermal and energy management of high-performance multicores: Distributed and self-calibrating model-predictive controller. *IEEE Transactions on Parallel and Distributed Systems*, 24(1):170–183, 2013. doi: 10.1109/TPDS.2012.117.

[5] P. Bolzern, R. Scattolini, and N. Schiavoni. *Fondamenti di controlli automatici*. Mc Graw Hill Education, iv edition, 2015.

[6] A. L. da Silva, A. L. del Mestre Martins, and F. G. Moraes. Mapping and migration strategies for thermal management in many-core systems. In *2020 33rd Symposium on Integrated Circuits and Systems Design (SBCCI)*, pages 1–6, 2020. doi: 10.1109/SBCCI50935.2020.9189933.

[7] T. da Silva, Moisés and P. R. Barros. A robust relay feedback structure for processes under disturbances: Analysis and applications. *Journal of Control, Automation and Electrical Systems*, 30:850–863, August 2019.

[8] A. Danowitz, K. Kelley, J. Mao, J. P. Stevenson, and M. Horowitz. Cpu db: Recording microprocessor history. *Commun. ACM*, 55(4):55–63, apr 2012. ISSN 0001-0782. doi: 10.1145/2133806.2133822. URL https://doi.org/10.1145/2133806.2133822.

[9] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger. Power challenges may end the multicore era. *Communication of the ACM*, 56(2):93–102, February 2013.

[10] N. Gomathi and K. Nagalakshmi. Criticality-cognizant energy-efficient task scheduling on heterogeneous multicore processor. *International Journal of Engineering Trends and Technology*, 70(4):203–214, April 2022.

[11] W.-L. Hung, G. Link, Y. Xie, N. Vijaykrishnan, and M. Irwin. Interconnect and thermal-aware floorplanning for 3d microprocessors. In *7th International Symposium on Quality Electronic Design (ISQED'06)*, pages 6 pp.–104, 2006. doi: 10.1109/ISQED.2006.77.

[12] Å. K. J. and H. T. Advanced pid control. *ISA-The Instrumentation: Systems and Automation Society.*, 2006.

[13] J. Kong, S. W. Chung, and K. Skadron. Recent thermal management techniques for microprocessors. *ACM Comput. Surv.*, 44(3), jun 2012. ISSN 0360-0300. doi: 10.1145/2187671.2187675. URL `https://doi.org/10.1145/2187671.2187675`.

[14] J. Lee, J.-S. Kim, J. Byeon, and W. Sung. Relay feedback identification for processes under drift and noisy environments. *AIChE Journal*, 57(7):1809–1816, July 2011.

[15] A. Leva, S. Negro, and A. Vittorio Papadopoulos. Pi/pid autotuning with contextual model parametrisation. *Journal of Process Control*, 20(4):452–463, 2010. ISSN 0959-1524. doi: https://doi.org/10.1016/j.jprocont.2010.01.005. URL `https://www.sciencedirect.com/science/article/pii/S0959152410000260`.

[16] A. Leva, F. Terraneo, I. Giacomello, and W. Fornaciari. Event-based power/performance-aware thermal management for high-density microprocessors. *IEEE Transactions on control systems technology*, 26(2):535–550, March 2018.

[17] M. S. Mohammed, A. A. M. Al-Kubati, N. Paraman, A. A.-H. Ab Rahman, and M. N. Marsono. Dtapo: Dynamic thermal-aware performance optimization for dark silicon many-core systems. *Electronics*, 9(11), 2020. ISSN 2079-9292. doi: 10.3390/electronics9111980. URL `https://www.mdpi.com/2079-9292/9/11/1980`.

[18] S. Moulik. Reset: A real-time scheduler for energy and temperature aware heterogeneous multi-core systems. *Integration*, 77:59–69, 2021. ISSN 0167-9260. doi: https://doi.org/10.1016/j.vlsi.2020.11.012. URL `https://www.sciencedirect.com/science/article/pii/S016792602030300X`.

[19] A. O'dwyer. *Handbook of PI and PID controller tuning rules*. World Scientific, 2009.

[20] A. Raghavan, L. Emurian, L. Shao, M. Papaefthymiou, K. P. Pipe, T. F. Wenisch, and M. M. Martin. Computational sprinting on a hardware/software testbed. *SIGARCH Comput. Archit. News*, 41(1):155–166, mar 2013. ISSN 0163-5964. doi: 10.1145/2490301.2451135. URL `https://doi.org/10.1145/2490301.2451135`.

[21] A. Sridhar, A. Vincenzi, M. Ruggiero, T. Brunschwiler, and D. Atienza. 3d-ice: Fast compact transient thermal modeling for 3d ics with inter-tier liquid cooling. In *2010*

*IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 463–470, 2010. doi: 10.1109/ICCAD.2010.5653749.

[22] S. W. Sung and I.-B. Lee. Enhanced relay feedback method. *Industrial & Engineering Chemistry Research*, 36(12):5526–5530, Dicember 1997.

[23] F. Terraneo, A. Leva, W. Fornaciari, M. Zapater, and D. Atienza. 3d-ice 3.0: efficient nonlinear mpsoc thermal simulation with pluggable heat sink models. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(4): 1062–1075, 2021.

[24] F. Terraneo, A. Leva, W. Fornaciari, and D. Atienza. Modeling and simulation challenges and solutions in cooling systems for nanoscale integrated circuits[feature]. *IEEE Circuits and Systems Magazine*, 23(1):36–56, 2023. doi: 10.1109/MCAS.2023.3234727.

[25] H. Wang, J. Ma, S. X.-D. Tan, C. Zhang, H. Tang, K. Huang, and Z. Zhang. Hierarchical dynamic thermal management method for high-performance many-core microprocessors. *ACM Trans. Des. Autom. Electron. Syst.*, 22(1), aug 2016. ISSN 1084-4309. doi: 10.1145/2891409. URL `https://doi.org/10.1145/2891409`.

[26] C.-C. Yu. *Autotuning of PID Controllers: Relay Feedback Approach*. Springer Science & Business Media, 2013.

# List of Figures

# List of Tables

# 8 | Ringraziamenti

Al termine di questa tesi desidero fortemente ringraziare il professor Alberto Leva, il quale mi ha permesso di intraprendere questo percorso di tesi, onorandomi di suggellare il percorso dell'università con un'esperienza unica e appagante.

Ringrazio il dottore Federico Terraneo, per avermi supportato durante lo sviluppo di questo elaborato.

Desidero dedicare questa tesi ai miei genitori Alessandra ed Antonio, i quali mi hanno sempre supportato in ogni avversità, permettendomi di diventare la persona che sono oggi, e la dedico a mia sorella, Francesca, che pure tra gli screzi comuni tra fratelli, mi è sempre stata accanto, fedele e gentile.

Dedico questa tesi alla mia fidanzata Cristiana, che da 7 anni mi affianca, immancabile, di fronte ad ogni difficoltà, le quali, senza di lei, sarebbero state molto più amare.

Dedico questa tesi ai miei nonni Anna, Rosangela, Gianni e Franco, i quali hanno reso la mia infanzia quel ricordo stupendo che ho la fortuna di avere.

La dedico ai miei amici di vecchia data Angelo, Gabriele, Elenalessandra, Marta e Sarah, che mi hanno sempre regalato esperienze uniche e addolcito quei momenti di sconforto o fatica grazie alla loro amicizia e vicinanza.

La dedico ai miei cari amici Marta e Alessandro, i quali ho la fortuna di avere vicino da ormai i tempi del liceo, di cui il ricordo non sarebbe lo stesso senza di loro, e che continuano a regalarmi momenti stupendi.

La dedico ai miei colleghi universitari Alessandro, Andrea, Gianluca, Roi e Tommaso, perché è solo grazie a loro se il ricordo che ho dell'università è così solare; colleghi e amici con cui ho potuto condividere i dolori e le gioie che l'università è solita dare, sia tra le mura del Politecnico sia al di fuori di esse, instaurando un legame straordinario ed indissolubile su cui, sono sicuro, potrò sempre contare. Grazie a loro il *Poli* non è stato un semplice luogo di studio, ma un punto di ritrovo e cardine della mia esperienza universitaria, gaio e raggiante. Grazie a loro anche una semplice pausa caffè poteva diventare qualcosa di

inaspettato, alleviando il tedio che ogni tanto veniva provocato da certe lezioni. Grazie a loro posso affermare di aver vissuto pienamente e a tutto tondo l'Università e per ciò sono grato a loro.