Alma Mater Studiorum – Università di Bologna

DOTTORATO DI RICERCA IN

Ingegneria Elettronica, delle Telecomunicazioni e Tecnologie dell'Informazione

Ciclo XXXI

Settore Concorsuale:
Area 09 (Ingegneria Industriale e dell'Informazione) – 09/E3 Elettronica

Settore Scientifico Disciplinare:
Area 09 (Ingegneria Industriale e dell'Informazione) – ING-INF/01 Elettronica

POWER AND THERMAL MANAGEMENT RUNTIMES FOR HPC APPLICATIONS IN THE ERA OF EXASCALE COMPUTING

Presentata da: Dott. Daniele Cesarini

Coordinatore Dottorato

Prof. Alessandra Costanzo

Supervisore

Prof. Luca Benini

Co-supervisore

Dr. Andrea Bartolini

Esame finale anno 2019

The computer is incredibly fast, accurate, and stupid.
Man is incredibly slow, inaccurate, and brilliant.
The marriage of the two is a force beyond calculation.

L. Cherne

Abstract

In the scope of technical and scientific computing the rush towards larger simulations, with higher complexity and accuracy, has been so far assisted by a steady downsizing of micro-processing units, which has allowed increasing the compute capacity of general-purpose architectures at constant power. As side effects of the end of Dennard's scaling, this process is now hitting its ultimate power limits and is just about to come to an end. Today is not more possible to maintain the expected grow of supercomputers performance without increase the power consumption. Thus, power and thermal walls stand in front of the evolution of supercomputers towards the exaflops era. These technological walls are big challenges to face for assuring a constant grow of performance in scientific applications. Next generation architectures for high-performance computing systems implement hardware and software components to address power and thermal issues for increasing performance and efficiency. The continuous grow of power consumption at peak performance of supercomputers, start to require to well-defined power budget at design time which should considers the worst-case power consumption of the computing resources to avoid outages. But supercomputers rarely cause the worst-case power consumption during their life limiting the performance achievable in normal conditions. Power capping mechanisms are necessary to aggregate more computing units to target the "average-case power consumption" of the system by limiting the performance at execution time when peak power consumption is demanded. Power capping systems can be implemented using hardware or software mechanisms, but they have different advantages and challenges which need to be explored. Another drawback of the end of the Dennard's scaling is that power density of high-end processors starts to increase at every technologically steps leading to overheating and thermal gradient on the same die. As result, thermal-bound machines show performance degradation and heterogeneity which limits the peak performance of the system. Thermal management solutions are needed in order to avoid thermal hazards which make the system too unreliable to use. Power capping and thermal controls are both safety mechanisms, but none of them address energy efficiency in applications. It is well known that in large application runs, the time spent by the application in the communication is not negligible and impacts the power consumption of the system. Today's supercomputer can leverage on low power states to reduce the power consumption during communication, but due to the time scale at which communication happens, transitioning in low power

states during communication's idle times may introduce overheads. As a matter of fact, there is no known low-overhead mechanism for reducing the power consumption during communication slacks. In this thesis have been developed software strategies in order to face the main bottlenecks of power and thermal issues that affects supercomputers of next generation. It targets scientific applications which are the principal candidates that are negatively affected from the power and thermal constraints of supercomputers. To respond to the above challenges, this work shows that propagating workload requirements from application to the runtime and operating system level is the key to provide efficiency. But this is possible only if the proposed software methodologies have little or no overhead in term of application performance. The run-times libraries developed in this work respect this constraint, and to prove it, they have been largely tested in real production HPC systems with widely used scientific applications and datasets. Moreover, in this thesis has been shown that hardware mechanisms to enforce a power cap at execution time can induce overhead due the unawareness of the application workload. While software policies, which can be integrated in the application, can take advantage on relaxing the power constraint consistently with the application's workload. While, in thermal-bound HPC machines, workload-aware runtimes can leverage hardware knobs to guarantee the best operating points in term of performance and power saving without violating thermal constraints independently for each computing unit. To conclude, it has been proposed a methodology and a runtime for identifying and automatically reducing the power consumption of the computing elements during communication and synchronization primitives filtering out phases which would detriment the time to solution of the application. The experimental results show a significant step forward respects to the current state-of-the-art solutions in power and thermal control of HPC systems.

Contents

List of Figures

Acknowledgements

It is a great pleasure to express my thanks to the many people who contributed and make possible my Ph.D. and this doctoral dissertation.

Firstly, I would like to express my sincere gratitude to my advisor Prof. Luca Benini for the continuous support of my Ph.D study and research's activity, for his motivation, and immense knowledge. His guidance helped me in all the time during my Ph.D. and writing of this thesis. I owe a great debt of gratitude for enlightening me the first glance of research. I could not have imagined having a better advisor and mentor for my Ph.D study.

My special thanks go to the Dr. Andrea Bartolini who have been my co-advisor during these years of doctoral study. Andrea has been a mentor and a friend, from whom I have learnt the vital skills and discipline of academic research. His professional guidance and careful scrutiny of my research work has been invaluable. Thank you again Andrea!

I would like to express my thanks to the dissertation committee: Dr. Carlo Cavazzoni and Prof. Mathieu Luisier, for their insightful comments and appreciations, but also for the hard question which helped me to make this thesis considerably better.

I again want to thank the Prof. Luisier who provided me a great opportunity to join his team at ETH Zurich as Ph.D visiting student, and who gave access to the laboratory, research facilities and financial support.

I would like to thank to all the CINECA staff, in particular to the Dr. Cavazzoni and Dr. Piero Bonfà. Without their precious support and possibility to use the HPC systems of CINECA, it would not be possible to conduct this research.

I thank my fellow labmates for their technical helps and friendship, I have appreciated a lot the fun we have had in these years. I list them in alphabetic order: Alessandro Capotondi, Andrea Borghesi, Andrea Marongiu, Davide Rossi, Federico Pittino, Francesco Beneventi, Francesco Conti, Francesco Paci, Giuseppe Tagliavini, Igor Loi, Manuele Rusci, Marco Balboni, Simone Benatti and Thomas Bridi. Also I

thank my colleagues and friends at ETH Zurich: Abbas Rahimi, Antonio Libri, Antonio Pullini, Daniele Palossi, Frank Gurkaynak, Lukas Cavigelli, Mario Osta, Renzo Andri and Sara Fiore. Also, I thank Angela Cavazzini for the invaluable help to fill-out the bureaucracy documents during these years.

I would also like to acknowledge the support and encouragement of all my friends (in no particular order): Lorenzo Monachini, Mariani's family, Vittorio Muzioli, Davide Medici, Luca Moretti, Denis Billi, Nannuzi's family, Mambelli's family, Re's family, Leonelli's family, Marsili's family, Battagli's family, Federico Ricci, Irene Menghi, Sara Valentini, Marco Lessi, Pablo Montemaggi, Giovanni Campana, Ferrante's family, Mariotti's family, Bertelli's family, Zangoli's families, Mariangela Leone and all the friends of SdC. Moreover, I want to thank my friends of "Meeting di Rimini" that accompanied me during these years: Andrea Battagli, Marco Pacelli, Sandro Ricci, Emanuele Forlani, Matteo Turchi, Eugenio Andreatta, Stefano Pichi Sermolli, Federico Prinetto, Michele Arduini, Stefano Re, Andrea Acquaviva and all the voluntaries.

Last but not the least, I must express my very profound gratitude to all my parents and in particular to my family: My father Massimo, my mother Silvia and my sister Elena for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching. This achievement would not have been possible without you.

Thank you all!
Daniele

Chapter 1

Introduction

1.1 Scientific Applications in the Era of Exascale Computing

In the past four decades scientific applications in many fields have evolved from simple homebrewed codes to complex community projects steadily growing thanks to a distributed, and often loosely coordinated, effort from a variety of groups, laboratories, and institutions. This lively and creative anarchy was facilitated by the smooth progress of hardware architecture, which did not require major disruptions of the programming models. Notwithstanding its benefits, this loosely coordinated development model also had a strong and detrimental impact on the long-term sustainability of software projects, which is becoming particularly critical in the proximity of the next hardware revolution that will occur with the transition to exascale computing. The slowing down of Moore's law and the end of the Dennard's scaling [1, 2] in CMOS technology sets new thermal and power challenges in forthcoming machines which would disrupt the typical programming model adopted so far by scientific community.

1.2 Energy and Power Walls

While Moore's law is approaching its end, Dennard's scaling has already run out of steam. This has caused a constant increase of the power density required to operate each new processor generation at its maximum performance: causing de facto, the total power consumption of each device to limit the practical achievable performance. In addition of the detrimental effect of power density on the final performance, total

power consumption needs to be delivered and removed through cooling consuming additional power. All these issues impact on the total power budget available to the data centres, which limits the computing capacity [3]. As a matter of fact, energy and power walls are the key challenges to be faced in order to deliver the planned performance growth in future.

The effects of energy and power walls are already visible today in the Top500[4] and Green500[5] lists. Until June 2013, every new most powerful supercomputer in the world (1st in the Top500 list) has marked an increase in its power consumption. Reaching with Tianhe-2, 17.8 MWatts of IT peak power consumption, which increases to 24MWatts when considering also the cooling power [6]. This has set the record for the power consumption of a single supercomputer installation, reaching a power provisioning around 20MWatts [7]. The total cost of ownership TCO of a supercomputer is moving from architecture to the power budget, this has set a new practical limit. The fact is that to maintain the performance grown of the past decades, is nowadays possible only increasing the power budget: as a matter of fact, supercomputers are becoming power limited [8]!

1.3 Time Constraints in Power-limited HPC Systems

As it previously seen, supercomputer power consumption is a critical aspect and imposes to target a well-define power budget for the system at design time. The power design must consider the worst-case power consumption of the computing resources of a supercomputer. However, supercomputer workloads rarely cause worst-case power consumption during their life making worst-case design approaches a bad choice which decreases the average supercomputing performance. Power capping approaches support the design for "the average power consumption case" by packing more computing capacity, with a feasible power budget under average workload and dynamically reducing the performance during the execution of workloads with peak power consumption.

At the basis of these approaches there is the capability of computing elements to trade off performance with power consumption. Dynamic and voltage frequency states (DVFS, also known as performance states or P-states [9])) allow the reduction of the power consumption during active computation. Dynamic power management policies take advantage of these HW states to create feedback loops adapting the

performance to workload phases aiming to reduce the energy consumption or ensure a specific power and thermal budget. Pure software implementations of these policies have clear software advantages but need to be executed on the same computational resources, interrupting the application flow and causing overheads.

Recently vendors have added HW components to implement these policies in hardware allowing a more fine-grain control. Intel Running Average Power Limits (RAPL) technology can enforce in hardware a given power limit. This is done exploiting power sensors and power management knobs. Current RAPL implementations aim to enforce a constant power consumption within a ten-milliseconds time window. However this is far below the timescale of supercomputing centres where the power budget must be respected in large time windows coming from the power grid and supplier [10, 11].

1.4 Thermal Capacitance and Heterogeneity in HPC Compute Nodes

Power capping mechanisms are very effective to control the power consumption during supercomputers' life. But the trend of increasing supercomputers' power consumption has also seen as collateral effects of the rapid increase of CPU's power density that in turn have limited the achievable performance caused by higher temperatures. Power capping mechanisms do not guarantee a safe and reliable use of high-performance computing (HPC) nodes from a thermal point of view, which need specific thermal policies and controllers to avoid overheating and thermal hazards.

The maximum safe temperature at which the processing elements of HPC nodes can run depends on the cooling technologies. For instance, Intel Xeon E5-26XX v3 server class processors have specifications on the maximum silicon temperature ranging from 69^oC to 101^oC according to the package thermal resistance (cost) and the nominal thermal design power (TDP)[1]. To enforce a safe working temperatures, HPC nodes use active-cooling solutions which translate in additional power consumption.

Dynamic thermal management (DTM) has been studied to limit the cooling effort by controlling and reducing the heat generation. This is achieved monitoring the HW thermal sensors and the application workload reacting on DVFS states. New

[1]Intel Xeon®Processor E5 v3 Family Thermal Guide

generation multi-core CPUs, which are used in HPC systems, can apply a different voltage and frequency to each core independently [12]. This opens new scenarios for fine-grain DVFS control in DTM solution. Operating systems (OS) use feedback loops between sensors and DVFS states of each core to scale down frequency states to avoid thermal hazards.

Cooling and heat generation are rapidly becoming new key limiters for high performance processors, especially for HPC data centres which typically are hosted clusters of thousands of compute nodes. Each compute node could be equipped with tens of computing units, thus scientific applications can take advantage of these parallel architectures to speed up the execution of large-scale simulations and workloads. The message-passing interface (MPI) programming model is the de facto standard in HPC programs for splitting the workload in tasks that execute in parallel in the compute nodes of the HPC machine. During the execution of an MPI-based application, the tasks alternate phases of computation with phases of communications. A critical design parameter in MPI applications is the balancing of the workload between the tasks, and the minimization of the waiting time for each tasks in the synchronization points [13, 14]. Critical tasks in a specific code segment, are the ones which carry on the most workload and arrive late at a synchronization point. In practice, they limit the application speed in the specific code segment. HPC application developers and users parameterize the application to balance the workload. This intended to limit the slowdown induced by critical tasks. DTM techniques can create local unbalance between cores to maximize processor's throughput. This can be significantly detrimental for application performance as it may slow down critical tasks in parallel applications.

Several solutions explore proactive techniques for DTM strategies to improve performance in thermal-bound systems [15, 16, 17, 18]. DTM strategies take advantages of the heterogeneity in the thermal dissipation of cores, which is related to chip and board design, and manufacture, to maximize the performance. However, these approaches often result in a performance unbalance between the cores. Coldest cores run faster than hottest cores.

1.5 Parallelization and Energy Efficiency

Power capping and thermal control systems are both safety mechanisms to avoid critical situations in HPC machines but none of them impact on the energy efficiency of the application and today the power consumption is one of the most important key limiting factor in HPC systems.

It is known that a significant amount of time in large scale HPC applications is wasted in communication synchronization-related idle times. A typical HPC application is composed of several processes running on a cluster of nodes which exchange messages through a high-bandwidth, low-latency network. These processes can access the network sub-system through the MPI library. Usually when the scale of the application increases, the time spent by the application in the MPI becomes not negligible and impacts the overall power consumption.

Low power design strategies enable computing resources to trade-off performance and power consumption through low power states. These states obtained by DVFS states (P-state), clock gating or throttling states (T-states), and idle power saving states, which switch off unused resources (C-states) [9]. Power states transitions are controlled by hardware policies, OS policies, and with an increasing emphasis in recent years, at user space by the final users [19, 20, 21, 22] and at execution time [23, 24].

While OS policies focus to maximize the utilization of the computing resources — increasing the processor's speed proportionally to the processor's utilization — two main families of power control policies are emerging in scientific computing. The first is based on the assumption that performance penalty can be tolerated to reduce the overall energy consumption [19, 25, 20, 21]. The second is based on the assumption that it is possible to slow down the speed of a processor only when it does not execute critical tasks: to save energy without penalizing application performance [23, 24, 26, 22]. Both approaches are based on the concept of application slack/bottleneck (memory and network communication) that can be opportunistically exploited to reduce power and to save energy. However, there are drawbacks which limit of usage of these concepts in a production environment. The first approach causes overheads in the application time-to-solution (Tts) limiting the supercomputer throughput and capacity. The second approach depends on the capability of predicting the critical tasks in advance with severe performance loss in case of mispredictions.

By default, when processes are waiting in synchronization primitives, the MPI

libraries use a busy-waiting mechanism. However, during MPI primitives the workload is primarily composed of wait times and memory/network accesses for which running an application in a low power mode may result in lower CPU power consumption with limited or even no impact on the execution time.

MPI libraries implement idle-waiting mechanisms, but these are not used in practice to avoid performance penalties caused by the transition times to enter and exit of low-power states [27]. As a matter of fact, there is no known low-overhead and reliable mechanism for reducing selectively the energy consumption during MPI communication slack.

1.6 Contributions and Organization

This thesis faces the power and thermal issues which will affect the next generation of supercomputers leveraging to the typical programming models adopted so far by scientific community. The presented SW approaches avoid to change the source code of scientific applications leaving users and developers to focus only to the usability and on the functionality of the applications. To make this possible, the proposed run-time libraries are able to extract application requirements at execution time and propagating them to the OS and HW level.

In the thesis is shown how i) using SW strategies for relaxing the power capping constraint in HPC systems can have benefit in performance instead to force a constant power consumption through HW mechanisms. ii) it has been developed an optimal thermal control for scientific applications to address thermal capacitance and heterogeneity in thermal-bound HPC nodes, and iii) it has been developed an MPI-aware runtime to save energy in large-scale scientific applications.

This thesis is organized as followed:

- In Chapter 2 are presented the state-of-the-art works in the research fields of power capping systems, dynamic thermal control and power/energy management approaches for scientific computing systems.

- In Chapter 3, it has been shown a novel exploration work on power capping mechanisms for power constrained HPC nodes and it is shown how scientific applications can have performance benefits relaxing the power constraint using large time windows.

- In Chapter 4, it has been developed a an optimal thermal management runtime for scientific applications in order to face thermal hazards and heterogeneity in HPC processors.
- In Chapter 5, it has been developed a run-time library, analysis tool and methodology to save energy in scientific applications by leveraging on the communication slack for real-production HPC systems.
- The conclusion of this thesis are presented in Chapter 6 with a summary of the outcomes.

Chapter 2

State of the Art

2.1 Power Capping

Several approaches in the literature have proposed mechanisms to constrain the power consumption in large-scale computing infrastructures. These can be classified into two main families. Approaches in the first class use predictive models to estimate the power consumed by a job before its execution. At job scheduling time, this information is used to allow into the system jobs that satisfy the total power consumption budget. HW power capping mechanisms like RAPL are used to ensure that the predicted budget is respected during all the application phases and to tolerate prediction errors in the job average power consumption estimation [28, 29, 30]. Approaches in the second class distribute a slice of the total system power budget to each active computing element. The per-compute power budget is ensured to hardware power capping mechanisms like RAPL. The allocation of the power budget to each compute node can be done statically or dynamically [31, 22, 32, 33]. It is the goal of the runtime to trade off power reduction with application performance loss. Eastep et al. propose GEOPM [22], an extensible and plug-in based framework for power management in large parallel systems. GEOPM is an open-source project and it exposes a set of APIs that programmers can insert into applications to combine power management strategies and HPC workload. A plugin of the framework target power constraint systems aiming to speed up the critical path migrating power to the CPU's executing the critical path tasks. Authors in [34] quantitatively evaluated RAPL as a control system in term of stability, accuracy, settling time, overshoot, and efficiency. While the work proposed in Chapter 3 evaluates the proprieties of RAPL mechanism

with respect to a software-based power capping mechanism and shows how vary the performance penalty in different application workload.

2.2 Dynamic Thermal Control

Different works were focused on thermal-aware workload allocation based on DVFS strategies. Those techniques include: (i) on-line optimization policies [35, 36, 37, 38], which are based on predictive models and embedded sensors to read the current temperatures on the system; (ii) scheduling approaches for off-line allocation [39, 40] which rely on simplified thermal models, usually embedded in the target platform [36] or simulating chip temperature [41].

Today's thermal management works range from mobile to large scale parallel machine, like supercomputer and HPC systems. Xie et. al. [42] show that mobile systems are thermally constraint. Interestingly, thermal constraints come from user experience and not from silicon limits. Conficoni et al. [43] show that the power cost of HPC cooling depends on several factors, for instance IT power consumption, the cooling control policies, and the ambient temperature. On the other hand, the power consumption is intertwined with workload execution and computation phases [44, 45], which can produce high thermal heterogeneity between nodes and CPUs. For this reason, over-provisioning cooling design can cause severe inefficiencies.

Wang et. al. [46] show that fan power can account for up to 23% of typical server power and scales super-linearly with node utilization. Authors in [47] extract a predictive thermal model directly from the multicore device correlating power, performance and thermal sensors implemented in hardware. They show that the thermal evolution of a multicore device can be modeled with a linear state-space representation. The leakage-power dependency from temperature can be modeled as a perturbation of the state matrix of the thermal model. Due to different materials present in the heat dissipation path, the thermal transient is multi-modal with time constants that vary from ms to tens of seconds. Beneventi et al. [48] shows in an Intel based computing nodes with 36 physical cores, that the increased number of processors integrated in the same die generates significant thermal gradients and this thermal heterogeneity can be exploited by thermal-aware MPI task allocation to reduce the fan speed and power without impacting the application performance.

To find a close form solution of the fast mapping problem under thermal constraint, Hanumaiah [49] assumes the absence of direct thermal exchange from the hot to the cold cores of the same die. Mutapcic et al. [40] formulate a convex optimization problem to control the speed of the processor, which is subject to environment thermal constraints. They solve it with a specialized algorithm. However, their optimization algorithm does not cover the case of a higher number of cores than the number of tasks (some cores remain in idle state).

Predictive controls are often based on thermal and optimization models which can guarantee a safe-working condition applying performance constraints to the systems. Rudi et al. [50] have developed an integer linear programming (ILP) model for task allocation and frequency selection to avoid thermal hazards in many-core architectures. This thermal control is able to leverage on the idleness of the cores when tasks are less than the number of available cores allocating tasks on the coldest cores and leaving hottest ones in idle states. The limit of [50] is the task allocation, which is not handled by the systems.

2.3 Energy-aware MPI

Several works developed mechanisms and strategies to maximize the energy savings at the expense of performance. These works focus on operating the processors at a reduced frequency for the entire duration of the application [19, 20, 21]. The main drawback of these approaches is the negative impact on the application performance which is detrimental to the data centre cost efficiency.

Fraternali et al. [19, 25] analyzed the impact on frequency selection on a green HPC machine which can lead a significant global energy reduction in real-life applications, but can also induce significant performance penalties. Auweter et al. [20] developed a energy-aware scheduler that relies on a predictive model to predict the wall-time and the power consumption at different frequency levels for each running applications in the system. The scheduler uses this information to select the frequency to apply to all the nodes executing the job to minimize the energy-to-solution allowing unbounded slowdown in the Tts. The main drawback of this approach is the selection of a fixed frequency for the entire application run which can cause a significant penalty on CPU-bound applications. Hsu et al. [21] present a different approach where users

can specify a maximum-allowed performance slowdown for their applications while the proposed power-aware runtime reduces frequency on time windows, respecting the user's specified constraint. For this purpose, the proposed runtime estimates the instruction throughput dependency over frequency and minimizes the frequency while respecting the user's maximum-allowed performance slowdown. Similarly to the previous approach, an energy gain is possible only by degrading the performance of the application.

The main drawback of the works mentioned so far is that they lead to a systematic increase of Tts, which may be acceptable for the user, but it is not easily acceptable by the facility manager, since it reduces the data centre cost efficiency and TCO [51]. For this reason, there is a trend in the literature towards HPC energy reduction methodologies with negligible or low impact on Tts of the running applications.

Sundriyal et al. [52, 53, 54, 55] analyze the impact of fine-grain power management strategies in MVAPICH2 communication primitives, with a focus on send/receive [52], AlltoAll [53], and AllGather communications [54]. In [52] the authors propose an algorithm to lower the P-state of the processor during send and receive primitives. The algorithm dynamically learns the best operating points for the different send and receive calls. In the [54, 53, 55] works, the authors proposes to lower also the T-state during the send-receive, AllGather and AlltoAll primitives as this increases the power savings. In these works, authors discover that the overhead of this solution is more prominent when these communication patterns are intra-node. Moreover, the overhead of the proposed strategy decreases with message dimension. Authors shows that this overhead is different within the different kernels implementing the communication primitives, thus they propose a low power implementation of the studied MVAPICH2 primitives where P-state and T-state are lowered in specific stages of these primitives. These approaches show that power saving can be achieved by entering in a low power mode during specific communication primitives. To be effective, these predicting algorithms needs to be aware of the internal logic of the communication primitive, thus they depend on a specific MPI implementation. Differently, the work in Chapter 5 shows that important savings can be achieve without impacting the implementation of the communication primitives.

Rountree et al. [56] analyzes the energy savings which can be achieved on MPI parallel applications by slowing down the frequencies of processors which are not in the critical path. Authors of the paper define tasks as the region of code between

two MPI communication calls. In this thesis, tasks are refer to phases. The critical path is defined as the chain of the tasks which bounds the application execution time. Indeed, cores executing tasks in the critical path will be the latest ones to reach the MPI synchronization points, forcing the other cores to wait. In [56] authors propose a methodology for estimating offline the minimum frequency at which the waiting cores can execute without affecting the critical path and the Tts.

A later work of the same authors [23], implements an online algorithm to identify the task and the minimum frequency at which it can be executed without worsening the critical path. In a similar way, Kappiah et al. [26] developed Jitter, an online runtime based on the identification of the critical path on the application among compute nodes involved in the application run. Liu et al. [57] use a similar methodology as Kappiah et al. [26] but they apply it to a multi-core CPU.

The authors of [58], as in [56, 23], focus on saving power by entering in a low power state for processes which are not in the critical path. The authors propose an algorithm to save energy by reducing the application unbalance. This is based on measuring the start and end time of each MPI_barrier and MPI_Allreduce primitives to compute the duration of application and MPI code. Based on that the authors propose a feedback loop to lower the P-state and T-state if in previous compute and MPI region the overhead was below a given threshold. The algorithm is based on the assumption that the duration of current application and MPI phases will be the same of the previous ones.

To save energy during MPI phases, Lim et at. [59] propose to reduce core's frequency in "long" MPI phases. Subsequent short MPI phases are grouped together and treated as a single long MPI phase. They use an algorithm to select the best P-state to be applied according to the micro-operation throughput in the MPI phase. Similarly to [56, 23], this approach is based on the assumption that the duration and instruction composition of current MPI phase will be the same as the previous ones. Moreover, by treating short MPI phases as a single long one, the application phases between them is executed at reduced frequency, leading to Tts overheads.

Li et al. [60] use a similar approach to [59] to reduce power consumption in synchronization points. This work focuses on collective barriers for parallel applications in shared-memory multiprocessors. Differently from the previous approaches, instead of using P-state, they use C-state and specific hardware extensions to account for their transitioning (sleep and wake-up) times. As in the previously described approaches,

this runtime uses a history-based prediction model to identify the duration of the next barriers.

The authors of [61] show that the approaches in [56, 23] and the ones which estimate the duration of MPI and communication phases based on a last-value prediction [59, 60] can lead to significant misprediction errors. The authors propose to solve this issue by estimating the duration of the MPI phases with a combination of communication models and empirical observation specialized for the different groups of communication primitives. If this estimated time is long enough, they will reduce the P-state. In Chapter 5, is shown the proposed approach of this thesis and how this can be achieved without a specific library implementation and communication models.

Li et al. [24] analyzed hybrid MPI/OpenMP applications in term of performance and energy saving and developed a power-aware runtime that relies on dynamic concurrency throttling (DCT) and DVFS mechanisms. This runtime uses a combination of a power model and a time predictor for OpenMP phases to select the best cores' frequency when application manifests workload imbalance.

The works in the second group, namely [26, 57, 24, 59, 60], have in common the prediction of future workload imbalances or MPI phases obtained by analyzing previous communication patterns. However, this approach can lead to frequently mispredictions in irregular applications [62] which cause performance penalties. The methodology proposed in Chapter 5, differs from the above approaches (and complements them) because it is purely reactive and does not rely on assumptions and estimation of the future workload unbalance.

The power management literature has analyzed in depth the issue of prediction inaccuracy and predictive model overfitting [63]. One key outcome form of this work proposed in 5, is that timeout-based policies are effective if predictions are not available (e.g. when data is being collected for building a predictive model), and are also essential in mitigating miss-prediction overheads. Timeout-based power management policies are simple to calibrate, as they only require to find a timeout threshold that prevents power state changes when the duration of the slow-down (or shut-down) phase is not long enough to amortize power state transition overhead. In the following Chapters, it is shown how to identify the critical timeout threshold in the specific context of MPI applications running on HPC machines.

In a similar manner, another plugin of GEOPM [22] can selectively reduce the frequency of the processors in specific regions of codes flagged by the user by differ-

entiating regions in CPU, memory, IO, or disk bound. Today, GEOPM is capable of identifying MPI regions and to reduce the frequency based on MPI primitive type. However, it cannot differentiate between short and long MPI and thus cannot control the overhead caused by the frequency changes and runtime in short MPI primitives. The methodology proposed in this thesis addresses this limitation and can be integrated in future releases of GEOPM, as its design principles are completely compatible with it (i.e. no application code modifications are required).

Chapter 3

HPC Power Capping

3.1 Overview

As explained in Section 1.3, leveraging on power capping mechanisms to aggregate more computing units in power-constrained systems adds both advantages and challenges. In this exploration work are evaluated the impact of HW power capping mechanisms on a real scientific application composed by parallel execution. By comparing a HW capping mechanism against static frequency allocation schemes, it can be seen that a speed up can be achieved if the power constraint is enforced in average, during the application run, instead of on short time periods. RAPL, which enforces the power constraint on a few milliseconds time scale, fails on sharing power budget between more demanding and less demanding application phases.

In detail, it has been analyzed the efficacy (measured as application Tts) of power capping to maintain the same power budget using (i) RAPL controller and (ii) a simple fixed frequency allocation that statically assigns cores' frequency for the entire application time. In the first case, RAPL imposes the power constraint every ten milliseconds modulating the operating point (i.e. clock frequency) to workload changes with a similar time granularity. On the contrary, the second set-up uses fixed frequency for the entire application time selected to obtain for the entire duration of the application the same average power consumption as RAPL. This work has been presented to the scientific community in [45].

3.2 HPC Architectures and Power Management Systems

3.2.1 HPC Architectures

A typical HPC system is composed of tens to thousands compute nodes interconnected with a low-latency high-bandwidth network. Compute nodes are usually organized in sub-clusters allocated at execution time from the system scheduler according to the users' request. Sub-clusters have a limited lifetime, after which resources are released to the system scheduler. Users request resources through a batch queue system, where they submit applications jobs. Even a single node can be split in multiple resources shared among users. The single indivisible units in an HPC system are: CPUs, memory and possibly accelerators (GPGPU, FPGA, many-core accelerator, etc.).

HPC applications typically is based on a single program multiple data (SPMD) execution model, where the same executable is instanced multiple time on different computing element of the cluster; each instance works on a partition of the global workload and communicates with other instances to orchestrate subsequent computational steps. For this reason, an HPC application can be seen as the composition of several tasks executed in a distributed environment which exchanges messages among all the instances. Achieving high-performance communication on distributed applications in large clusters is not an easy task. The MPI runtime responds to these demands by abstracting the level of network infrastructure using a simple but high-performance interface for communication which can scale up on thousands of nodes.

HPC machines are extreme energy consumers and the server rooms require a proportioned cooling system to avoid overheating. The extreme working conditions of this machines bring a lot of inefficiencies in terms of energy and thermal control, that turn in computational performance degradation. HW power managers are becoming a fundamental component to control power utilization using different strategies in order to reduce energy waste and, at the same time, assure a safe thermal environment.

3.2.2 Power Management in HPC Systems

Nowadays, operating systems can communicate with different hardware power managers through an open standard interface called advanced configuration and power

interface (ACPI) [9]. This work focuses on ACPI implementation of Intel architectures, since most of the HPC machines are based on Intel CPUs. Intel implements the ACPI specification defining different component states where a CPU can use to reduce power consumption. Today's CPU architectures are composed of multiple processing elements (PE) that can communicate through a on-chip network subsystem. Intel architecture optimizes ACPI using different power saving levels for cores and uncore components. The ACPI standard defines P-state to select DVFS operating points targeting the reduction of active power, while also defines specific idle states to manage power consumption during inactivity time of the CPU. In this work, it has been only considered P-states to manage DVFS control knobs, this because tipycal HPC applications do not manifest idle time during the execution.

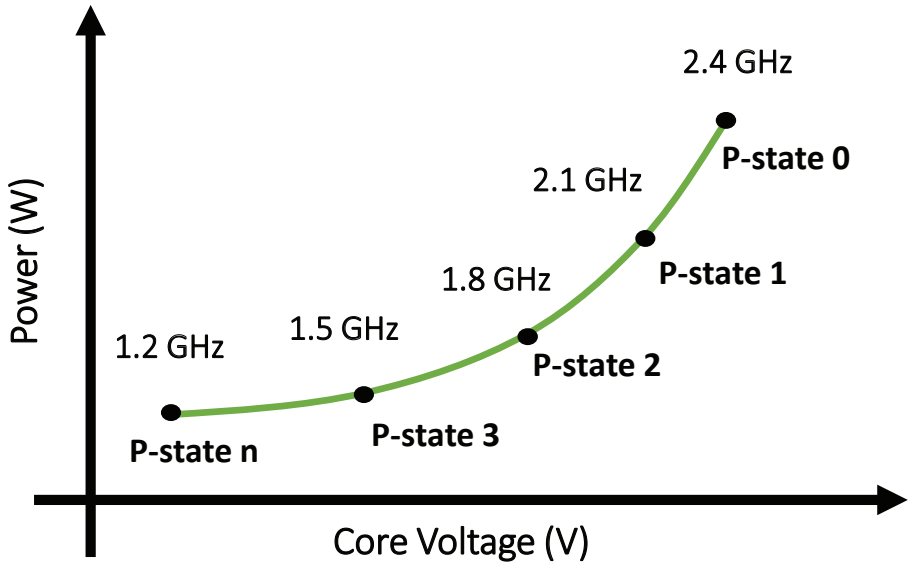


Figure 3.1: Example of how vary the power/voltage respect to the frequency for different P-state level in Intel architectures

Intel P-states show in Figure 3.1 defining a number of levels which are numbered from 0 to n, where n is the lowest frequency and 0 is the highest frequency with the possibility to take advantage of *Turbo Boost technology*. *Turbo Boost* is an Intel technology that enables processors to increase their frequency beyond the nominal via dynamic control of clock rate. The maximum turbo frequency is limited by the power consumption, thermal limits and the number of cores that are currently using turbo frequency. Since Haswell architecture, Intel cores allow independent per-core P-state.

Linux Power Management Driver

Intel P-states are managed by the OS power governor implemented in the Linux kernel driver. By default on Linux system, Intel architectures are managed by a kernel module called *intel_pstate* developed by Intel. But *intel_pstate* driver does not support a governor that allows users to select per-core fixed frequency. Differently, the standard power management driver of Linux *acpi-cpufreq* does it. *acpi-cpufreq* is similar to Intel driver but implement a large set of governors. The available governors are:

1. *powersave*: this governor runs the CPU always at the minimum frequency.
2. *performance*: runs the CPU always at the maximum frequency.
3. *userspace*: runs the CPU at user specified frequencies.
4. *ondemand*: scales the frequency dynamically according to current load [64].
5. *conservative*: similar to *ondemand* but scales the frequency more gradually.

It has been used *acpi-cpufreq* driver with *userspace* governor that allows to select per-core fixed frequency.

3.2.3 Hardware Power Controller

Today's CPU architectures implement reactive HW controller to maintain the processor always under an assigned power budget. The HW controller tries to maximize the overall performance while constraining the power consumption and maintaining a safe silicon temperature. Intel architectures developed its own HW power controller called Running Average Power Limit (RAPL) depicted in Figure 3.2. RAPL is a control system, which receives as input a power limit and a time window. As consequent, RAPL continuously tunes the P-states to ensure that the limit is respected in the specified time window. RAPL can scale down and up cores' frequencies when the power constraint is not respected overriding the selected P-state. RAPL power budget and time window can be configured writing a model-specific register (MSR). Maximum and minimal values for both power budget and time window are specified in a read-only architectural register. Values for both power and time used in RAPL

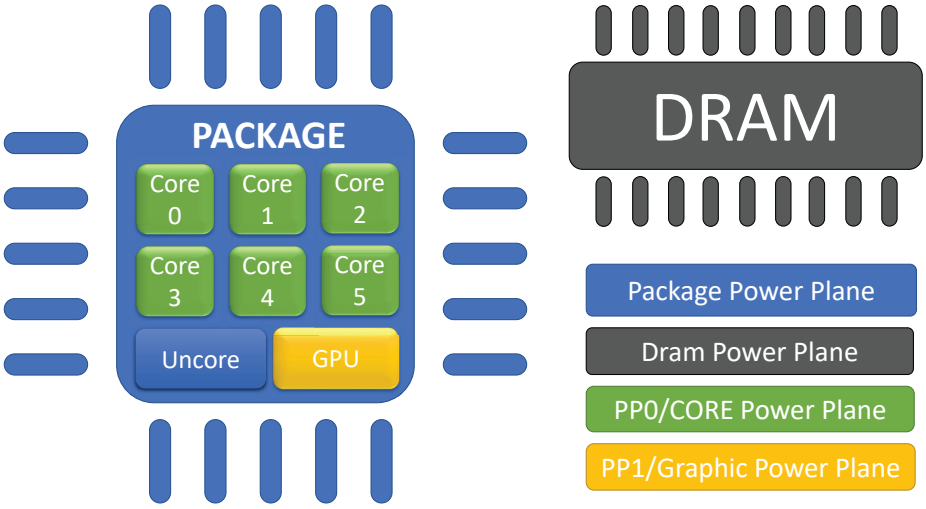


Figure 3.2: Power domain identification of Intel RAPL

are represented as multiple of a reference unit contained in a specific architectural register. At the machine startup, RAPL is configured using maximum power budget of the CPU with a 10ms time window. RAPL also provides 32bit performance counters for each power domain to monitor the energy consumption and the total throttled time. RAPL implements four power domains which can be independently configured:

1. *Package Domain*: this power domain limits the power consumption for the entire package of the CPU, this includes cores and uncore components.

2. *DRAM Domain*: this power domain is used to power cap the DRAM memory. It is available only for server architectures.

3. *PP0/Core Domain*: is used to restrict the power limit only to the cores of the CPU.

4. *PP1/Graphic Domain*: is used to power limit only the graphic component of the CPU. It is available only for client architectures due Intel server architectures do not implement graphic component into the package.

In the experimental result, the exploration focus on the package domain of RAPL controller because core and graphic domains are not available for Intel Xeon processors while DRAM domain is left for future exploration works. Also the time windows of package domain (which can be set in a range of 1ms to 46ms in target system) was

tested to check its impact on application performance. The experimental results show that this parameter does not lead to noticeable changes in the results obtained. For this reason, results report only the default 10ms time window configuration.

3.3 Benchmarking Scenario

3.3.1 Architecture Target

The architecture target used in the experimental Section is a Tier-1 high-performance computing infrastructure based on IBM NeXtScale cluster. Each node of the system is equipped with 2 Intel Haswell E5-2630 v3 CPUs, with 8 cores with 2.4 GHz nominal clock speed and 85W TDP [12]. As regards the software infrastructure, SMP CentOS Linux distribution version 7.0 with kernel 3.10, runs on each node of the system. The experimental tests have been conducted using the complete software stack of Intel systems for HPC production environment. In particular, the Intel *MPI Library 5.1* has been used as communication library and Intel *ICC/IFORT 16.0* has been used as the toolchain. This Tier-1 supercomputer is currently classified in the Top500 supercomputer list [4].The analysis focuses on a single node of the cluster.

3.3.2 Application Target

The thesis work is focused on a widely used scientific application, QuantumESPRESSO (QE) [65]. QE has been selected as principal benchmark because is a paradigmatic suite of scientific applications largely used in several HPC centres. QE shows several architectural bottlenecks of typical HPC applications, like CPU, memory and IO boundness. Moreover, the collaboration with the developers has been a key factor in the correctness of use and in the tuning of the application. The QE version used in the experimental Sections of this thesis is v5.4.0.

In detail, QE is an integrated suite of scientific codes for electronic-structure calculations and materials modelling at the nanoscale. It is an open source package for research in molecule dynamics simulations and it is freely available to researchers around the world under the terms of the GNU General Public License. QE is commonly used in high-end supercomputers and it is based on main computational kernels, which include dense parallel linear algebra (LA) and 3D parallel fast Fourier transform (FFT).

Moreover, most of application workload is based on LA and FFT mathematical kernels, which makes the exploration work relevant for many HPC codes. In the following experimental Section of this Chapter, it has been selected the QE Car-Parrinello (CP) simulation, which prepares an initial configuration of a thermally disordered crystal of chemical element by randomly displacing the atoms from their ideal crystalline positions. The dataset used for this simulation is a silicon atom (SiO2) due it is one of the common chemical element occurs in nature.

3.3.3 Monitoring Framework

In this Section, it is described in detail the monitoring framework used to profile the application and the system. The monitoring framework is composed of two monitoring tools. The first one can monitor several hardware performance counters with a regular time stamping. The second monitoring tool is synchronized with parallel phases of the application allowing to isolate performance and architectural metrics for each program phase. However, due to the higher number of monitoring points per time unit, it access only on a sub-set of performance counters of the first tool to avoid excessive overhead. The monitoring framework have a minimal overhead, less than 1% with respect to application execution time.

System-aware Monitoring Tool

The system-aware monitoring tool is based on Examon [66]. This monitoring tool can be used to read periodically per-core frequency, CPI and scalar/vector instructions retired. In addition, it can monitor for each socket the DRAM memory bandwidth and package power consumption using RAPL performance counters. This monitor is a simple daemon process that can access to the performance counters of the CPU using MSR read/write operations. The daemon starts at a given T_{samp} rate and in the following benchmarks are used T_{samp} of 1 second.

Application-aware Monitoring Runtime

To extract application-based behaviours, it has been developed a monitor runtime to extract system information synchronously with the application flow. The runtime is a simple wrapper of the MPI library where every MPI function has been enclosed

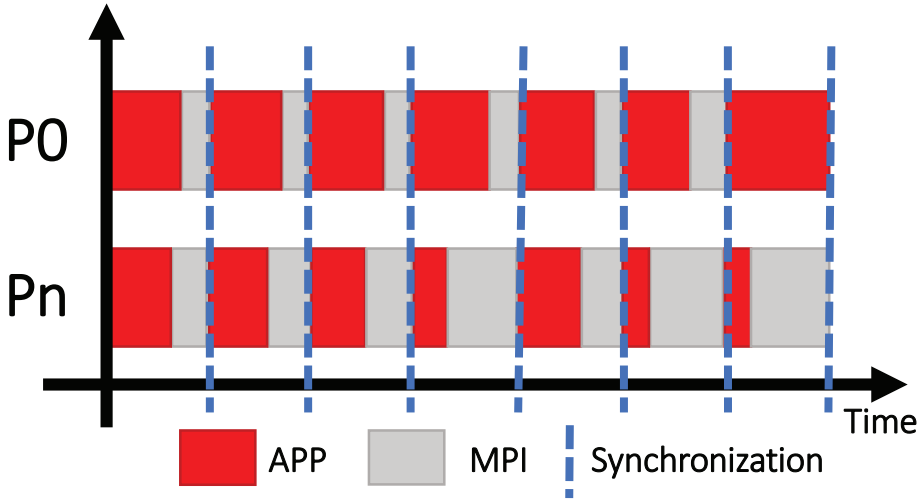


Figure 3.3: Phases of computation and communication identified by application-aware monitoring Runtime

by an epilogue and a prologue function using the MPI standard profiling interface (PMPI). PMPI APIs allow to intercept all the MPI library functions without modify the application source code. The runtime is integrated in the application at linked time. Hence, *Application-aware Monitoring Runtime* is able to extract information distinguishing application and MPI phases as shows in Figure 3.3. The monitor runtime uses *RDPMC* and *RDTSC* assembly instructions to access respectively the time stamp counter (TSC) and Intel performance monitoring unit (PMU) counters with an overhead of few hundreds of cycles for each counter access. PMU counters are programmable through standard MSR operations which require administrative permissions and are costly in terms of access overhead. However, the counter value can be read using the *RDPMC* instruction directly from user space without involving syscalls. To monitor frequency, CPI, and scalar/vector instructions retired has been programmed the per-core PMU registers. The monitor runtime intercept all MPI calls of the application even in application regions where the density of the calls are very high. For this reason is not possible to use MSR operations to access low level performance counters through syscalls, which can cause high-performance penalty.

3.4 Experimental Results

3.4.1 Methodology

The QE-CP has been run with a configuration of 16 MPI processes with a one-to-one bind to each core of the HPC node. This allocation is able to load as much as possible the node. Different configurations are compared for power capping in the test environment. Initially, the power budget has been split in an equal manner on both sockets, RAPL has been set to maintain 48W on each socket, for a global power envelope of 96W. This test shows that the cores' frequency on different sockets are heterogeneous, suggesting that the two sockets have different inherent power efficiency. To have the same frequency among all the cores, the tested computing node needs of 11.31% higher power on socket 0. As consequence of this result, a set of benchmarks has been run using the OS power manager to fix the same frequency for all the cores while monitoring the power consumption of each socket during an application run. These per-socket power budgets are set to power constraints the node to obtain the same frequency among all the cores. The tests has been executed again using RAPL to impose these per-socket power caps and leave RAPL decides the actual frequency.

Table 3.1 shows the results of experimental set using different levels of power caps. In the first column are reported the target frequencies used to extract the power limits specified in the second column. Second and third columns show the sum of power consumption of both sockets using fixed frequency (FF) allocation and RAPL mechanisms for power capping. it can be seen that the power consumption is the same, so the power cap is respected, and the tests are comparable. In the frequency columns are reported the average frequencies for the entire application among all the cores. These columns show that RAPL has an average frequency of 11.14% higher than FF but, it is possible to see at the execution time (reported in next columns), FF has a lower execution time, in average 2.87% faster than RAPL. In the next Sections, the exploration shows why FF has a lower execution time respect to RAPL which, in contrast, has a higher average frequency.

3.4.2 System Analysis

First, It is clearly shown the correct behavior of power capping logic by looking at the core's frequencies and package power consumption (first two top plots). In the

Table 3.1: Quantum ESPRESSO - Power Capping

	Power		Frequency			Execution Time		
	FF	RAPL	FF	RAPL	FF vs RAPL	FF	RAPL	FF vs RAPL
1.5 GHz	$95.56W$	$94.81W$	$1499MHz$	$1766MHz$	-15.11%	$311.43sec$	$328.16sec$	5.10%
1.8 GHz	$111.86W$	$110.63W$	$1797MHz$	$2144MHz$	-16.22%	$274.11sec$	$274.42sec$	0.11%
2.1 GHz	$122.87W$	$120.71W$	$2094MHz$	$2323MHz$	-9.86%	$247.60sec$	$254.59sec$	2.75%
2.4 GHz	$134.44W$	$131.32W$	$2392MHz$	$2476MHz$	-3.37%	$231.19sec$	$239.65sec$	3.53%

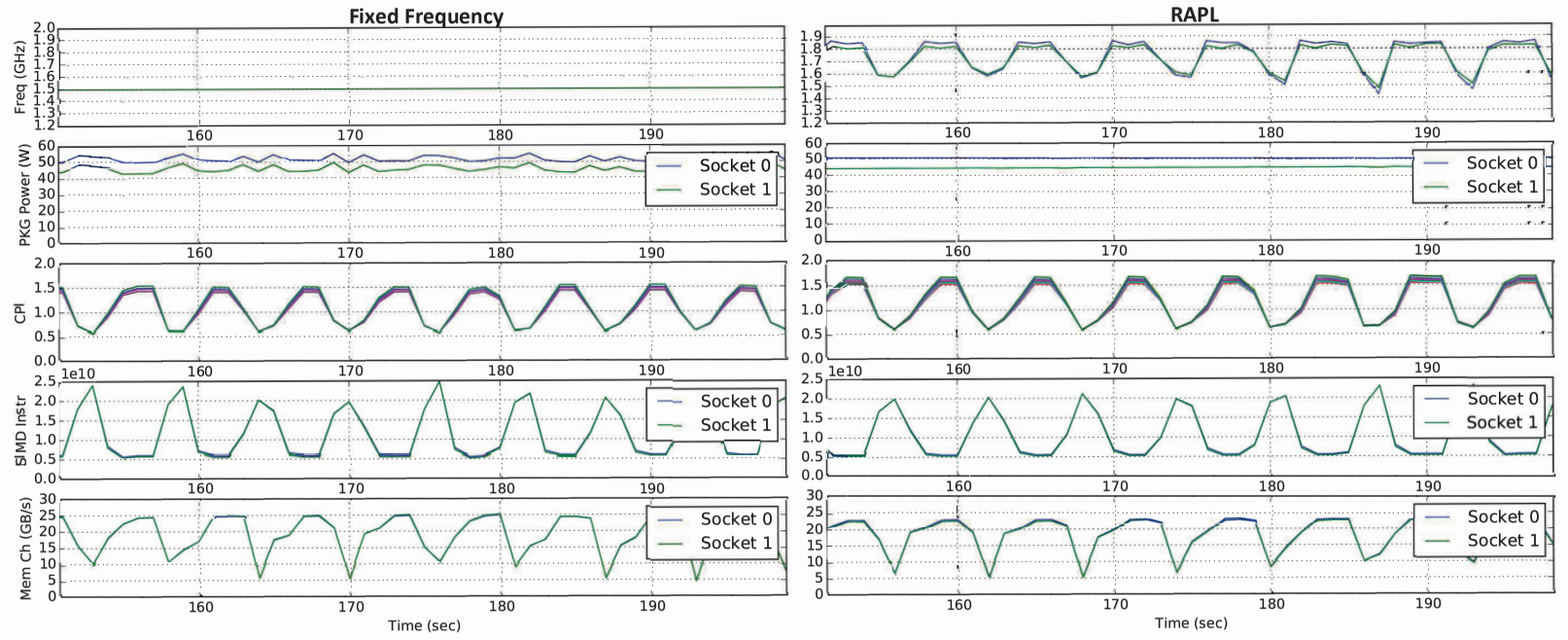


Figure 3.4: Time window of 50 seconds of the system monitor, every value is averaged over 1-second-interval window

FF plot on the left part of Figure 3.4, cores' frequency are fixed at 1.5GHz while package power consumption floats around the average value as effect of the different application phases. In contrast, RAPL (on the right) maintains constant the power consumption for both sockets while cores' frequency changes following the current application phase. Table 3.1 reports a similar average power consumption for both cases, thus the power cappers are working as expected. Both benchmarks show a lower CPI when the memory bandwidth is low and SIMD instructions retired are high. In these phases, RAPL imposes lower core's frequency than the FF case as effect of the higher power demand of SIMD instructions. On the other hand, RAPL assigns higher core's frequency than FF when CPI is high and this happens when the application is moving data from/to memory as proved by the high memory traffic/bandwidth reported by the Mem Ch[GB/s] plot. In these phases, the number of SIMD instructions retired are lower and, as already pointed out and shown in the RAPL plot, the cores' frequency selected by RAPL increases above average due the higher available power budget. However, increasing cores' frequency when the application is memory bound does not reflect in a consequent performance gain due the higher CPI and sub-linear dependency of application speed-up with frequency in these phases.

Figure 3.4 shows a time window of the system-aware monitoring tool for both power capping mechanisms while QE-CP iterates on the same computational kernel. The test reports the case of a power constraint relative to 1.5 GHz for FF and RAPL. So, the results are comparable directly.

Hence, FF is more efficient of RAPL (shorter execution time) for two reasons: i) FF executes with higher instruction per seconds when application has high SIMD instructions density. ii) RAPL instead reduces the core's frequency in the same phase to avoid excessive power consumption. On the contrary, RAPL increases the core's frequency during memory bound phases obtaining a similar average power as the FF case.

3.4.3 Application Analysis

In this Section, it shows what happen in the system through the application-aware monitoring runtime. This runtime is able to recognize application phases marked by global synchronization points as depicted in Figure 3.4. In the Figures 3.5, 3.6, and 3.7 are reported the average values of performance counters of RAPL gathered

into frequency operational intervals. In Figure 3.5 is depicted the amount of time for computation (APP) and for communication (MPI) phases. Figure 3.6 shows the time gain for the FF respect to RAPL distributed on different frequency operational intervals. Negative values mean seconds of application time saved by FF with respect to RAPL. Figure 3.7 shows the values for CPI and SIMD instructions retired for the same phases that RAPL executes at a given frequency.

To explain the behavior that characterizes FF and RAPL, it should be taken a look at all the three plots together. Starting from Figure 3.6, it is possible to recognize that in the frequency intervals 1.3 to 1.5 GHz and 1.7 to 1.9 GHz, FF obtains its highest speed up. The first gaining interval is justified by the high number of SIMD instructions retired and by the lower CPI with respect to other application phases. Indeed from Figure 3.7, it can be noticed that most of the SIMD instructions are executed with these lower frequencies by RAPL.

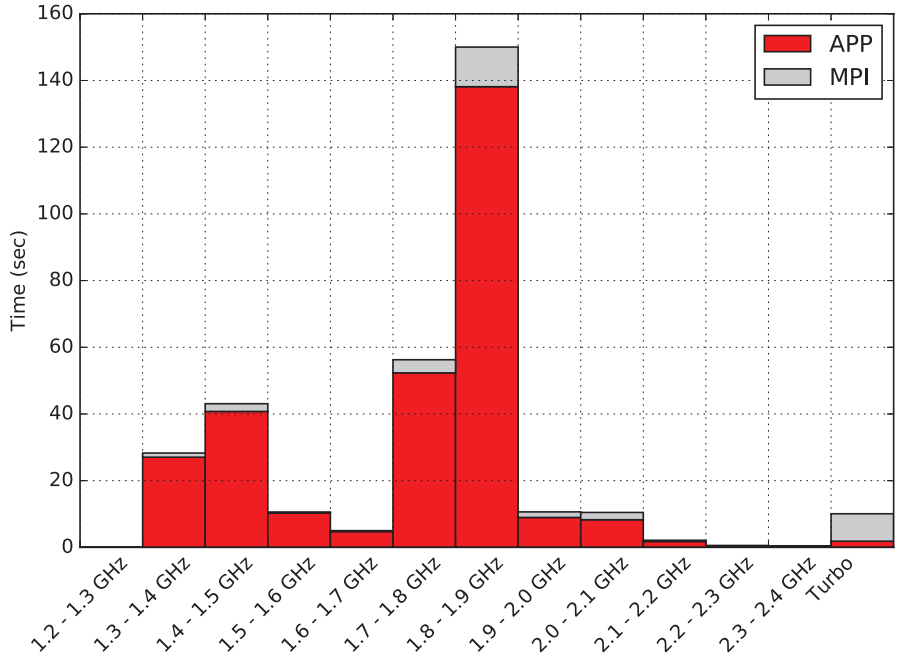


Figure 3.5: Sum of RAPL and application time grouped for frequency operational intervals.

In the interval 1.7 to 1.9GHz, the CPI is higher and the SIMD instructions retired are not negligible. From Figure 3.5, it is possible to recognize that most the application time is spent in this frequency range with a lower SIMD instructions density

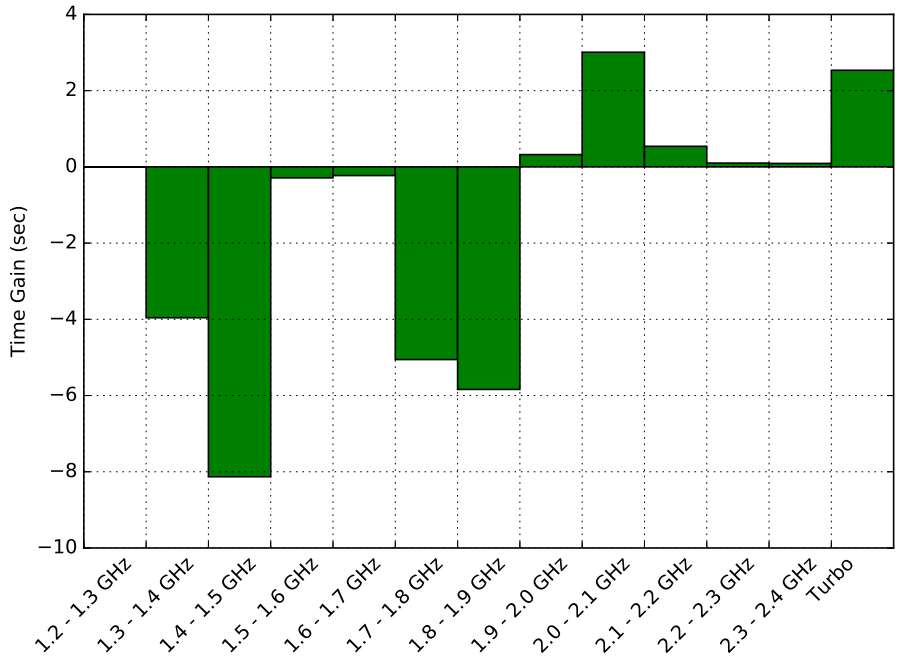


Figure 3.6: Time gain of FF w.r.t RAPL grouped for frequency operational intervals.

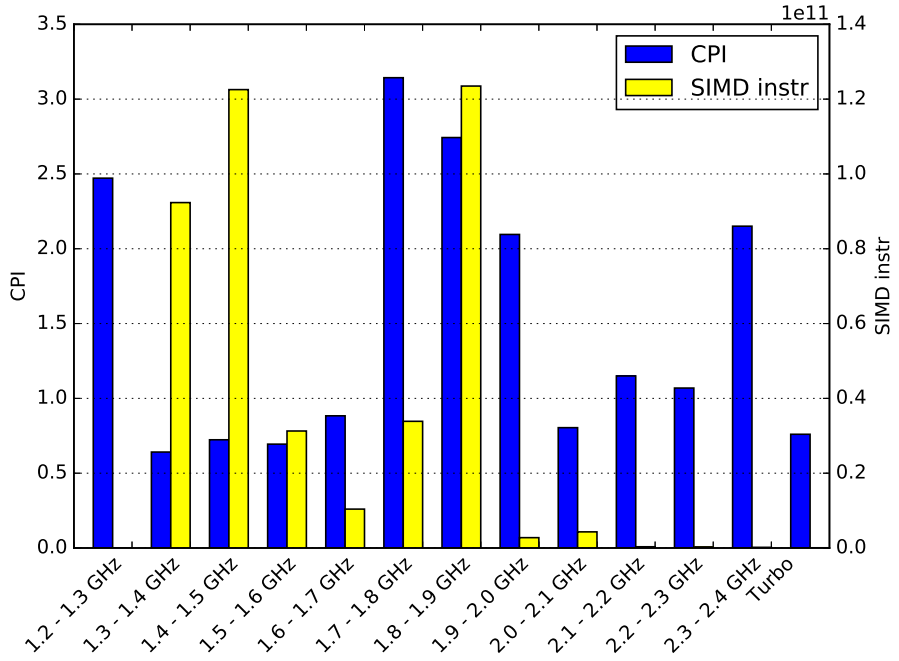


Figure 3.7: Average CPI and number of SIMD instructions retired grouped for frequency operational intervals.

respect to the 1.3 to 1.5 GHz interval. Hence, high CPI, low density of SIMD instructions and high frequency suggest memory bound phases as shown by previous Section. Interesting these phases run at higher frequency than the FF but leads to a performance penalty. This suggests side effects of high frequency in terms of memory contentions.

In the interval 2.0 to 2.1GHz, RAPL has a performance gain with respect to FF. This behavior is explained by the CPI and the number of SIMD instructions retired during this phase. In this interval, RAPL has a low CPI and does not perform SIMD instructions, so this phase scales its execution with the frequency. RAPL can dynamically manages the available power budget made available by the low SIMD instructions and can increases the frequency. This leads to a consequent performance increment.

In the turbo frequency interval, RAPL performs better than FF as it is an MPI reduction phase where only the root process is active. During the reduction, all the processes except the root MPI remain in a barrier to wait the termination of the root. This is explained by the high MPI runtime time (figure 3.5) presents at this frequency interval. Hence, RAPL can use the power budget released by the processes in barriers to speed up the root process leading to a performance gain.

3.5 Summary on HPC Power Capping

In this work, it has been presented a novel exploration on power capping mechanisms for power constrained HPC nodes. Differently from state-of-the-art works, the analysis focus on power capping strategies used in real HPC system nodes. In details, it explored the characteristics of Intel RAPL and a fixed frequency allocation during the execution of a real scientific HPC application which is performance constrained by the power budget assigned to the node.

The exploration work explains why RAPL mechanism has an average performance penalty of 2.87% (up to 5.10%) with respect to the FF case, even if RAPL shows a higher average frequency for the entire application time. This proved that dynamically manage the available power budget without be aware of the application phases is not always beneficial from performance point of view. Furthermore, RAPL mechanism always increases the cores' frequency during less demanding phases to fill in the

available power budget, leading to unnecessary power consumption in memory bound phases that can cause interference and as consequence performance loss.

In future works, it will be possible leverage on these considerations to design a software-based power capping mechanism which enforces a power cap in average on a user defined time-period. This will complement today RAPL mechanisms allowing power shifting in between application phases. For this purpose, the following plan is to extend previously presented approaches, which combine speculative power management on short-time periods (OS ticks) with the ability of maintaining average power management goals on longer periods (aging periods) [67, 68].

Chapter 4

HPC Optimal Thermal Control

4.1 Overview

As presented in Section 1.4, advanced and costly cooling solutions are needed to sustain the high computing densities of exascale HPC systems. To reduce cooling costs and cooling overprovisioning, DTM strategies aim at controlling the device temperature by modulating online the performance of processing elements. While OS allows the migration of threads between cores, in HPC systems the threads of parallel applications are pinned to the cores at start-time to avoid job-migration overheads. In this scenario state-of-the-art DTM solutions, which use thermal models to map jobs to cores, are based on long-term predictions to map the most critical job to the coldest core. Instead, turbo-mode and DVFS controllers are based on short-term predictions to squeeze the thermal capacitance allowing for short period performance boosts which are thermally unsustainable.

In this Chapter is presented a DTM solution for HPC systems to increase performance of thermal-bound HPC machines exploiting thermal capacitance. It is first proposed a novel thermal model description derived from state-space representation of a real HPC node. It is presented a study on the sensitivity of the application walltime to cores' frequency changes in the communication phases. The exploration reveals that the penalty in the application walltime caused by the frequency reduction decreases proportionally with time spent in the MPI library. After that, the focus of this work continue on workload distribution of a real supercomputer's application. Critical tasks are identified and prioritized with respect to the other MPI tasks. Secondly, two novel integer linear programming (ILP) formulations are proposed for thermal-aware

task mapping and frequency selection in large parallel heterogeneous many core. A task criticality model has been developed to intercept application workload and synchronization constraints in order to reduce the slack times. Moreover, it has been done a thermal characteristics of the compute node to formulate both the ILP problems. In this context, the impact of the time horizons is explored at which future temperatures are predicted in the efficacy of the proposed DTM solution. The optimization models shows significantly improve the performance in supercomputer environments without inducing significant overhead in Tts. This work has been presented to the scientific community in [69, 70, 71, 72].

4.2 Workload and Thermal Modelling of HPC

Dynamic thermal management policies aim to reduce the cooling effort by adapting the processing element's performance to ensure a safe working temperature. First, it is introduced the nomenclature and the thermal properties of HPC nodes with direct measurements. Then, a model from real scientific parallel workload has been extracted linking the performance knob to the real performance of the final application. Finally, it has been analyzed how the application workload is distributed among all the cores.

The target machine of this work is the same presented in the Section 3.3.1.

4.2.1 Power and Thermal Management

The power management states of computing elements are divided in sub-groups. As previously explained in Section 3.2.2, the P-states include DVFS operating points which target the reduction of active power. Instead, C-states target idle power reduction strategies. Both P-states and C-states are numbered from 0 to n. Higher number means higher power saving. However, in case of C-state this means also longer transitions in and out of the state itself. In recent Intel architectures [27] P-states can be selected independently for each core. C-states are instead defined independently for cores and the "uncore" region.

C-states are triggered from the firmware of the CPU but the OS can provide hints on the appropriate C-state to the hardware through OS idle governors. Table 4.1 shows the different architecture impacts and dependencies for the different core and package C-states.

Table 4.1: **C-states**

		Core C-states				
		C0	C1	C3	C6	
Package C-states	C0					Active State
	C1E					Lower P-state
	C2					Only L3 Snoop
	C3					Flush L3 - Off
	C6					Low Voltage
		Active State	Clock Gated	Flush L1,L2 Off	Power Gated	

Light gray cells represent a valid configuration for core and package C-states, dark gray cells are invalid ones.

4.2.2 Thermal Model

To understand the thermal properties of a computing node, three main stress tests has been executed using the following methodology: (i) Kept the system in idle and measured the total power and the temperature for each core after ten minutes; (ii) Then, executed a stressmark[1] in sequence on each core of each socket in the node, leaving idle the remaining ones. The workload has been maintained constant for ten minutes and it has been measured the power consumption and the temperature, this test has been used to extract the maximum steady state temperature gradient. Finally, (iii) the stressmark has been simultaneously executed for ten minutes in all the cores of the node and measured the temperature and the power consumption. In all the previous tests the temperature and power values are measured using an infrastructure similar to the one presented in [73], the *Turbo Boost* was disabled to avoid power consumption to workload dependency. The results of the analysis are reported in table 4.2.

The extracted characteristics have been used to create a thermal model using a distributed resistor capacitor (RC) approach [36], with one tuned RC per core to have similar thermal characteristics as the measured ones.

[1] Cpuburn stressmark by Robert Redelmeier: it is a single-threaded application which takes advantage of the superscalar architecture to load the CPU

Table 4.2: Thermal Model

AVG temperature - Idle cores	15.93^oC
AVG temperature - Active cores	33.39^oC
Gradient - Idle cores	4.47^oC
Gradient - Active cores	4.79^oC
Gradient - Active core vs idle cores	8.05^oC
Stady-state time	$120sec$

4.2.3 Power Model

To model the impact of DVFS states on the target system, the stressmark has been executed in each core while scaling down the frequency for each core in all the available speed steps. Each configuration has been maintained for ten minutes and the power consumption has been monitored for each CPU. All these measurements have been collected in a lookup-tables (LUTs), one for each CPU. Then, the LUTs has been used to compute the power dissipated by each CPU on each available frequency. A total power of 17.86W has been measured when all cores in a computing node are idle. The total power raises to 92.44W when all the cores are active. Then, the power consumed has been extracted by each core for each DVFS level with an average standard deviation in between cores of 0.1W. The average uncore region of the CPUs contributes for 11.84W and 17.85W respectively when idle or active. The Figure 4.1 shows the average power consumption for each core of the system at all available frequency levels.

4.2.4 Workload Model

HPC communications happen by sending explicit messages through a standard MPI programming model which takes advantage of the high-performance interconnect subsystem. HPC tasks are composed by computationally intensive phases on independent data segments interrupted by synchronization points and communications. This characteristic impacts the sensitivity of the application to each core's performance as computational imbalance can lead to longer synchronization phases.

As support to this statement, the same benchmark presented in Section 3.3.2 has been used, QE [65]. The following experiment have explored how the different ratio

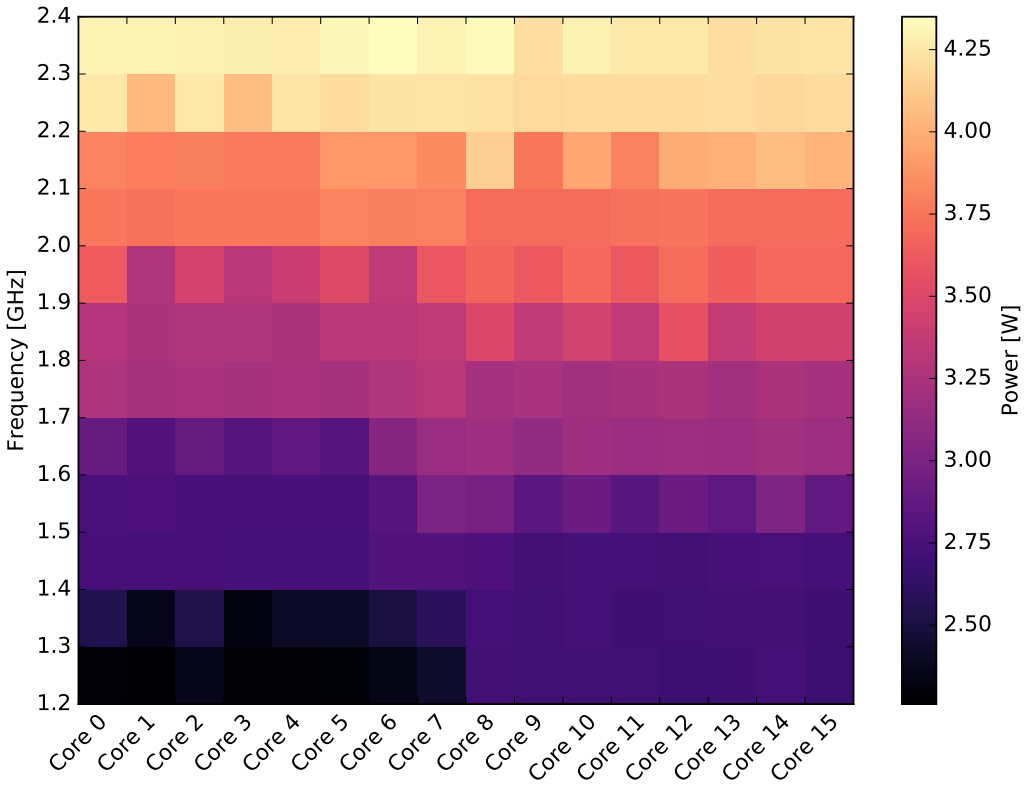


Figure 4.1: Average power consumption of cores at all available frequency levels.

of active code and MPI library for each QE task changes the impact of frequency scaling on the overall application execution time. QE-CP has been executed on two computing nodes with 32 MPI tasks to increase the number of results respects to a run on a single node and it has been run for 32 times. At each run, one core of the 32 has been sequentially configured at minimum frequency while the other cores was maintained at the maximum. The results has been compared with the run in which all the cores are at the nominal frequency. Then, the overall QE-CP slow-down has been compared with the MPI percentage of the slowed down task. Figure 4.2 shows that the impact of frequency reduction increases with percentage of MPI library present in each task. This information has been used for extracting on-line the sensitivity to frequency for each MPI task and it will be used to address energy saving at execution time.

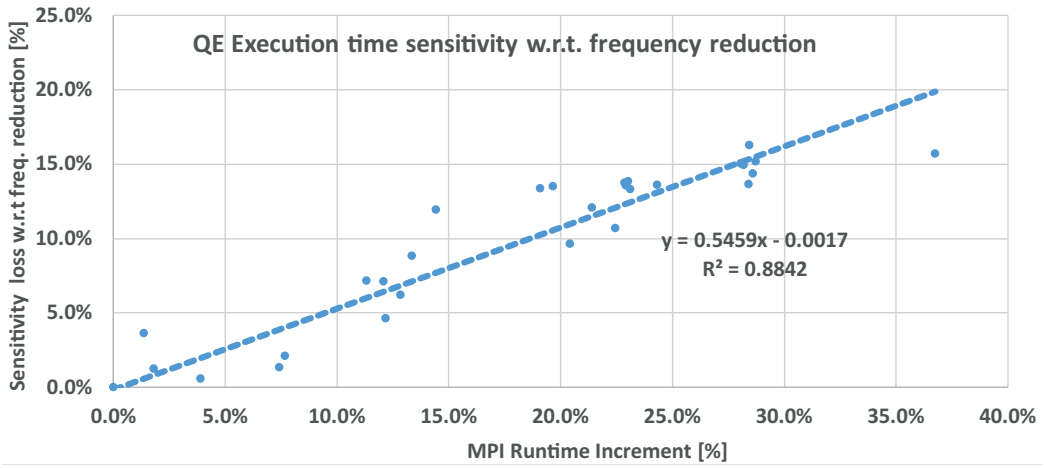


Figure 4.2: Sensitivity loss with respect to the reduction of frequency compared with the increment of the time spent into MPI library

Workload Distribution

While Figure 4.2 shows the workload unbalance for the entire application run, it does not show how this unbalance is distributed in time -at a finer granularity-. In this Section is explored how the workload is spread among all the MPI tasks and in time. QE-CP has been computed on a single compute node with 16 MPI tasks. For each MPI task, the time spent in the application has been extracted and compared with the time spent in the MPI library. Every 10 seconds, the ratio between application time and MPI time has been calculated. The plot with this result can be seen in Figure 4.3. First of all, It can be seen that the MPI task 0 spends more time in the application with respects to the others. In the benchmark, the core that mostly slows down the application execution is the core that host the MPI task 0. If this core was slowed down, the penalty in the total execution time would be the highest one.

In the next Section is explained how this information can be extracted and used in the thermal management problem.

4.3 Optimal Thermal Control

In this Section, the dynamic and thermal management ILP formulation is presented, namely the optimal thermal controller (OTC). OTC matches all the requirements of a proactive thermal control for HPC systems: (i) limiting the future temperature of all

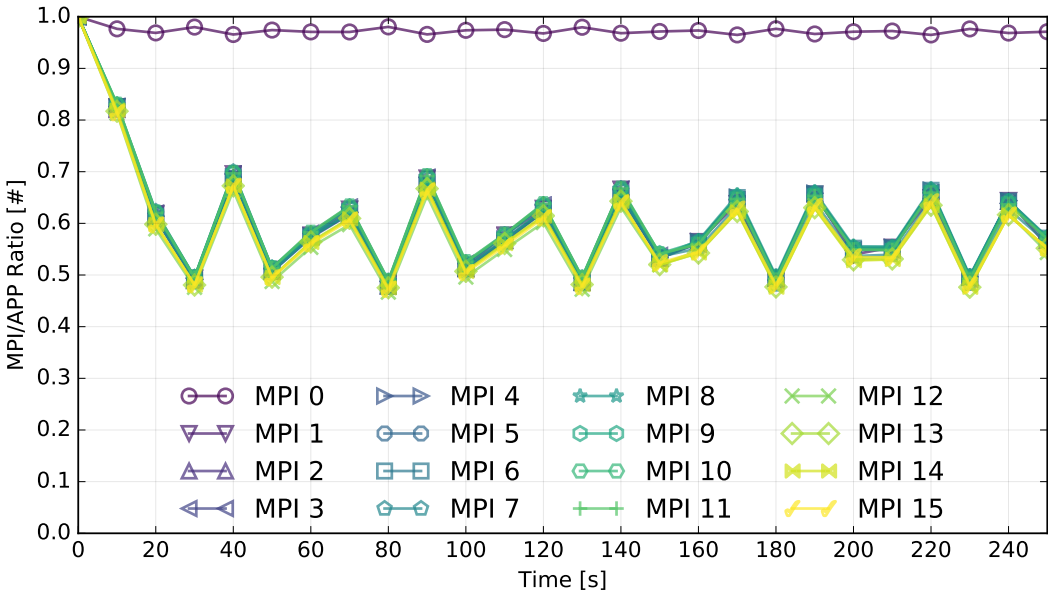


Figure 4.3: Ratio of the time spent in application phases and MPI phases for each core and every 10 seconds.

the cores below a critical threshold by selecting the proper frequency for each core; (ii) maximizing the application performance (frequency of all the cores); (iii) identifying cores that host critical tasks to promote their performance; (iv) slowing down the cores' frequency during communication.

As shown in Figure 4.4, the OTC operates at node level and it is composed of two main components: the thermal-aware task mapper and controller and the energy-aware MPI wrapper.

The thermal-aware task mapper and controller (TMC) is triggered: (a) after the job scheduler has deployed the parallel application on the reserved portion of the HPC machine for its execution; (b) periodically, with period T_s, and (c) at the start/end of every MPI call. At scheduling point (a) the TMC specifies the task to core mapping which will be maintained until the application completion. Clearly, if a critical task is mapped to a thermally inefficient core, will more likely cause a severe degradation of the final application performance. To capture the task criticality, a criticality generation module has been used to intercepts every MPI call extracting the time spent in both application and MPI library. At every scheduling point, this runtime uses timestamps of the MPI calls to identify the criticality level (later named task criticality) for each task, as will be described in 4.3.1. At scheduling point (b), the TMC

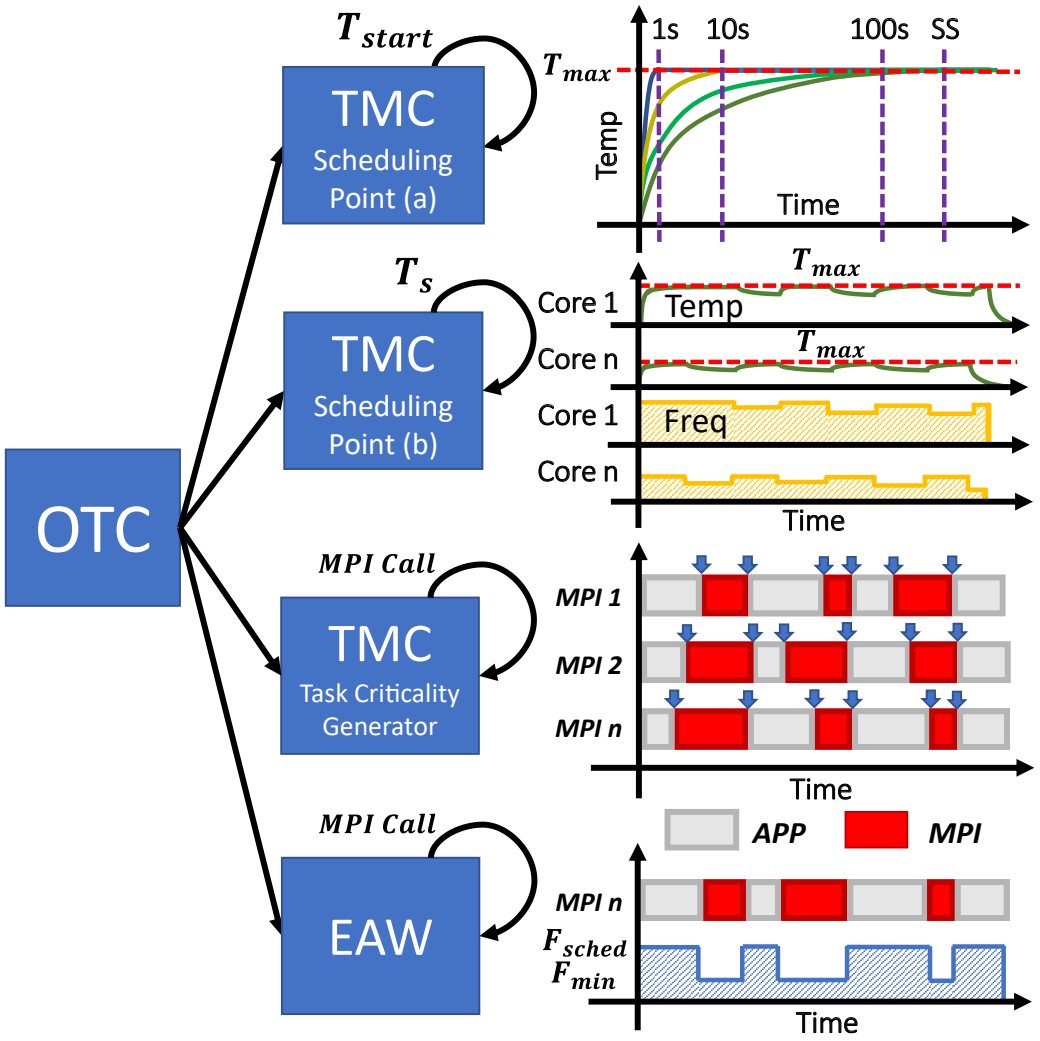


Figure 4.4: Optimal thermal controller at node level

selects the optimal frequencies to be applied to the different cores for the following interval (to maintain the future cores' temperature below a safe threshold). The OTC solution solves the scheduling points (a) and (b) with an ILP formulation and custom solver strategies as described in 4.3.2 and 4.3.3.

The energy-aware MPI wrapper (EAW) is event-driven mechanism that acts as a bridge between the MPI synchronization primitives and the core's frequency selection. This programming model interface is reactive and reduces the core's frequency when the MPI library is in a busy waiting mode. When the execution flow returns to the application code, the frequency is restored to the one selected by the thermal

controller.

4.3.1 Task Criticality Generator

The per-task criticality level is calculated based on the time spent by the task in the application and waiting in the global synchronization points for each interval time. It is not enough to consider only the total time spent in the application during the last interval time to compute a criticality level. It needs to consider each global synchronization point independently and for each of them compute the waiting time of each task.

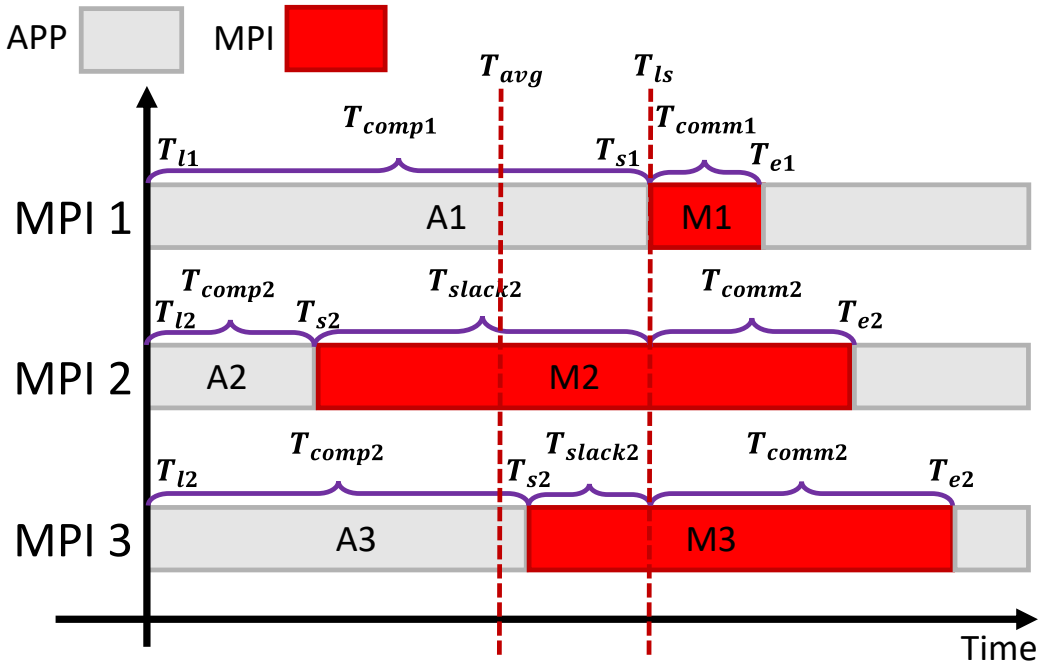


Figure 4.5: General HPC application section with the naming convention used to calculate the criticality for each MPI task.

The per-task criticality level between two global synchronization points has been extracted to calculate the criticality of each task for all the global synchronization points in a single interval. The criticality level has been defined for each task in this interval time like the average of the criticality levels weighted by the time which lasts between each pair of global synchronization points.

Figure 4.5 shows a general HPC application section enclosed by two global synchronization points where all the MPI tasks are involved. Every time that an MPI task

encounters a global synchronization point, it must wait all other tasks to continue its execution. For each task has been identified three key points: T_l, T_s, and T_e which represent the exit time of the last MPI call, the start time of the current MPI call, and the exit time of the current MPI call respectively. $[i]$ has been used as the index to identify the MPI task id.

$$T_{ls} = MAX(T_{s[i]}) \tag{4.1}$$

$$T_{comp[i]} = T_{s[i]} - T_{l[i]} \tag{4.2}$$

$$T_{slack[i]} = T_{ls} - T_{s[i]} \tag{4.3}$$

$$T_{comm[i]} = T_{e[i]} - T_{ls} \tag{4.4}$$

$$T_{avg} = AVG(T_{s[i]}) \tag{4.5}$$

$$\delta_i = \frac{T_{comp[i]}}{T_{avg} - T_{l[i]}} = \frac{T_{s[i]} - T_{l[1]}}{T_{avg} - T_{l[i]}} \tag{4.6}$$

The last task that enters in the global synchronization point unlocks all the waiting tasks which can now continue their execution. T_{ls} in eq. (4.1), identifies the time at which the last task enters in the synchronization point. For each application section and for each task $[i]$, has been defined a computation time $T_{comp[i]}$ in eq. (4.2) as the time spent in the application code and a MPI time, which is the time spent in the MPI library. The MPI time is composed by two factors: (i) $T_{slack[i]}$ in eq. (4.3), which represents the time that a task spends in the MPI library waiting the last task reaching the synchronization point, (ii) $T_{comm[i]}$ in eq. (4.4), which identifies the time spent to copy data. T_{avg} in eq. (4.5) is the average of all the $T_{comp[i]}$. The task criticality level δ_i in eq. (4.6) has been calculated as the ratio between the $T_{comp[i]}$ and the T_{avg}. This metrics is proportional to the unbalance between the tasks in each application section.

4.3.2 The First Step Problem - FSP

This optimization problem is solved during the initialization of the application (FSP). Its purpose is to allocate the application tasks on the available cores and selecting for each of them the maximum frequency which meets the thermal constraint T_{max} in the prediction interval (PI_{FSP}). As it will be seen in the experimental results,

the prediction interval (i.e. the time horizon) plays an important role. Indeed, if it is too short, the TMC cannot predict the impact of a task allocation on long term core's temperature as its effect is hidden by the thermal capacitance, making the problem trivial. On the contrary if the time horizon is too long the TMC cannot take advantage of the thermal capacitance for sustaining short time power burst.

In addition, not all tasks have the same criticality. This is captured by the optimization model which maximizes the frequency of the highest critical task penalizing the frequencies of other ones in case a thermal limit is reached. The optimization model considers K tasks to be assigned to N cores where the number of tasks is lower or equal to the cores i.e., $K \leq N$. Each core can be configured with a frequency in a set of M level of frequencies. The Objective Function ($O.F.$) maximizes the sum of frequencies of all active cores γ_{jf} weighted by the criticality δ_i of the task assigned on that core.The problem has been modeled use two sets of binary decision variables, an ILP model with three constraints has been formulated to model the assignments and the thermal bounds:

$$x_{jf}^{i} = \begin{cases} 1 & \text{if core } j(j = 1, \ldots, N) \text{ works at frequency} \\ & f(f = 1, \ldots, M) \text{ executing task } i(i = 1, \ldots, K) \\ 0 & \text{otherwise.} \end{cases} \tag{4.7}$$

$$y_j = \begin{cases} 1 & \text{if core } j(j = 1, \ldots, N) \text{ is idle,} \\ 0 & \text{otherwise, i.e., if it is working.} \end{cases} \tag{4.8}$$

$$O.F. = max \sum_{i=1}^{K} \sum_{f=1}^{M} \sum_{j=1}^{N} \delta_i \gamma_{jf} x_{jf}^i \tag{4.9}$$

$$\sum_{j=1}^{N} \sum_{f=1}^{M} x_{jf}^i = 1 \tag{4.10}$$

$$(i = 1, \ldots, K)$$

$$\sum_{i=1}^{K} \sum_{f=1}^{M} x_{jf}^i + y_j = 1 \tag{4.11}$$

$$(j = 1, \ldots, N)$$

$$\sum_{j=1}^{N} GS_{jl} \left(\vec{p}_j y_j + \sum_{i=1}^{K} \sum_{f=1}^{M} p_{jf} x_{jf}^i \right) + T_l^0 + T^a \leq T_{MAX} \tag{4.12}$$

$$(l = 1, \ldots, N)$$

The constraint (4.10) specifies that a task must be assigned only on a single core, which works at a given frequency. In addition, it specifies that all the N tasks must be assigned. Constraint (4.11) is needed to determine the y decision variables which represent the idle cores. These variables are used in constraint (4.12) in case there are less tasks than cores i.e., $K \leq M$n. Finally, constraints (4.12) guarantee that the temperature of each core does not exceed T_{max} over the prediction interval (PI_{FSP}). In the last constraint (4.12), GS is a gain matrix with dimension $N \times N$. This matrix is used to calculate the increment of temperature of all the cores when a core is subjected to a constant power input for PI_{FSP} seconds. T_0^l represents the dependency of the future temperature(@ PI_{FSP}) from the current core's temperature. These values can be derived from a state-space thermal model as described by [50]. T_a is the ambient temperature. When tasks are less than cores the decision variable y_i is used in conjunction with the vector of idle powers $\bar{p}$, to add the idle power components.

4.3.3 The i-th Step Problem - ISP

After the tasks have been assigned to the cores in the FSP, the TMC has to periodically solve, at a finer time scale, the assignment problem of frequencies to cores. The ISP has the same objective function as FSP 4.3.2 as well as the same thermal model

formulation. However, the prediction interval for the ISP (PI_{ISP}) can be generally different from the FSP.

Differently from the previous case, the model considers only active cores (T), because the thermal constraints cannot be broken by any idle core. This reduces the overall complexity. As tasks have been already allocated in FSP in this model, tasks and core do not need separate variables, thus a criticality is referred to a core.

$$x_{rf} = \begin{cases} 1 & \text{if core } r(r = 1, \ldots, T) \text{ works at frequency} \\ & f(f = 1, \ldots, M), \\ 0 & \text{otherwise.} \end{cases}$$

The ISP model has fewer constraints than FSP due the lower number of variables.

$$O.F. = max \sum_{a \in A} \sum_{f=1}^{M} \delta_a \gamma_{af} x_{af} \tag{4.13}$$

$$\sum_{f=1}^{M} x_{af} = 1 \tag{4.14}$$

$$(\forall a \in A)$$

$$\sum_{a \in A} \sum_{f=1}^{M} GS_{la} p_{af} x_{af} + \sum_{i \in I} GS_{li} \vec{p_i} + T_l^0 + T^a \leq T_{MAX} \tag{4.15}$$

$$(\forall l \in A)$$

The constraint (4.14) bounds each core to a selected frequency. The constraint (4.15) guarantees the thermal limits imposed on the model. Where the set $A = a_i$ contains the index of the active cores and the set $I = i_i$ contains the index of idle cores directly defined from the solution of FSP. Where $A \cap I$ is empty. In general, the ISP problem is computationally simpler than the FSP problem due to the much lower number of decision variables and constraints.

In the next Section, it will be evaluated the performance of the proposed TMC in a realistic scenario and under different trade-offs in between the predicted horizons of

the FSP and ISP problems.

4.4 Experimental Results

In this Section, it will first described the emulation framework, starting from the results of the characterization of computing nodes and real scientific workload conducted in Section 4.2. The emulation framework has been developed to study the implication of the prediction interval/horizon and the task criticality generator in the thermal-aware task mapping and control of supercomputer nodes.

4.4.1 Emulation Framework

The emulation framework is composed by the following components:

- The workload traces. The traces have been extracted using a commercial tracing and profiling tool called Intel Trace Analyzer and Collector [74]. The traces contain all the MPI activities (MPI call, data copy, source/destination MPI task) with time stamps. These have been extracted for the QE-CP running on a computing node.

- The thermal simulator. It is a first order discrete state-space model matched with the computing node as described in Section 4.2.2. The model has a sample time of 10ms (Ts_{TM}), and as state variables has the temperature of each core of the node. Each core's power is computed with the power model presented in Section 4.2.3. Workload traces, which have resolution than the 10ms, have been averaged on this period to produce the percentage of time in which each task was in the MPI library for each (Ts_{TM}) interval. This value is used to model the energy-aware MPI wrapper impacts on core's power consumption.

- The thermal-aware task mapping and control problem. The TMC optimization problem proposed in Section 4.2 has been solved using IBM Ilog CPLEX 12.6.1 [75]. The emulator calls CPLEX each time there is a new TMC problem to be solved. This happens once at the application start (FSP) and periodically each ISP interval Ts_{ISP} which matches the prediction interval in the ISP problem (PI_{ISP}).

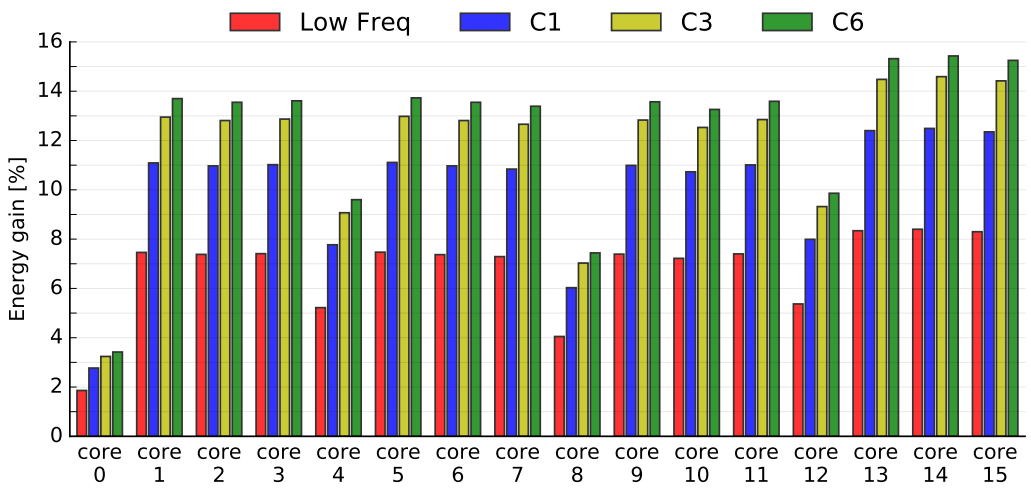


Figure 4.6: Energy saving of MPI with *reactive policy* with respect to the MPI busy-waiting

At each CPLEX call, the emulator builds a new instance of the problem with the new thermal parameters and the criticality of the tasks and it waits for CPLEX results. During the waiting time the emulator is frozen, in this way the overhead time does not impact on the chronological MPI events. CPLEX has been executed on the same machine of the emulation framework, which is the HPC node, therefore the time overheads reflect real measurement.

4.4.2 Energy-aware MPI Wrapper

The maximum achievable energy saving for the energy-aware wrapper has been quantified under ideal conditions: no interrupt, library and transitions overheads. It has been used a simulator and the MPI traces as explained above. The results has been compared with the baseline policy of Intel MPI busy-waiting considering the reactive policy to select *Low Freq* and C-state *C1*, *C3* and *C6* during communication slack.

Figure 4.6 shows with different bars the low-power state used by the reactive policy during communication phases. In y-axis is reported the achievable energy-saving with respect to the busy waiting baseline for each core (x-axis). From the plot it can be first recognized that different cores have different communication-to-computation ratio, which leads to a significant workload unbalance in QE. This is reflected by a different achievable saving for each core. When comparing the achievable energy-

reduction, Figure 4.6 shows that DVFS states are more effective in saving energy during communication phases. Indeed, while reducing the frequency to *Low Freq* saves in average 7%, using *C1* reduces the energy consumption by 10%. Using *C3* and *C6* leads to an additional saving reaching 11.7% and 12.4% respectively. Considering the different latencies intrinsic to these low-power states, the analysis suggests that the MPI runtime should maximize the usage of *C1* during communication slack with an energy saving opportunity of 10% for QE[2].

4.4.3 Evaluation of Prediction Horizons

This Section will explore how vary the frequency level for high and low critical tasks using different prediction horizons for FSP and ISP problem. The following experiments has been conducted with different prediction intervals for both FSP and ISP problems. It will be considered PI_{FSP}=1s,10s,100s,steady state (SS) and PI_{ISP}=1s,10s,100s,steady state (SS) because the thermal propagation in the system is in the order of tens of seconds as reported in table 4.2. In the following, these tests has been named with the notation $PI_{FSP} - PI_{ISP}$. It must be noted that 1s-1s represent state-of-the-art DTM solutions with no thermal-aware task-to-core mapping, while SS-SS represents state-of-the-art static DTM solutions.

For all the experiments, the temperature limit has been set to 65% of maximum temperature which can be reached by the hottest core at the maximum frequency.

Figure 4.7 shows on the y-axis the temperature evolution of the coldest core (#0) for five cases. Namely no thermal control active, no energy-aware MPI wrapper active (NoTMC-NoEAW), no thermal control active but energy-aware MPI wrapper active (NoTMC-EAW), TMC active with (1s-1s), (SS-1s), (SS-SS). For the same configurations, the Figure 4.8 shows on the y-axis the temperature evolution of the hottest core. Clearly, according to the capability of the FSP problem, to predict the long-term thermal evolution the higher critical task will be mapped on the coldest core. Indeed from Figure 4.7, it can be noticed that if no TMC calls are executed, the coldest core executes a low critical task. When the FSP is empowered with a steady-state thermal predictor instead the TMC allocates the higher critical task on the coldest core and manages to run it always at the maximum frequency. Vertical

[2]In this exploration are only considered Core C-states because package C-states are always close to 0% of utilization due the continuous presence in $C0/C1$ state on at least one core

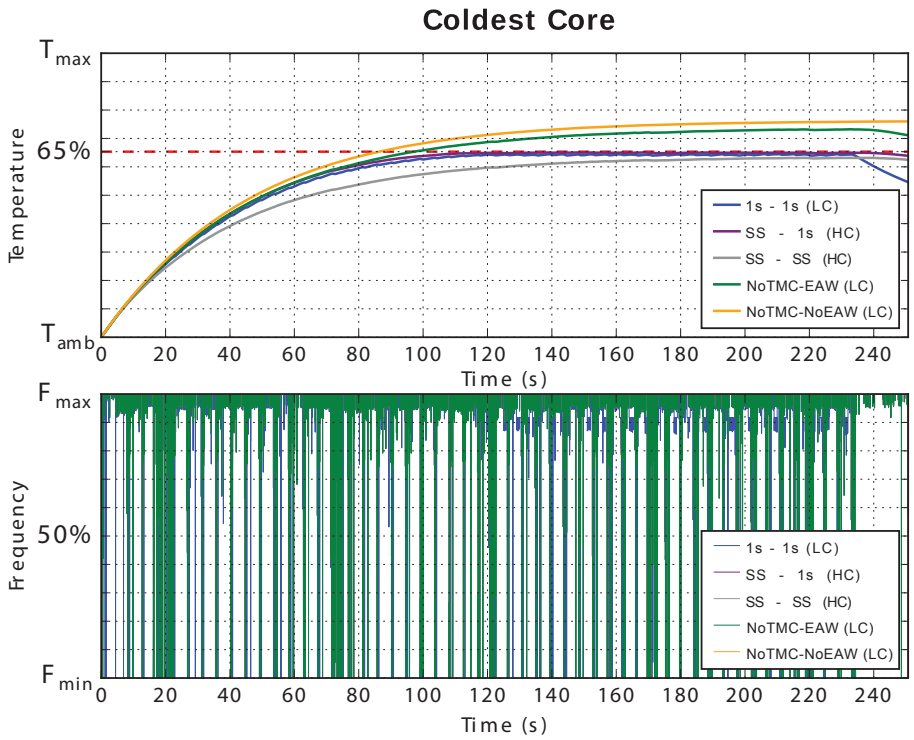


Figure 4.7: Temperature and frequency evolution for the coldest core of the system - core #0

spikes of the frequency are caused by the energy-aware MPI wrapper, which sets the minimum frequency of the core during the MPI phases. As a consequence, the maximum temperature reached by NoTMC-EAW is lower than NoTMC-NoEAW; showing its effectiveness in reducing the power consumption. Differently, short time horizons (1s-1s) in the FSP do not allow the solver to "see" the constraint and thus lead to a sub-optimal task mapping allocation. As a consequence, the high critical task needs to be frequency limited to meet the thermal constraint as the thermal capacitance effect vanishes.

4.4.4 Evaluation of the Task Criticality Generator

As previously introduced, the task criticality is a key parameter for the final application performance. Figure 4.9 shows the penalty in term of the execution time of application when the criticality level is considered equal for each task respected of the

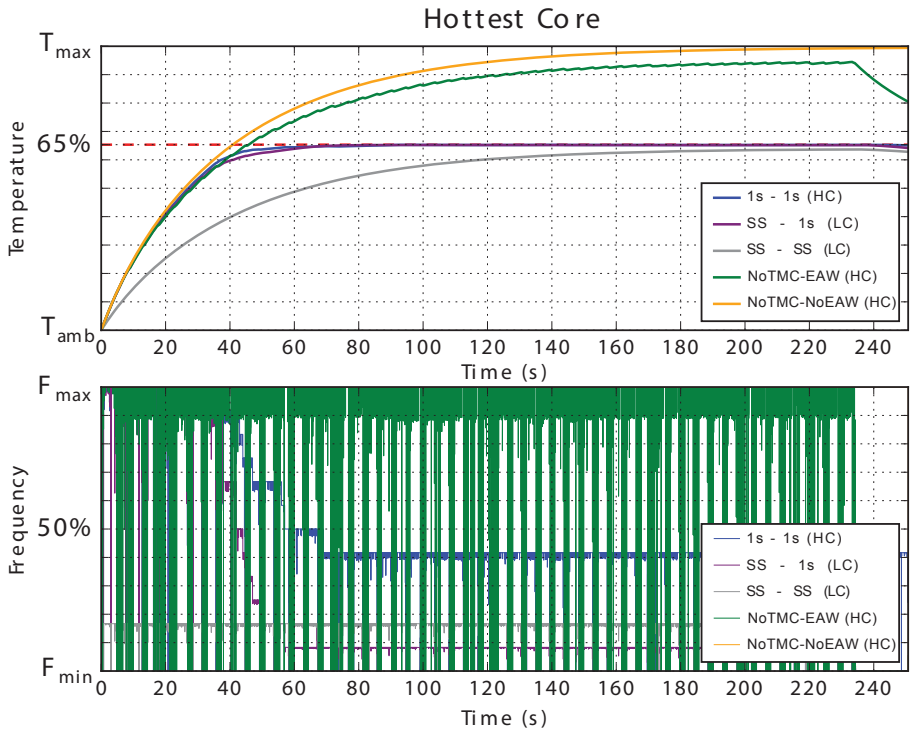


Figure 4.8: Temperature and frequency evolution for the hottest core of the system - core #14

one obtained by the TMC task criticality generator presented in Section 4.3.1.

Figure 4.9 reports on the x-axis the coreId where the highest critical task is allocated. When the highest critical task is located on the core #6 or on the core #8 the highest and lowest penalty is given to the application, respectively 21.18% and 0.33%. When the root MPI task is located on the core #8, it is a lucky case, this means that it runs on the "coldest" core of the system where the TMC runtime can easily increase the core's frequency without violating the thermal constraint. On the other hand, when the root MPI task is located on the "hottest" core there is a high penalty due the difficult of the runtime to increase the frequency on that core. To conclude, It can be evidently seen that in all cases the TMC criticality generator outperforms the cases with task with the same criticality.

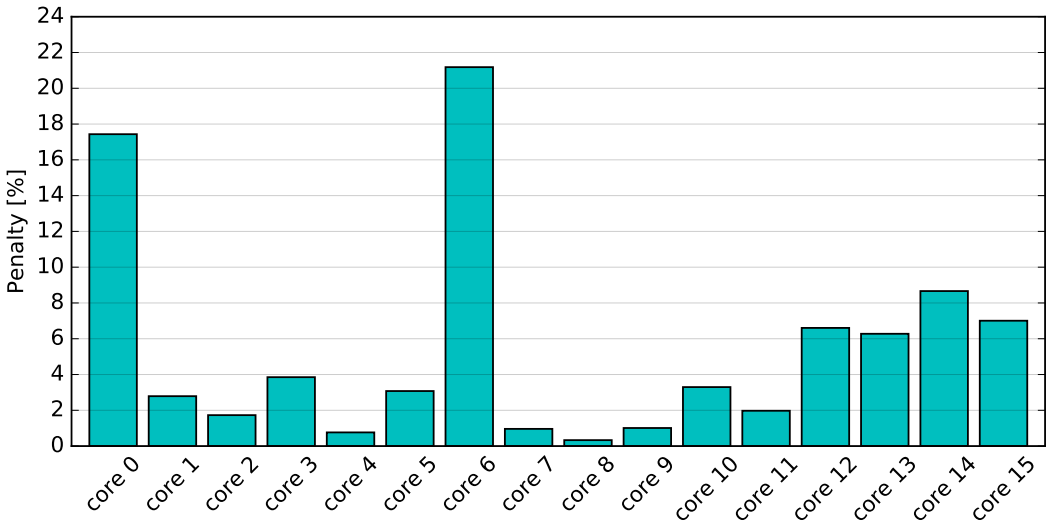


Figure 4.9: Execution time penalty in benchmarks with equal per-task criticality level with respect to the benchmark with the TMC criticality generator. Every run identify on which core was pinned the highest critical task.

4.4.5 Performance Gain

Figure 4.10 depicts the average frequency of the cores that host the highest critical tasks and the average frequency for all the cores in each configuration. Interestingly, in all the cases the highest critical task never reaches the maximum average frequency. This is the effect of the energy-aware MPI wrapper which reduces the core frequency during MPI calls.

The error bars show the variance for each configuration among different executions of the same QE-CP problem while moving the highest critical task from the MPI root task to another one. This means that if the default position of highest critical task shift from 0 to 15 in the MPI rank, all the configurations with predict interval in the FSP (PI_{FSP}) of 1 and 10 seconds have a huge variation. This can be explained by the fact that in both experiments the FSP has a prediction horizon which is too short to see the effect of long-term thermal evolution and thus it cannot predict which core will hit the thermal constraint. For this case, the allocation FSP problem is trivial, tasks are allocated on the first available core following a simple numerical binding where the task 0 will be allocate to the core 0 and so on. This binding is also the default on the Intel MPI library. In this particular case, if the highest critical task is lucky, it will be pinned on a "cold" core. Vice versa, if the highest critical task is

unlucky, it will be mapped on a "hot" core. At the steady-state the frequency of the core will be limited by the ISP to respect the thermal constraint. On the other cases, the PI_{FSP} is always enough to sense the thermal constraint. The optimization model will avoid the binding of the highest critical task on a "hot core". In this case the highest critical task will be pinned on a "cold" core allowing the highest critical task to work at maximum frequency.

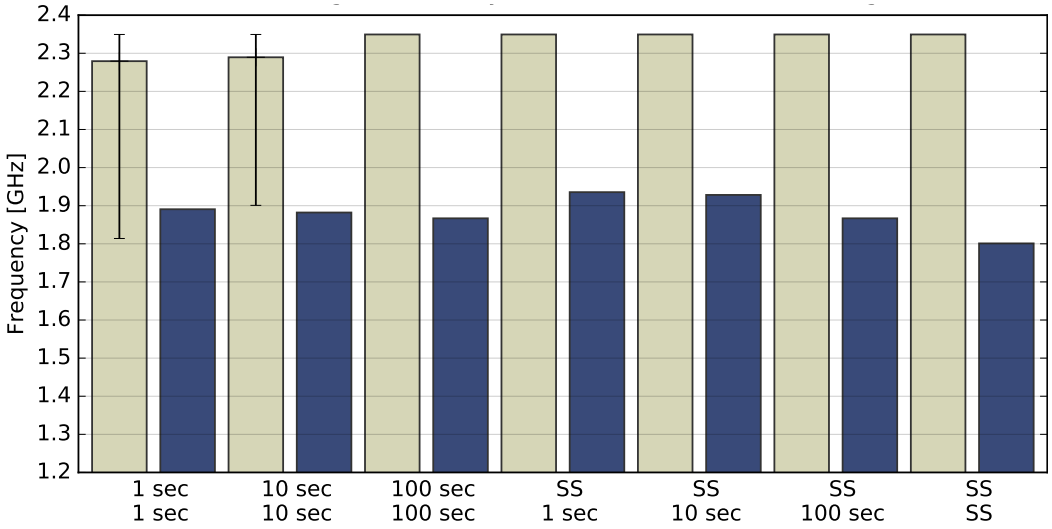


Figure 4.10: Comparison between average core's frequency (dark bars) and the frequency of the highest critical core (light bars) using different configuration for the optimization problem.

The baseline is the SS-SS configuration, which model state-of-the-art solutions based on static allocation of tasks and frequency. The 1s-1s and 10s-10s induces performance penalties on the highest critical task, while they lead to an increase of performance of the 4.97% and 4.50% respectively in average in all the cores. For the remaining configurations, no penalty are measured for the highest critical tasks and a gain of to 7.46%, 7.06% and 3.65% respectively for the configuration SS-1s, SS-10s and SS-100s. These results show that short horizon predictive models pay off in the ISP as it allows to take advantage of the thermal capacitance. In the next Section will be presented the conclusions on the solver overhead.

4.4.6 Overhead Time

Figure 4.11 shows cumulative overhead for different configurations and quantify the induced performance loss as it sums up to the execution time. The FSP bars represent the overhead time of the FSP problem solved only once at the application start, while the ISP bars are the sum of the overhead times of all iterations of the ISP solver.

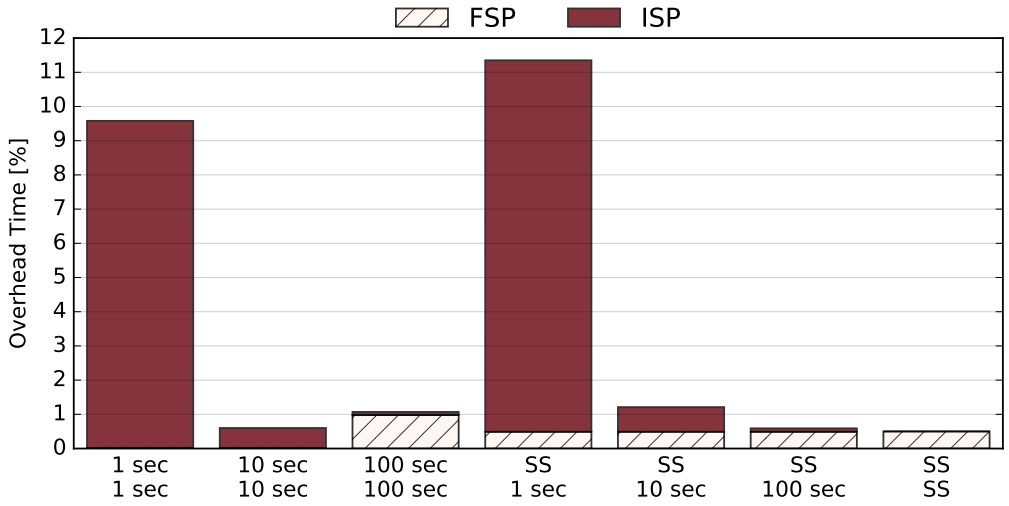


Figure 4.11: Cumulative overhead induces by the optimization problem using different configurations for the optimization problem.

For all the instances and the configurations, the solver is capable of finding the optimal solution. CPLEX allows to bound the solution time by the so-called deterministic ticks, this approach has been used to limit the solution time in case of harder problem. Authors of [50] show for a 60 core instance that the optimally gap always reduces below the 0.002% with a maximum number of 180 ticks.

It can be seen that for the 1s-1s and 10s-10s configuration the FSP solver time is negligible. After 1 second or 10 seconds the thermal transient has not reached the thermal constraint, for this reason the solution is trivial and consequently it immediately converge. Instead, all the other configurations have an average overhead time of 0.59% of total Tts.

The total overhead time for the ISP significantly changes when vary the PT_{ISP} and the Ts_{ISP}. Obviously, the ISP with a prediction interval of 1 second will be called hundred times more than a ISP with a prediction interval of 100 seconds. The results respect this trend, in particular for 1 seconds of prediction interval leads to an average

penalty of 10.20% of total execution time, which makes this configuration worse than a static allocation SS-SS and it causes the solution overhead (7.46% of performance gain - 10.20% of overhead). Interesting the 10 seconds case (SS-10s) reduces the total penalty to the 0.64% which in conjunction to the 7.06% of performance gain with respect to the static-allocation lead to an overall performance gain of the 6%. At 100 seconds the total overhead penalty decreases to the 0.09%. However, for this case the performance gain is only of the 3.46% making it less performing than the SS-10s case.

4.5 Summary on HPC Optimal Thermal Control

In this Chapter has been proposed a thermal-aware mapping and control of thermally-bound HPC nodes. The system implements a novel ILP formulation for thermal-aware optimization and an exploration analysis on the workload application to address performance promoting critical MPI tasks. The work is focused on real HPC hardware and workload. The thermal characteristics has been extracted to a HPC compute node as well as workload traces to study the workload distribution to identify critical MPI tasks. The control system relies on this information to optimize the task allocation and the frequency selections in thermal-constraint HPC nodes.

In the experimental Section, the system has been compared with state-of-the-art DTM solutions which dynamically control only the frequency selection of the cores or it can choose a statically task allocation with a specific frequency. The experimental results show that using a long-time horizon for the task allocation and a short time horizon for selecting DVFS levels at execution time can lead up to 6% performance gain including overheads. Moreover, the task criticality model embedded in the DTM system can avoid the pinning of critical tasks on hot cores where OTC cannot promote this task with high frequency. This can cause high performance degradation up to 21.18% of the entire application execution.

Chapter 5

Energy-aware MPI Runtime

5.1 Overview

Energy and power consumption are prominent issues in today's supercomputers and are foreseen as a limiting factor of future installations. A significant amount of power in large scale HPC applications is wasted in communication synchronization-related idle times. However, due to the time scale at which communication happens, transitioning in low power states during communication's idle times may introduce overheads in scientific applications.

In this Chapter COUNTDOWN will be presented, a run-time library supported by a methodology and analysis tool for identifying and automatically reducing the power consumption of the computing elements during communication and synchronization. COUNTDOWN saves energy without imposing significant time-to-completion increase by lowering CPU's power consumption only during idle times for which power state transition overhead are negligible. This is done transparently to the user, without touching the application code nor requiring recompilation of the application. This methodology has been tested in a production Tier-1 system.

The main contributions of this work are:

- An analysis of the effects and implications of fine-grain power management in today's supercomputing systems targeting energy saving in the MPI library. The analysis shows that in today's HPC processors there are significant latencies in the hardware to serve low power states transitions. This delay is at the source of inefficiencies (overheads and saving losses) in the application for fine-grain power management in the MPI library.

- A set of benchmarks running on a single HPC node have been used to show that: (a) there is a potential saving of energy with negligible overheads in the MPI communication slack of today's scientific applications; (b) these savings are jeopardized by the time the hardware takes to perform power state transitions; (c) when combined with low power states it can leverage on the turbo logic improving the execution time of the application.

- COUNTDOWN library consists of a runtime able to automatically track at fine granularity MPI and application phases to inject power management calls. Indeed, it is able to identify MPI calls with energy-saving potential for which it is worth to enter a low power state, leaving low-wait-time MPI calls unmodified to prevent overheads in low power state transitions. COUNTDOWN's principles can be used to both inject DVFS capabilities as well as configure properly the MPI runtime and take advantage of MPI idle-waiting mechanisms. COUNTDOWN works at execution time without requiring any off-line knowledge of the application and it is completely plug and play. In fact, it does not require any modification of the source code and compilation toolchain. COUNTDOWN can be dynamically linked with the application at loading time, this means that it can intercept dynamic linking to the MPI library instrumenting all the application calls to MPI functions before the execution workflow jumps to the MPI library. The runtime also provides a static version of the library which can be injected in the application at linking time. COUNTDOWN supports C/C++ and FORTRAN HPC applications and most of the open-source and commercial MPI libraries.

- Evaluating COUNTDOWN with a wider set of benchmarks and low power state mechanisms. In large HPC runs, COUNTDOWN leads to savings of 23.32% in average for the NAS[76] parallel benchmarks and to 22.36% for an optimized QE run on 3456 cores, which increases to the 37.74% when QE is executed without optimization for the communication.

This work has been presented to the scientific community in [77].

5.2 Background

In this Section are used two practical examples to see the implications and challenges of transitioning into low power states (P/C/T-states) during synchronization and communication primitives for energy-savings.

The test platform is based on a compute node equipped with two Intel Haswell E5-2630 v3 CPUs, with 8 cores at 2.4 GHz nominal clock speed and 85W TDP and a real production software stack for Intel systems. Intel *MPI Library 5.1* has been used as the runtime for communication coupled with Intel *ICC/IFORT 18.0* as toolchain. The Intel software stack it is currently used in the target systems as well as well supported in most of HPC machines based on Intel architectures.

All the tests in this work have been executed with QE presented in Section 3.3.2. CP package has been used for these single nodes tests. QE mostly used packages are: (i) *Car-Parrinello* simulation; (ii) *Plane-Wave Self-Consistent Field* (PWscf) which solves the self-consistent Kohn and Sham KS equations and obtain the ground state electronic density for a representative case study. The code uses a pseudo-potential and plane-wave approach and implements multiple hierarchical levels of parallelism implemented with a hybrid MPI+OpenMP approach. As of today, OpenMP is generally used when MPI parallelism saturates, and it can improve the scalability in the highly parallel regime. Nonetheless in the following it will be only refereed to data obtained with pure MPI parallelism since this is the main focus of this work and this choice does not significantly impair the conclusions reported later.

To exploit the system behavior for different workload distributions in a single node evaluation, the computation of the band structure of the Silicon has been focused along the main symmetry. When executed by a user with no domain expertise and with default parameter QE runs with a hybrid MPI parallelization strategy with only one MPI process used to perform the diagonalizzation and all the MPI processes used to perform the FFT kernel. It will be later refer to this case as *QuantumESPRESSO CP Not Expert User* (QE-CP-NEU). Differently, when an expert user runs the same problem, he changes the parameters to better balance the workload by using multiple MPI processes to parallelize also the diagonalization kernel. It will be later refer to this case as *QuantumESPRESSO CP Expert User* (QE-CP-EU). In the QE-CP-NEU case, when a single process works on the linear algebra kernel, the other ones remain in busy wait on the MPI call. In the following text, the fine-grain power management

solution will be compared with the busy-waiting mode (default mode) of MPI library where processes continuously polling the CPU for the whole waiting time in MPI synchronization points.

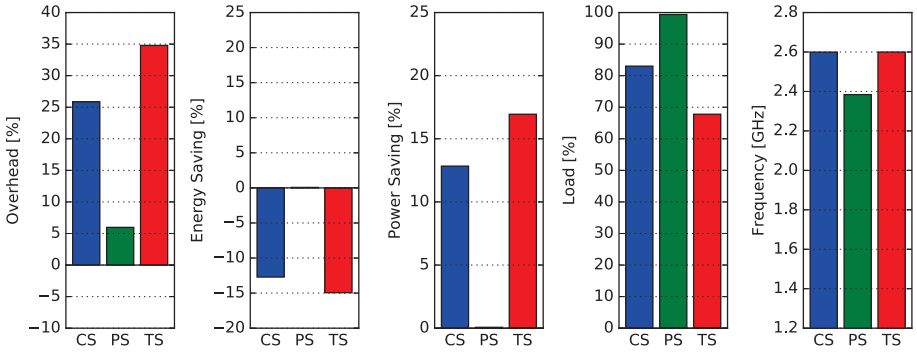
5.2.1 Wait-mode/C-state MPI library

By default, MPI libraries use a busy-waiting policy in collective synchronizations to avoid performance penalties. This is also the case of Intel MPI library. This library can also be configured to release the control to the idle task during waiting time to leverage the C-states of the system. This allows cores to enter in sleep states and being woken up by the MPI library when the message is ready through an interrupt routine. In Intel MPI library, it is possible to configure the wait-mode mechanism through the environment variable *I_MPI_WAIT_MODE*. This allows the library to leave the control to the idle task, reducing the power consumption for the core waiting in the MPI. Clearly the transitions into and out of the sleep mode induce overheads in the application execution time.

In Figure 5.1 are reported the experimental results, the wait-mode strategy is identify with *CS*. The overhead induced by the wait mode with respect to the default busy-waiting configuration, which worsen by 25.85% the execution time. This is explained by the high number of MPI calls in the QE application which leads to frequent sleep/wake-up transitions and high overheads. From the same Figure, it can be seen that the energy saving is negative, which is -12.72%, this is because the power savings obtained in the MPI primitives does not compensate the overhead induced by the sleep/wake-up transitions. Indeed, the power reduction is 12.83%. This is confirmed by the average load of the system, which is 83.02% as effect of the C-states activity in the MPI primitives. The average frequency is 2.6GHz, which is the standard turbo frequency of the target system.

Surprisingly, the QE-CP-NEU case has a negative overhead (-1.08% overhead is a speedup). This speed up is given by the turbo logic of the system. Indeed, the average frequency is slightly higher than 2.6GHz, which means that the process doing the diagonalization can leverage the power budget which is freed by the other processes not involved in the diagonalization which are waiting in a sleep state in the MPI runtime. In Figure 5.2, it is reported the average frequency of the process working on the diagonalization and the average frequencies of all the other MPI processes. In the

(a) All MPI processes are involved in the diagonalization QE-CP-EU



(b) Single MPI process is involved in the diagonalization QE-CP-NEU

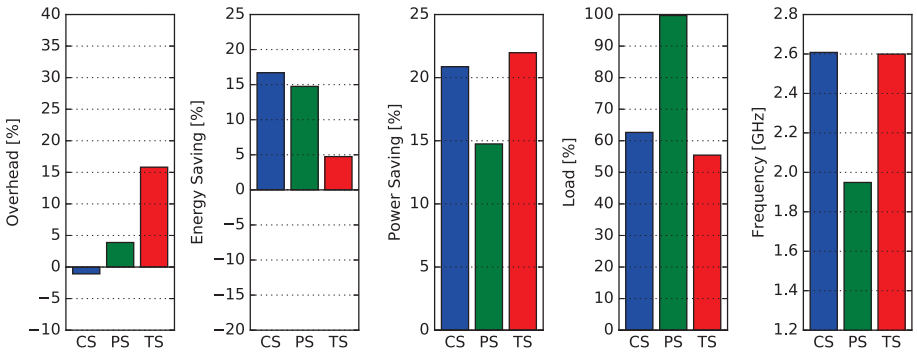


Figure 5.1: Overhead, energy/power saving, average load and frequency for QE-CP-EU (a) and QE-CP-NEU (b). Legend: C-state (CS), P-state (PS) and T-state (TS) mode. Baseline is busy-waiting mode (default mode) of MPI library.

target system, a single core can reach up to 3.2 GHz if only one core is running, this is what happens when all cores are waiting in a sleep state for the termination of the diagonalization workload. The benefit of this frequency boosting unleashed by the idle mode on the MPI library and the unbalanced workload, can save up to 16.69% of energy with a power saving of 20.86%.

As conclusion of this first exploration, it is possible to leverage on the wait mode of the MPI library to save power but without increasing the execution time. Energy savings and the impact on the Tts depends on the MPI call granularity which can lead to significant penalties if the application is characterized by frequent calls.

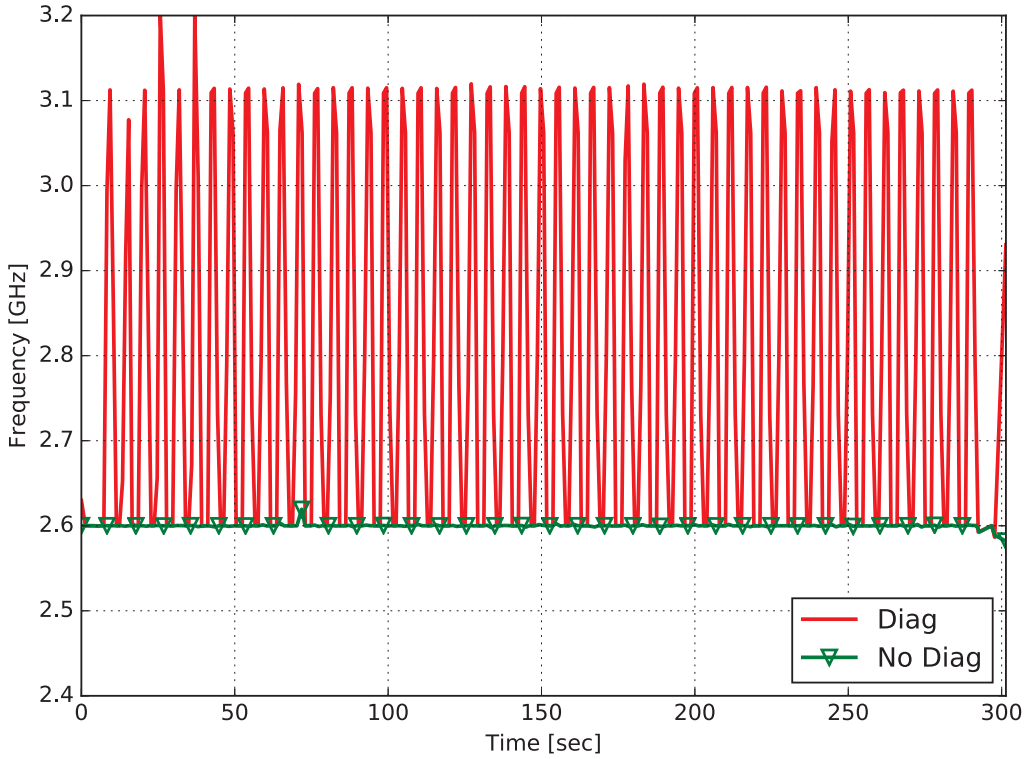


Figure 5.2: Time plot of frequency for QE-CP-NEU identifies the frequency of the MPI process working on the diagonalization, *No Diag* is the average frequency of MPI processes not involved in the diagonalization.

5.2.2 DVFS/P-state MPI library

To overcome the overheads of C-state transitions, an initial exploration focuses on the active low power states (DVFS states). Intel MPI library does not implement such a feature, so it has been manually instrumented all the MPI calls of the application with an *epilogue* and *prologue* function to scale down and raise up the frequency when the execution enters and exits from an MPI call. To avoid interference with the power governor of the OS, the power governor has been disabled in the compute node granting the complete control of the frequency scaling. The MSR driver has been used to change the current P-state writing *IA32_PERF_CTL* register with the highest and lowest available P-state of the CPU, which corresponds to turbo and 1.2GHz with respect to points. Figure 5.1 reports the results of this exploration, where the P-state case is labelled with *PS*.

In the overhead plot in Figure 5.1.a, it can be seen that the overhead is signifi-

cantly reduced with respect to C-state mode, reducing the 25.85% overhead obtained previously to 5.96%. This means that the overhead of scaling the frequency is lower respect to the sleep/wake-up transitions cost. However, the energy and power savings are almost zero. This is due to the fact that as before in QE-CP-EU, all the MPI processes participate to the diagonalization; thus, a high number of MPI calls report a very short duration. This is also confirmed by the average frequency, which does not show significant variations with respect to the baseline (busy waiting), with a measured average frequency of 2.4GHz. The load bar reports 100% of activity, which means that there is no idle time as expected.

Focusing in the QE-CP-NEU case, in Figure 5.1.b, the overhead is 3.88% which is reduced with respect to QE-CP-EU. In addition, this case has a significant energy and power saving, respectively of 14.74% and 14.75%. These saving are due to the workload unbalance and to long time spent in the MPI calls from the processes not involved in the diagonalization. This is confirmed by the lower average frequency (1.95GHz) while the load is unaltered as expected.

In conclusion using DVFS states for fine-grain power management instead of the idle mode allows to better control the overhead for both balanced and unbalanced workload. However, the overhead is still significant and in HPC the Tts is the prime goal.

5.2.3 DDCM/T-state MPI library

One important question is when the overheads of fine-grain power management strategies are induced by the specific power management states. In that case, it would be worth to try duty-cycling low power states. In this Section, it has been used the Dynamic Duty Cycle Modulation (DDCM) (also known as throttling states or T-states) available in the Intel architectures aiming to reduce the overheads. DDCM has been supported in Intel processors since Pentium 4 and enables on-demand software-controlled clock modulation duty cycle. In Intel CPUs, DDCM is used by the *HW power controller* to reduce the power consumption when the CPU identifies thermal hazards. Similarly to [78], DDCM is used to reduce the power consumption of the cores in MPI calls. The application has been manually instrumented using a *prologue* function of each MPI call to configure DDCM to 12.5% of clock cycles, which means for each clock cycle are gated the next 7; while in the *epilogue* function, the DDCM

is restore to 100% of clock cycles, this is possible writing to the DDCM configuration register, called *IA32_CLOCK_MODULATION*, through the MSR driver.

In Figure 5.1.a, the DDCM results are reported with *TS* bars. Surprisingly, the overheads induced by T-states are greater than the wait mode, and equal to 34.78%. As consequence, the energy saving is the worst, leading to an energy penalty of 14.94%. The load is greatly reduced owing to the throttling, in average of 67.78%, while the frequency is constant to 2.6GHz.

Figure 5.1.b reports T-state results for QE-CP-NEU. Even for this unbalanced workload case the T-states are the worst. T-state transitions introduce an overhead of 15.82% consequent of the power reduction, with a very small energy saving, only of the 4.75%, and a power saving of 21.97%. The load of the system is reduced to 55.45%, similarly to the idle mode, and the frequency remained unchanged as expected.

As a matter of fact, the phase agnostic fine-grain power management leads to significant application overheads which may nullify the overall saving. Though, it is needed to bring knowledge of the workload distribution and the communication granularity of the application in the fine-grain power management. In the next Section will be introduced the COUNTDOWN approach which addresses this issue.

5.3 Framework

COUNTDOWN is a run-time library for profiling and fine-grain power management written in C language. COUNTDOWN is based on a *profiler* and on a *event* module to inspect and react to MPI primitives. The key idea in COUNTDOWN can be summarized as follows: Every time the application calls an MPI primitive, COUNTDOWN intercepts the call with minimal overhead and uses a timeout strategy [63] to avoid changing the power state of the cores during fast application and MPI context switches, where doing so may result only in state transition overhead without a significant energy and power reduction. In a nutshell, each time the MPI library asks to enter in low power mode, COUNTDOWN defers the decision for a defined amount of time (the timeout). If the MPI phase terminate within this amount of time, the hardware does not enter in low power state. Thus, COUNTDOWN filters short MPI phases with no potential for power-management-based energy reduction.

In Figure 5.3 are depicted the COUNTDOWN's components. COUNTDOWN ex-

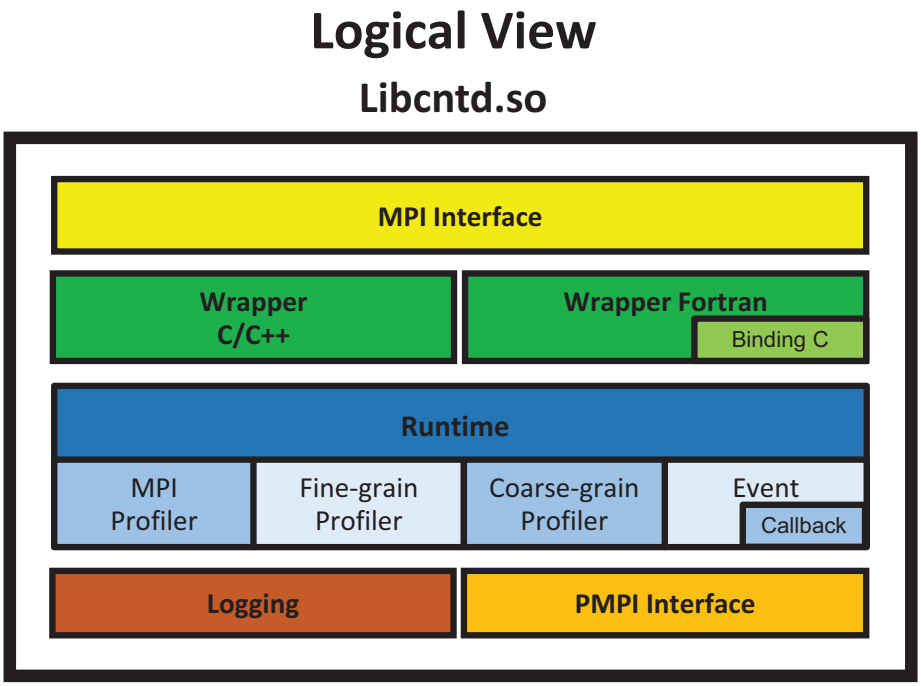


Figure 5.3: Logical view of all the components of COUNTDOWN.

poses the same interface as a standard MPI library and it can intercept all MPI calls from the application. COUNTDOWN implements two wrappers to intercept MPI calls: i) the first wrapper is used for C/C++ MPI libraries, ii) the second one is used for FORTRAN MPI libraries. This is mandatory due to the fact that C/C++ and FORTRAN MPI libraries produce assembly symbols which are not application binary (ABI) compatible. The FORTRAN wrapper implements a marshalling and unmarshalling interface to bind FORTRAN MPI handlers in compatible MPI C/C++ handlers.

When COUNTDOWN is used with an application, every MPI call is enclosed in a corresponding wrapper function that implements the same signature. The wrapper function calls the equivalent PMPI call, but after and before a *prologue* and an *epilogue* function. Both functions are used by the profile and by the event modules to support monitoring and power management, respectively. COUNTDOWN interacts with the *HW power manager* through a specific *Events* module in the library. The *Events* module can also be triggered from system signals registered as callbacks for timing purposes. COUNTDOWN configurations can be done through environment variables, it is possible to change the verbosity of logging and the type of HW performance

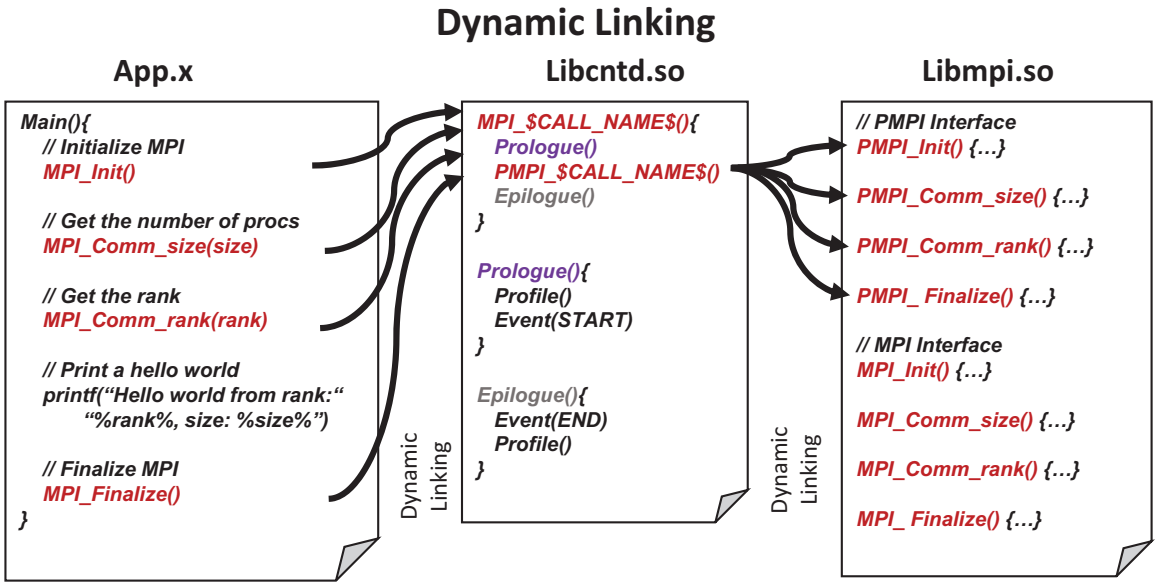


Figure 5.4: Dynamic linking events when COUNTDOWN is injected at loading time in the application.

counters to monitor.

The library targets the instrumentation of applications through dynamic linking, as depicted in Figure 5.4, without user intervention. When dynamic linking is not possible, COUNTDOWN also implements a fallback as a static-linking library, which can be used while building the the application, to add COUNTDOWN at compilation time. The advantage of using the dynamic linking is the possibility to instrument every MPI-based application without any modifications of the source code nor the toolchain, even without re-compiling it. Linking COUNTDOWN to the application is straightforward: it is enough to configure the environment variable *LD_PRELOAD* with the path of COUNTDOWN library and start the application as usual.

5.3.1 Profiler Module

COUNTDOWN uses three different profiling strategies targeting different monitoring granularity.

- The *MPI profiler*, is responsible to collect all information regarding the MPI activity. For each MPI process, it collects information on MPI communicators, MPI groups, and the coreId. In addition, the COUNTDOWN run-time library

profiles each MPI call by collecting information on the type of the call, the entrance and exit times and the amount of data exchanged with the other MPI processes.

- The *fine-grain micro-architectural profiler*, collects micro-architectural information at every MPI call along with the *MPI profiler*. This profiler uses the user-space RDPMC instruction to access the performance monitoring units implemented in Intel's processors. It monitors the average frequency, the *HW power controller* and the instructions retired for each MPI call and for the application phase. Moreover, it is able to access configurable performance counters that can be used to monitor user-specific micro-architectural metrics.

- The *coarse-grain profiler* monitors a larger set of HW performance counters available in the system. In Intel architectures, privileged permissions are required to access HW performance counters. Such level of permissions cannot be granted to the final users in production machines. To overcome this limitation, the MSR_SAFE [79] driver has been configured to grant the access to standard users to a subset of privileged architecture registers, while avoiding security issues. At the core level, COUNTDOWN monitors TSC, instructions retired, average frequency, C-state residencies and temperature. At uncore level, it monitors CPU package energy consumption, C-state residencies and temperature of the packages. This profiler uses RAPL to extract energy/power information from the CPU. The *coarse-grain profiler*, due the high overhead needed by each single access to the set of HW performance counters monitored, uses a time-based sample rate. Data are collected at least Ts second delay from the previous collection. The *fine-grain micro-architectural profiler* at every MPI calls checks the time stamp of the previous sample of *coarse-grain profiler* and, if it is above Ts seconds, triggers it to get a new sample. These capabilities are added to the application through the *prologue* and *epilogue* functions as shown in Figure 5.4.

COUNTDOWN also implement a logging module to store profile information in a text file which can be written in local or remote storage. While the log file of *MPI profiler* can grow with the number of MPI primitives and it can become significant in long computation (thus the information is stored in binary files), the logging module

also reports a summary of these information in an additional text file.

5.3.2 Event Module

COUNTDOWN interacts with the *HW power controller* of each core to reduce the power consumption. It uses MSR_SAFE to write the architectural register to change the current P-state independently per core. When COUNTDOWN is enabled, the *Events* module decides the performance at which to execute a given phase.

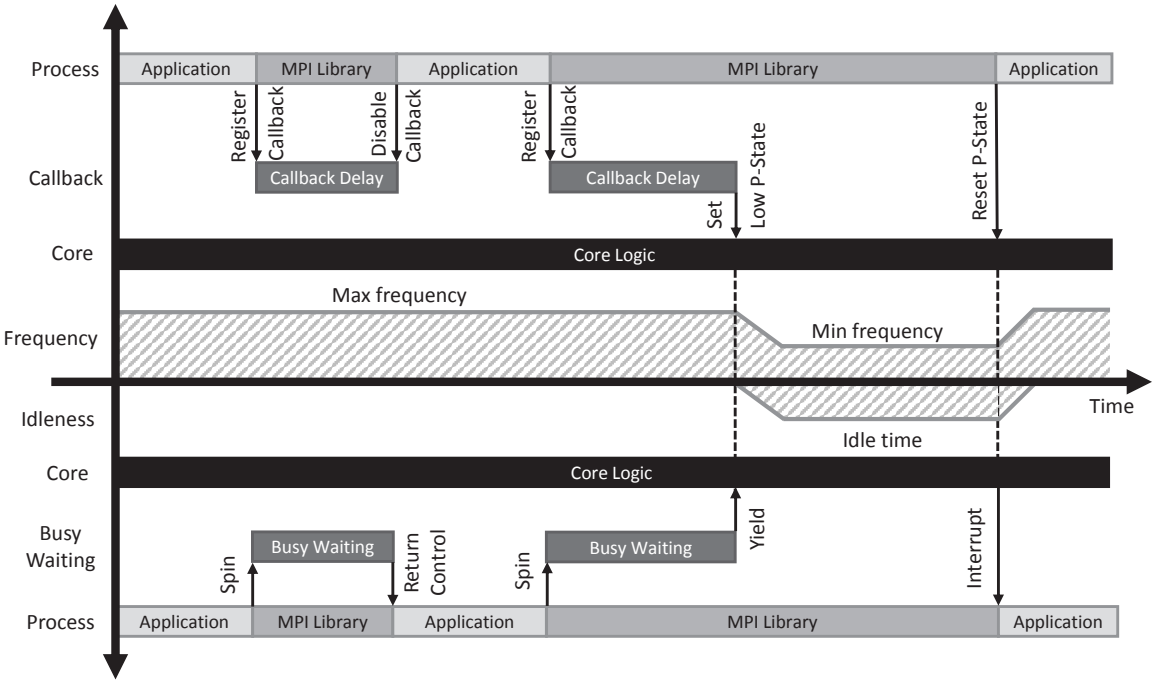


Figure 5.5: On the upper side is depicted the timer strategy utilized in COUNTDOWN, while in the lower side is depicted the idle-wait mode with timer implemented in the Intel MPI library.

COUNTDOWN implements the timeout strategy through the standard Linux timer APIs, which expose the system calls: *setitimer()* and *getitimer()* to manipulate users-space timers and register callback functions. This methodology is depicted in Figure 5.5 in the top part. When COUNTDOWN encounters an MPI phase, in which opportunistically can save energy by entering in a low power state, COUNTDOWN *registers* a timer callback in the *prologue* function (Event(start)), after that the execution continues with the standard workflow of the MPI phase. When the timer expires, a

system signal is raised, the "normal" execution of the MPI code is interrupted, the signal handler triggers the COUNTDOWN *callback*, and once the callback returns, the execution of MPI code is resumed at the point it was interrupted. If the "normal" execution returns to COUNTDOWN (termination of the MPI phase) before the timer expiration, COUNTDOWN *disables* the timer in the *epilogue* function and the execution continues like nothing happened. The callback COUNTDOWN can be configured to enter in the lower T-state (12.5% of load), later refereed as *COUNTDOWN THROTTLING*, or in the lower P-state (1.2GHz) later refereed as *COUNTDOWN DVFS*.

Intel MPI library implements a similar strategy, but it relies on the sleep power state of the cores. Its behavior is depicted in the bottom part of Figure 5.5. If the environment variable *I_MPI_WAIT_MODE*, presented in Section 5.2.1, is combined with the environment variable *I_MPI_SPIN_COUNT*, it is possible to configure the spin count time for each MPI call. Time after which the MPI library leaves the execution to the idle task of the CPU. This parameter does not contain a real time value but contains a value which is decremented by the spinning procedure on the MPI library until it reaches zero. This allows the Intel MPI library to spin on a synchronization point for a while, and after that, enter in an idle low power state in order to reduce the power consumption of the core. The execution is restored when a system interrupt wakes up the MPI library signaling the end of the MPI call. Later, It will be refereed to this mode as *MPI SPIN WAIT*.

Next Section will clarify though experiment why the timeout logic introduced by COUNTDOWN is effective in making fine-grain power management possible and convenient in MPI parallel applications.

5.4 Experimental Results

In this Section will be presented: (i) an overhead analysis of COUNTDOWN, (ii) the effect of timeout strategy using different timeout delays, and (iii) the evaluation on a single node and on a real-production HPC system with real applications.

5.4.1 Framework Overheads

The performance overhead has been evaluated running MPI applications instrumented with the profiler module of COUNTDOWN without changing the cores' frequency. QE-CP-EU has been run on a single node, which is the worst case for COUNTDOWN in term of number and granularity of MPI calls to profile, because all network-related overheads in MPI calls are nullified, intra-chip communication and synchronization are orders-of-magnitude faster than the inter-chip or inter-node ones. Hence, MPI wait-times exploitable for power management are generally much shorter.

In this run have been counted more than 1.1 million of MPI primitives for each process in the diagonalization task: the run-time library needs to profile in average an MPI call every 200us for each process. The overhead has been measured comparing the execution time with and without COUNTDOWN instrumentation. The test has been repeated five time and reported the median case. Experimental results show that even in this unfavorable setting, the COUNTDOWN profiler introduces an overhead in the execution time which is less than 1%. This result is also supported by the overhead measurements of the prologue and the epilogue routines that COUNTDOWN injects in the application for each MPI call, which costs between 1us and 2us. The same test has been repeated changing the cores' frequency to assess the overhead of a fine-grain DVFS control. To measure only the overhead caused by the interaction with the DVFS knobs, COUNTDOWN has been forced to write always the highest P-state in the DVFS control registers. Thus, the application slowdowns caused by the frequency variation has been avoided only profiling the overhead caused by the register access. The experimental results report of 1.04% of overhead to access to the DVFS control register and doing the profile. While the overhead of memory and IO are completely negligible due the fact that the memory required is in the order of hundreds of kilobytes for each MPI process and COUNTDOWN does not communication among the instances.

These results prove that the source of the overheads of phase agnostic fine-grain power management is not related to issuing the low power state transition (DVFS in this case). Figure 5.6 focuses on understanding the source of this by replicating the tests of Section 5.2 for both QE-CP-EU and QE-CP-NEU, but now entering in the low power state only for MPI phases longer than a given time threshold. For the P-state and T-state (Figure 5.6.b and Figure 5.6.c), it has been obtained that by profiling

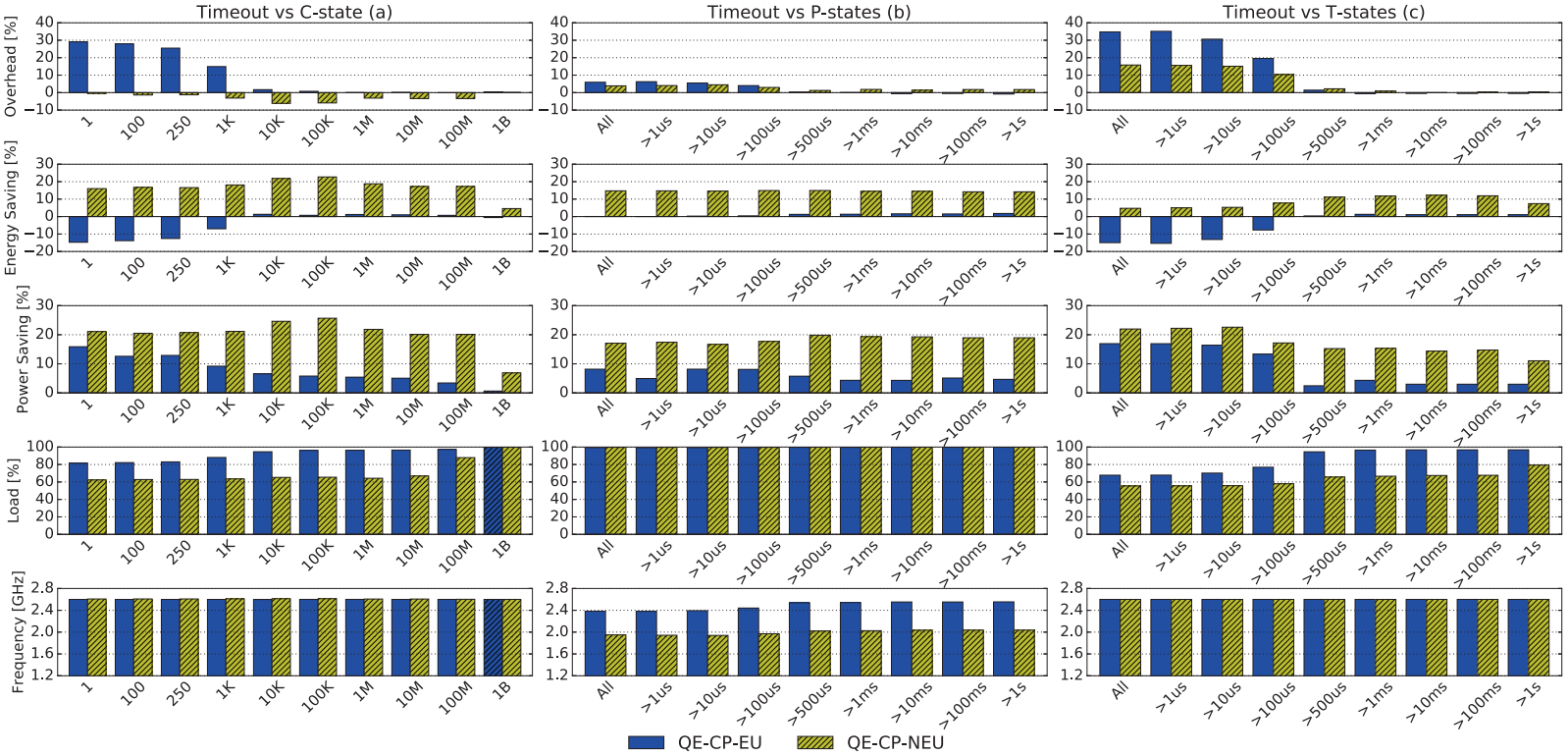


Figure 5.6: Impact of MPI phases duration on the overheads, energy and power savings of C/P/T-state for QE-CP-EU and QE-CP-NEU.

in advance the duration of each MPI phase instrumenting with low-power routines only the phases which had a duration longer than the threshold. On the x-axes is reported the time threshold value. For C-state (Figure 5.6.a) it has been used the *I_MPI_SPIN_COUNT* parameter of Intel MPI library to filter out short phases. On the x-axis is reported the *I_MPI_SPIN_COUNT* parameter.

From the plot, it is possible to recognize that there is a well-defined threshold of 500us for the T-state and P-state case and of 10K iteration steps for the C-state. After this threshold the overhead introduced by the fine-grain power management policy is greatly reduced and the energy savings becomes positive. The next Section analyze why this happens by focusing on the P-state case.

5.4.2 DVFS Overheads and Time Region Analysis

To find the reason of the higher overhead when frequency reduction is applied in all the MPI phases as highlighted in previous Section, two scatter plots has been realized. On x-axis of the left plot is reported the time duration of each MPI phase and on the right plot is reported the time duration of each application phase. For both plots, the y-axes measures the average frequency of that phase. This test is conducted by instrumenting through COUNTDOWN each MPI call: i) with a *prologue* function setting the lowest frequency (1.2GHz) and ii) with an *epilogue* function setting the highest frequency (Turbo).

In theory would be aspect that all MPI phases had executed at the minimum frequency and application phases had run always at maximum frequency. Indeed, it is a matter of fact that MPI phases running at high frequency may cause energy waste, while application phases running at low frequency cause performance penalty to the application. The experimental results show that for phases with a time duration between 0us and 500us, the average frequency vary in the interval between the high and low CPUs' frequency values, while above it, it tends to the desired frequency for that phase. This can be explained by the response time of *HW power controller* in serving P-state transition of Intel Haswell[1] CPUs [27]. The *HW power controller* periodically reads the DVFS control register to check if the operating system has specified a new frequency level, this interval has been reported to be 500us in previous study [27] and matches with empirical threshold shown in this work.

[1]The Intel Broadwell's *HW power controller* works as well.

This means that every new setting for the core's frequency faster than 500us could be applied or completely ignored, depending on when the register was sampled the previous time. This can cause all sort of average frequencies. Clearly, application phases which execute at lower frequency than the maximum one may lead to a slow-down in the application, while MPI phases which execute at higher frequency than the minimum one may lead to energy saving loss. It is nevertheless interesting to notice that phases with a duration in the region of 0s and 500us are more likely to have the highest frequency for the MPI phases and the lowest frequency for the application phases, which is opposite of what expected. This will be explained in the next analysis.

Thus, it is not possible to have an effective control on the frequency selection for phases shorter than 500us, while for longer phases is possible to recognize an asymptotic trend toward the requested frequency. The assumption is that in phases shorter than 500us the average frequency depends more on the previous phase frequency than the requested one.

Following this intuition, Figure 5.8 correlates the time duration of each application phase with the time duration of the following MPI phase and its average frequency. In the y-axis is reported the time duration of the application phase, in the x-axis the time duration of the subsequent MPI phase, and with the color code is reported the average frequency. While the left plot reports the average frequency of the MPI phase, instead the right plot reports the average frequency for the application phase. For both plots is possible to identify four regions/quadrants:

- **Application & MPI>500us**: this region contains long application phases followed by long MPI phases. Points in this region show low frequency in MPI phases and high frequency in application phases. This is the ideal behavior, where applying frequency scaling policy reduces energy waste in MPI but with no impact on the performance of the application. Phases in this region are perfect candidates for fine-grain DVFS policies.

- **Application>500us & MPI<500us**: this region contains long application phases followed by short MPI phases. Points in this region show for both application and MPI phases high average frequency. This is explained by the short duration of the MPI phases, which does not give enough time to the *HW power controller* to serve the request to scale down the frequency (*prologue*), before

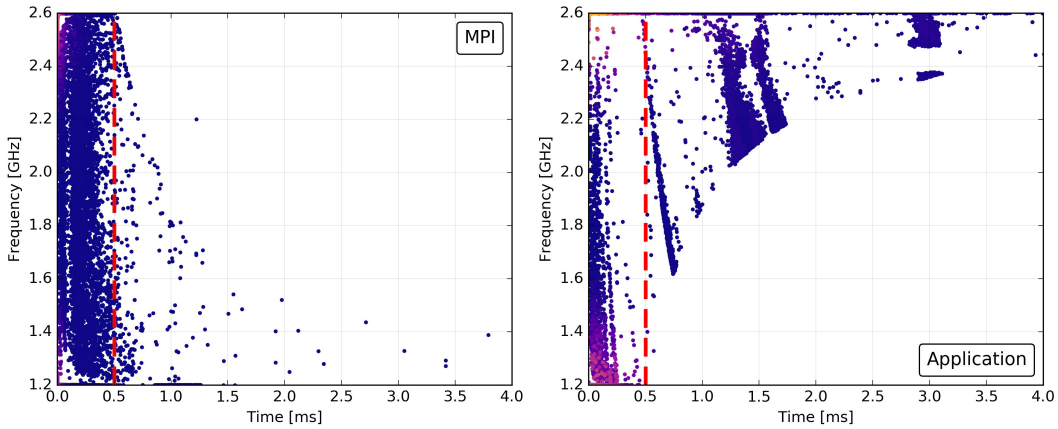


Figure 5.7: Average frequency and time duration of Application/MPI phases of the single node benchmark for QE-CP-EU. The lighter zones identify higher point density.

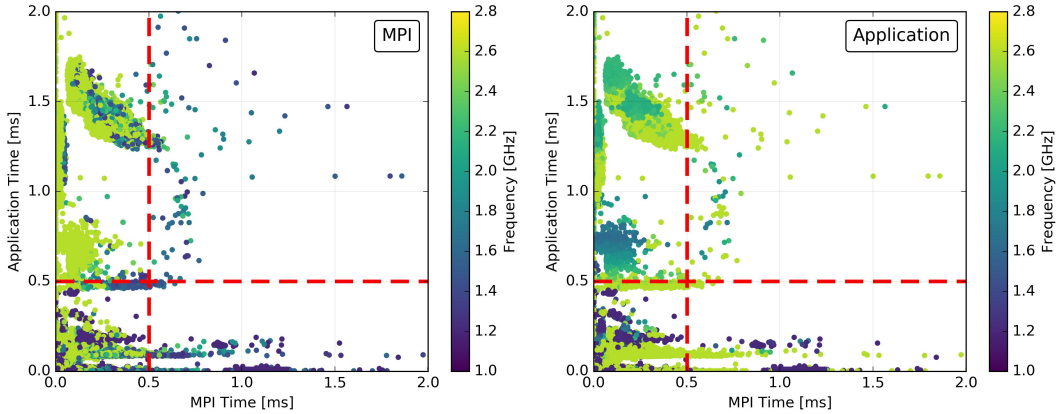


Figure 5.8: Time and average frequency of Application/MPI phases of the single node benchmark for QE-CP-EU.

this setting is overwritten by the request to operate at the highest frequency (*epilogue*).

For this reason, fine-grain DVFS control in this region does not save energy as the frequency reduction in MPI phases is negligible, but it also does not deteriorate the performance as the application phases are execute at the maximum frequency. Phases in this region should not be considered for fine-grain DVFS policies, being preferable to leave frequencies unaltered at the highest level.

- **Application<500us & MPI>500us**: this region contains short application phases followed with long MPI phases.

This is the opposite case of *Application>500us & MPI<500us* region. Points in this region show for both application and MPI phases low average frequencies. This is explained by the short duration of the application phases, which does not give enough time to the *HW power controller* to serve the request to raise up the frequency (requested at the exit of the previous MPI phase), before this setting gets overwritten by the request to operate at the lowest frequency (at the entrance of the following MPI phase).

Applying fine-grain DVFS policies in this region can save power but detriments the overall performance, as application phases are executed at low frequencies. Phases in this region should not be considered for fine-grain DVFS policies due to the high overheads in the application execution time.

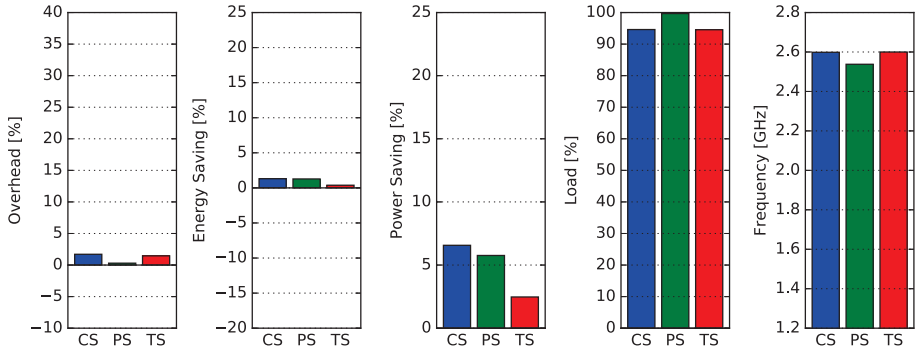
- **Application & MPI<500us**: This region shows the opposite behavior of *Application & MPI>500us* region. Both application and MPI phases executes randomly at high and low average frequencies due the inability of the *HW power controller* to capture and to serve the requested frequency changes. The average frequency at which MPI and application phases execute are strictly related to type of the previous long phase: if it was an application phase the following short phases will execute at high frequency in average; On the contrary, if it was an MPI phase the following short phases will execute at low frequency in average. Applying fine-grain DVFS policies in this region leads to unexpected behaviors which can detriment application performance. All phases shorter than 500us should be never considered by fine-grain power managers.

5.4.3 Single-node Evaluation

The experiments reported in Section 5.2 was repeated using COUNTDOWN. COUNTDOWN has been configured to scale down the P- and the T-states 500us after the prologues of MPI primitives. To reproduce the same timeout strategy leveraging the C-states, *MPI SPIN WAIT* described in Section 5.3.2 has been used with 10K MPI spincounter parameter.

The *HW power controller* of Intel CPUs, has a different transition latency for sleep states with respect to DVFS scaling, as described in [27]. For this reason, the best spin counter has been empirically determined to maximize the energy efficiency and

(a) All MPI processes diagonalization QE-CP-EU



(b) Single MPI process diagonalization QE-CP-NEU

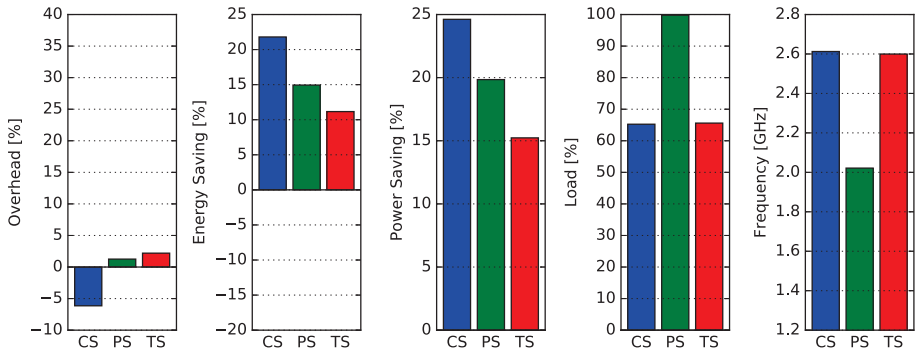


Figure 5.9: Overhead, energy/power saving, average load and frequency using COUNTDOWN for QE-CP-EU (a) and QE-CP-NEU (b). Legend: C-state (CS), P-state (PS) and T-state (TS) mode. Baseline is busy-waiting mode of MPI library.

to minimize the overhead for the target application.

Figure 5.9 report the experimental results using *COUNTDOWN THROTTLING*, *COUNTDOWN DVFS* and *MPI SPIN WAIT*. It can be seen that in all cases the overhead, the energy saving, and the power saving are greatly improved with respect to the baseline (only MPI library).

Figure 5.9.a shows the experimental results for QE-CP-EU. For the C-state mode the overhead decreases from 25.85% to 1.70% by using *MPI SPIN WAIT*. Instead using *COUNTDOWN DVFS* for the P-state the overhead decreases from 5.96% to a negligible overhead, and for the T-state from 5.96% to 0.29% using *COUNTDOWN THROTTLING*. All evaluations report a non-negative energy saving, as it was for the MPI library without timeout strategy, but with better results. Energy saving shows

21.80%, 14.94%, and 11.16% improvements and power saving reports 6.55%, 5.77%, and 2.47% respectively for C-state, P-state, and T-state. These experimental results confirm the exploration on the time duration of MPI phases reported in Figure 5.7. Most of the MPI calls of this benchmark have been skipped from COUNTDOWN due their short duration to avoid overheads.

Figure 5.9.b show similar improvements for QE-CP-NEU. In this configuration, for C-state mode the speed up increases from 1.08% to 6.14% using *MPI SPIN WAIT*. Instead using COUNTDOWN, the overhead of P-state decreases from 3.88% to 1.25%, and for the T-state from 15.82% to 2.19%. As a result, the energy saving is 21.80%, 14.94%, and 11.16% while power saving corresponds to 24.61%, 19.84% and 15.23% respectively for C-state, P-state, and T-state.

5.4.4 HPC Evaluation

After the exploration on a single compute node, this work has been extended in a real HPC system. The target architecture has been a Tier-1 HPC system based on an IBM NeXtScale cluster which is currently classified in the Top500 supercomputer list [4]. The compute nodes of the HPC system, are equipped with 2 Intel Broadwell E5-2697 v4 CPUs, with 18 cores at 2.3 GHz nominal clock speed with a peak power consumption of 145W and interconnected with an Intel QDR (40Gb/s) Infiniband high-performance network.

Two set of applications has been used to benchmark the parallel performances of the HPC system. The first one is the NAS parallel benchmark suite using the large dataset E. The NAS parallel benchmarks have been executed on 1024 cores distributed on 29 compute nodes. The second one is the QE PWscf software configured for a complex large-scale simulation. For this purpose, QE performs ten iterative steps of the self-consistent loop algorithm that optimizes the electronic density starting from the superposition of atomic charge densities. The selected system comes from an actual scientific report [80] and reproduces a layered structure of Iridium, Cobalt and Graphene plus a molecular compound (iron-phthalocyanin) deposited on top. The whole simulation box includes 662 atoms which are described with 3662 KS states. The total number of plane waves is more than a million and, during the execution, main memory occupation may peak at 2 to 6 terabytes depending on the selected parallelization parameters.

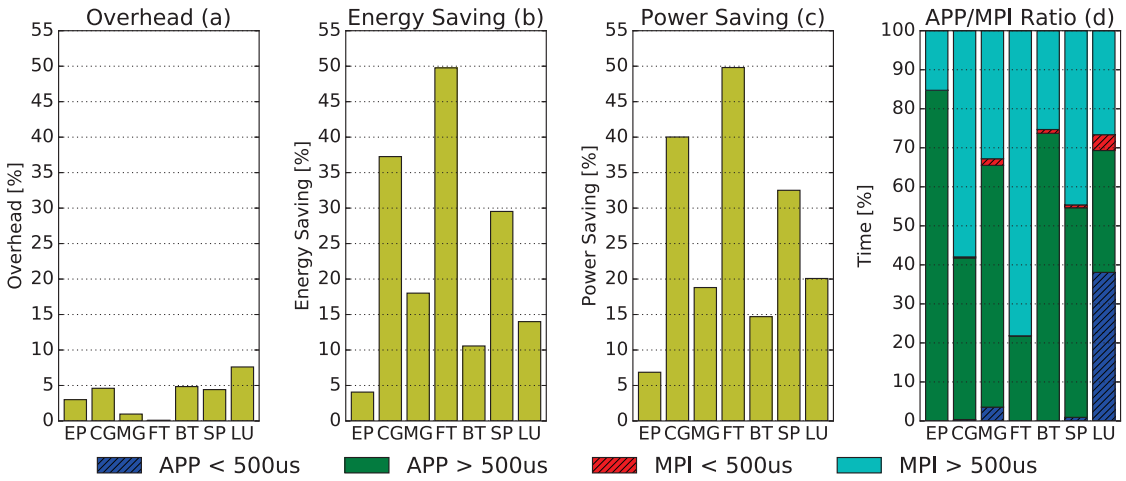


Figure 5.10: Experimental results of NAS parallel benchmarks using COUNTDOWN. Baseline is busy-waiting mode (default mode) MPI library.

During each iteration the CPU time is mostly spent in linear algebra (matrix-matrix multiplication and matrix diagonalization) and FFT. Both these operations are distributed on multiple processors and operate on distributed data. As a consequence, FFT requires many AllToAll MPI communications while parallel diagonalization, performed with the PDSYEVD subroutine of ScaLAPACK and requires mostly MPI broadcasting messages. QE run on 3456 cores, using 96 compute nodes and 12 TB of DRAM. The input dataset used for this run is capable to scale on such number of cores and the run has been configured with a set of optimized parameters to avoid network bottlenecks, which would limit the scalability. This configuration has been named QE Expert User (QE-PWscf-EU), to differentiate it from the same problem but solved without optimizing the internal parameter as it was run by a user without domain specific knowledge, which has been called QE Not Expert User (QE-PWscf-NEU).

In these tests, T-state mode has been excluded, because in the single-node evaluation it always reported worst results that the P-state mode. Also the C-state mode has been excluded because the Intel MPI library does not support idle mode when multiple nodes are involved in the run. The Intel MPI library overrides the request of idle mode with the busy-wait mode when the application runs in a distribute environment. For this reason, only P-state mode (*COUNTDOWN DVFS*) has been used in the HPC evaluation. The benchmarks has been run with and without COUNTDOWN on the same nodes and the results has been compared .

Figure 5.10 shows the results for the NAS parallel benchmark suite[76] when executed on 1024 cores, while Figure 5.11 shows the results for the QE-PWscf-* application when executed on 3456 cores. The different plots for Figure 5.10 reports the time-to-solution overhead, the energy and power saving as well as the MPI and application time phases distribution (in percentage of the the accumulated time spent in phases longer and shorter than 500us) for the different large-scale benchmarks and application runs. All the values are normalized against the default MPI busy waiting policy. The Figure 5.10.c shows that COUNTDOWN is capable of significantly cutting the energy consumption of the NAS parallel benchmarks from 6% to 50%. Figure 5.10.c shows that this savings follows the percentage of time the benchmark passes in MPI phases longer than 500us. From the overhead plot (Figure 5.10.a), it can be seen that all these energy savings happen with a very small time-to-solution overhead, in average below 5%. These results are very promising as they are virtually portable to any application, without the need to touch the application binary. When looking at the QE (QE-PWscf-*) case reported in Figure 5.11, COUNTDOWN attains similar results of the NAS also with real application production run optimized for scalability — COUNTDOWN saves the 22.36% of energy with an overhead of the 2.88% in the QE-PWscf-EU case —.

Figure 5.11.a shows the total time spent in application and in MPI phases which are shorter and longer than 500us for the QE-PWscf-EU case. On the x-axis, the Figure reports the Id of the MPI rank, while in the y-axis reports in percentage of the total time spent in phases longer and shorter than 500us. It can be immediately seen that in this real and optimized run, the application spends a negligible time in phases shorter than 500us. In addition, the time spent in the MPI library and in the application is not homogeneous among the MPI processes. This is an effect of the workload parameters chosen to optimize the communications, which distribute the workload in subsets of MPI processes to minimize broadcast and AlltoAll communications. Using this configuration, the experimental results report 2.88% of overhead with an energy saving of 22.36% and a power saving of 24.53% thanks to COUNTDOWN.

Figure 5.11.c shows that for the case QE-PWscf-NEU where the parameters are not optimized, all MPI processes have the same workload composition as they are part of same workgroup, and due the large overhead in the broadcast and AlltoAll communications, all the processes spend almost the 80% of the time in the MPI library. Even if it is suboptimal, this happens to HPC users running the application

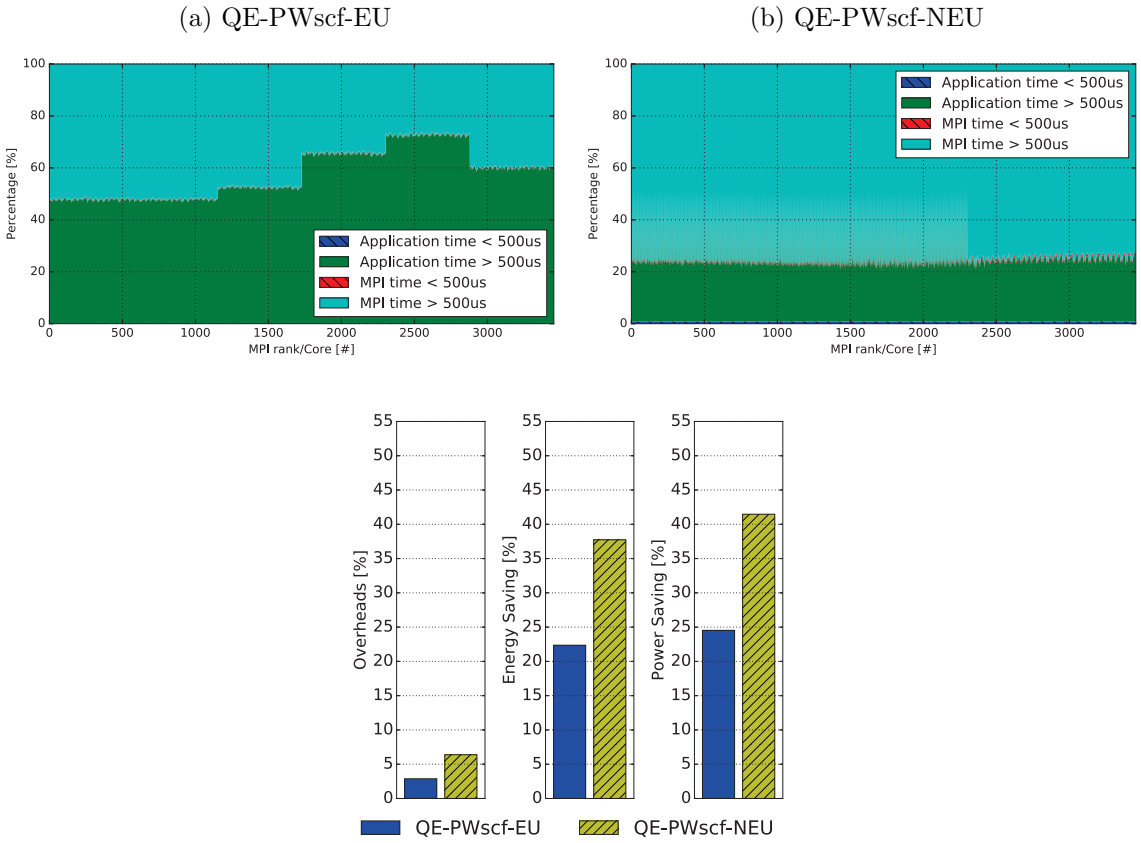


Figure 5.11: (a,b) Sum of the time spent in phases longer and shorter than 500us for QE-PWscf-EU and QE-PWscf-NEU.

without being domain experts or before tuning the execution parameters. This is a rather common scenario in scientific computing as only runs that are repeated multiple times are carefully optimized by domain experts.

In this situation, COUNTDOWN increases its benefits, reaching up to 37.74% of energy saving of and a power saving of 41.47%. In this condition, COUNTDOWN induces a small but relevant overhead of 6.38% suspecting that some MPI primitives suffer more than others from the frequency scaling. This problem will analyze in dept in future works aiming to guarantee that the COUNTDOWN overhead always remains negligible. However, an overhead below 10% is more than acceptable in many HPC facilities, especially when considering the massive energy savings.

In summary, the results achieved by COUNTDOWN in at production scale and application are very promising and if systematically adopted would dramatically reduce

the power cost of today supercomputers.

5.5 Summary on Energy-aware MPI Runtime

In this Chapter has been presented COUNTDOWN, a methodology and a tool for profiling HPC scientific applications and adding DVFS capabilities into standard MPI libraries. COUNTDOWN implements a timeout strategy to avoid application slowdown and for exploiting MPI communication slacks to drastically reduce energy consumption. COUNTDOWN has been demonstrated on real HPC systems and workloads and does not require any kind of modification to application source code nor to the compilation toolchain. The COUNTDOWN approach can leverage several low power state technologies — P/T/C states.

COUNTDOWN has been compared with state-of-the-art power management approaches for MPI libraries, which can dynamically control idle and DVFS levels for MPI-based application. The experimental results show that using the tuned timeout strategy to take decisions on power control can drastically reduce overheads, maximizing the energy efficient in small and large MPI communications. COUNTDOWN can lead up to 14.94% energy saving and 19.84% of power saving with a less than 1.5% performance penalty on a single-compute node. However, the benefits of COUNTDOWN increase with the scale of the application. In NAS parallel benchmark runs, using 1K cores, COUNTDOWN always saves energy with a saving that depends on the application. It ranges from 6% to 50% at a negligible overhead (below 6%). In a full-scale production run of QE on more than 3.4K cores, COUNTDOWN saves 22.36% of energy with only 2.88% performance overhead. Energy reduction reaches 37.74% when the application is executed with a default conservative parallelization setting.

COUNTDOWN is an effective, non-intrusive and low overhead approach to cut today's supercomputing center energy-consumption transparently to the user. In future work, COUNTDOWN would be integrated within standard power management infrastructure, such as GEOPM [22], and to complement it with predictive and application-driven power management techniques.

Chapter 6

Conclusion

In Chapter 1 has been surveyed the main challenges for the supercomputers in the exaflops era. These can be briefly summarized as follow:

- It is well established that the pace dictated by the Moore's law on technological scaling come at the cost of increasing power consumption and leads to thermally-bound computing systems. Supercomputers as well as data centres are on the cutting edge of this crisis, because of aggressive performance, integration density and sustainable power budget [2, 3].

- For supercomputers ensuring a power budget at design time it is crucial to avoid power outages, but this require to consider the worst-case power consumption of the system. However, this approach does not consider that supercomputers rarely cause worst-case power consumption during their life making this power design a bad choice. Instead to packing more computing resources in power-limited systems, designers should target the "the average power consumption case" and reducing the system's performance in the peaks of power consumption. This is today possible using HW components implemented in recent CPUs, which allow fine-grain tuning of power consumption. While the utilization of these HW mechanisms is straightforward, they can induce performance penalties due the unawareness of application workload and to their fast reaction on the power consumption changes.

- As the side effect of the end of Dennard's scaling, the power density of new generation CPUs have started to grow with every technologically scaling step. High-power densities need aggressive cooling solutions to maintain a safe-working

temperature and to avoid overheating situations which creates reliability problems. Furthermore, with the continuous growth of the number of computing unit integrated on same die, high-end processors started to show significant thermal gradients and heterogeneity [48] which can cause performance unbalance and degradation in application workloads.

- In the supercomputing ecosystem, the aggressive race of performance scaling shown in the last decades in Top500 list [4], always ignored the power consumption in the design of new generation of supercomputing systems. Today, this is not more sustainable because supercomputing facilities reach their practical limit in power provisioning, which is around 20MWatts [7]. HPC vendors and scientific community needs to take care about efficiency in HPC systems in terms of energy consumption.

The results presented in this thesis make it clear that propagating application requirements to the operating system and runtime library is a powerful technique to improve energy efficiency and thermal control of computing systems. Furthermore, power and thermal control policy should embedded in the middle-ware and runtime of the system leaving the users and developers focusing only on the functionality and on the performance of the applications. The software mechanisms developed in this thesis follow the above direction, they are completely application agnostic and prove that can be very effective even without modify the application's code.

In detail, this thesis targets the challenges listed above:

- In Chapter 3 has been explored the characteristics of the power management of Intel architecture and it has shown that managing the available power budget without be aware of the application workloads is not beneficial from a performance point of view. To prove it, the application workloads the impact has been analyzed and explained using different power capping strategies and it has been shown how the execution time can vary using different mechanisms.

- While in Chapter 4 has been targeted energy and thermal management policies. Differently from state-of-the-art solutions, the analysis has focused on a real supercomputing system from which has been modeled the real thermal and power characteristics as well as extracted real scientific workload traces. A

thermal-aware allocation scheduler has been developed that guarantees a safe working temperature while it maximizes performance on critical MPI tasks. Different configurations has been explored to find-out the best performance point in thermally-bounded systems.

- In Chapter 5, COUNTDOWN has been presented as a methodology and a tool for profiling HPC scientific applications and injecting DVFS capabilities into standard scientific applications. COUNTDOWN implements a timeout strategy to avoid costly performance overheads and leveraging on communication slacks to drastically reduce energy consumption. COUNTDOWN targets real HPC systems and workloads and does not require any kind of modification to the source code nor to the compilation toolchain of the application. Our experimental results show that using timeout strategy to take decisions on power control can drastically reduce overheads maximizing the energy efficient in small and large MPI communications. COUNTDOWN is able to avoid high overheads induced by short MPI calls where the *HW power manager* is not fast enough to react to high-frequency requests.

The contribution of this thesis shapes the energy efficiency and thermal control in a significant way respect to the state-of-the-art works. It would contribute in the energy efficiency and thermal management field for HPC systems ready to enter in the exaflops era.

Glossary

10s-10s A prediction horizon of 10s for the FSP problem and a prediction horizon of 10s for the ISP problem. 63, 64

1s-1s A prediction horizon of 1s for the FSP problem and a prediction horizon of 1s for the ISP problem. 59, 63, 64

ABI Application Binary Interface. 73

ACPI Advanced Configuration and Power Interface. 30

API Application Programming Interface. 22, 35, 77

APP Application. 39

CMOS Complementary metal-oxide semiconductor. 15

CP Car-Parrinello. 34, 36, 39, 47, 48, 57, 62

CPU Central Processing Unit. 17, 19, 22, 23, 24, 26, 27, 29, 30, 32, 33, 34, 35, 45, 46, 66, 68, 70, 72, 76, 78, 81, 84, 86, 91

C-state Idle power saving state. 11, 19, 26, 45, 58, 69, 70, 71, 76, 79, 81, 84, 85, 86, 87

CPI Cycles Per Instruction. 10, 35, 37, 39, 40, 42

DCT Dynamic Concurrency Throttling. 27

DRAM Dynamic Random Access Memory. 33, 35, 86

DTM Dynamic Thermal Management. 17, 18, 44, 59, 65

DVFS Dynamic and Voltage Frequency States. 16, 17, 19, 23, 27, 30, 44, 45, 46, 58, 65, 67, 70, 72, 79, 81, 82, 83, 84, 90

EAW Energy-aware MPI wrapper active. 51, 59

FF Fixed Frequency. 10, 37, 38, 37, 39, 40, 42

FFT Fast Fourier Transform. 34, 68, 86

FPGA Field Programmable Gate Array. 29

FSP First Step Problem. 53, 55, 56, 57, 59, 62, 63, 64

GEOPM Global Extensible Open Power Manager. 22, 27

GPGPU General-Purpose computing on Graphics Processing Units. 29

HPC High-Performance Computing. 5, 10, 13, 17, 18, 19, 20, 21, 23, 24, 25, 29, 30, 34, 36, 42, 44, 45, 47, 49, 50, 52, 57, 65, 66, 67, 68, 72, 78, 86, 87, 88, 89, 90, 92, 93

HW Hardware. 16, 17, 20, 22, 29, 30, 32, 72, 74, 76, 77, 81, 82, 83, 84, 91, 93

ICC Intel C language Compiler. 34, 68

IFORT Intel FORTRAN language Compiler. 34, 68

ILP Integer Linear Programing. 24, 44, 49, 50, 54, 65

IO Input/Output. 27, 34, 79

ISP i-th Step Problem. 55, 56, 57, 59, 62, 63, 64

KS Kohn and Sham. 68, 86

LA Linear Algebra. 34

LUT LookUp-Table. 46

MPI Message Passing Interface. 10, 11, 18, 19, 20, 23, 25, 26, 27, 30, 34, 35, 36, 39, 42, 44, 47, 48, 49, 50, 51, 52, 53, 57, 58, 57, 58, 59, 61, 62, 65, 66, 67, 68, 69, 70, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 81, 82, 83, 84, 85, 87, 86, 87, 88, 89, 90, 92, 93

MSR Model-Specific Register. 32, 35

NoEAW No energy-aware MPI wrappe active. 59

NoTMC No thermal control active. 59

OpenMP Open Multiprocessing. 27, 68

OS Operating System. 17, 19, 20, 31, 36, 43, 44, 45, 70

OTC Optimal Thermal Controller. 49, 50, 65

PE Processing Element. 30

PMPI Profile Message Passing Interface. 35, 74

PMU Performance Monitoring Unit. 35

P-state Performance State. 11, 16, 19, 25, 26, 27, 30, 31, 32, 45, 69, 70, 77, 79, 81, 84, 85, 86, 87

PWscf Plane-Wave Self-Consistent Field. 68, 86

QE QuantumESPRESSO. 34, 36, 39, 47, 48, 57, 58, 62, 67, 68, 69, 86, 87

QE-PWscf-NEU QuantumESPRESSO - Plane-Wave Self-Consistent Field - Not Expert User. 12, 86, 88

QE-PWscf-EU QuantumESPRESSO - Plane-Wave Self-Consistent Field - Expert User. 12, 86, 87, 88

QE-CP-NEU QuantumESPRESSO - Car-Parrinello - Not Expert User. 11, 68, 69, 72, 73, 79, 84, 86

QE-CP-EU QuantumESPRESSO - Car-Parrinello - Expert User. 11, 68, 69, 71, 72, 78, 79, 82, 84, 85

RAPL Running Average Power Limits. 10, 17, 22, 29, 32, 33, 32, 33, 35, 36, 37, 38, 37, 39, 40, 42, 43, 76

RC Resistor-Capacitor. 46

SIMD Single Instruction, Multiple Data. 10, 37, 39, 40, 42

SMP Symmetric MultiProcessor. 34

SPMD Single Program, Multiple Data. 30

SS-100s Static allocation for the FSP problem and a prediction horizon of 100s for the ISP problem. 63

SS-10s Static allocation for the FSP problem and a prediction horizon of 10s for the ISP problem. 63, 64

SS-1s Static allocation for the FSP problem and a prediction horizon of 1s for the ISP problem. 59, 63

SS-SS Static allocation for both optimization problems. 59, 63, 64

SW Software. 20

TCO Total Cost of Ownership. 16, 25

TDP Thermal Design Power. 17

TSC Time Stamp Counter. 35, 76

T-state Throttling State. 11, 19, 25, 26, 67, 69, 72, 73, 77, 79, 81, 84, 85, 86, 87

TMC Thermal-aware task Mapper and Controller. 10, 50, 53, 55, 56, 57, 59, 60, 61, 60

Tts Time to Solution. 19, 24, 25, 26, 29, 44, 64, 70, 72

Bibliography

[1] M. Horowitz, E. Alon, D. Patil, S. Naffziger, R. Kumar, and K. Bernstein, "Scaling, power, and the future of cmos," in *Electron Devices Meeting, 2005. IEDM Technical Digest. IEEE International*. IEEE, 2005, pp. 7–pp.

[2] L. B. Kish, "End of moore's law: thermal (noise) death of integration in micro and nano electronics," *Physics Letters A*, vol. 305, no. 3, pp. 144–149, 2002.

[3] M. K. Patterson, "The effect of data center temperature on energy efficiency," in *Thermal and Thermomechanical Phenomena in Electronic Systems, 2008. ITHERM 2008. 11th Intersociety Conference on*. IEEE, 2008, pp. 1167–1174.

[4] "Top500.org. top 500 supercomputer sites," http://www.top500.org, 2018.

[5] W.-c. Feng and K. Cameron, "The green500 list: Encouraging sustainable supercomputing," *Computer*, vol. 40, no. 12, 2007.

[6] J. Dongarra, "Visit to the national university for defense technology changsha, china," *Oak Ridge National Laboratory, Tech. Rep., June*, 2013.

[7] K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, J. Hiller *et al.*, "Exascale computing study: Technology challenges in achieving exascale systems," *Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech. Rep*, vol. 15, 2008.

[8] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, and L. Benini, "Scheduling-based power capping in high performance computing systems," *Sustainable Computing: Informatics and Systems*, 2018.

[9] E. J. Hogbin, "Acpi: Advanced configuration and power interface," 2015.

[10] H. Chen, M. C. Caramanis, and A. K. Coskun, "Reducing the data center electricity costs through participation in smart grid programs," in *Green Computing Conference (IGCC), 2014 International*. IEEE, 2014, pp. 1–10.

[11] ——, "The data center as a grid load stabilizer," in *Design Automation Conference (ASP-DAC), 2014 19th Asia and South Pacific*. IEEE, 2014, pp. 105–112.

[12] P. Hammarlund, R. Kumar, R. B. Osborne, R. Rajwar, R. Singhal, R. D'Sa, R. Chappell, S. Kaushik, S. Chennupaty, S. Jourdan *et al.*, "Haswell: The fourth-generation intel core processor," *IEEE Micro*, vol. 34, no. 2, pp. 6–20, 2014.

[13] K. A. Huck and J. Labarta, "Detailed load balance analysis of large scale parallel applications," in *Parallel Processing (ICPP), 2010 39th International Conference on*. IEEE, 2010, pp. 535–544.

[14] O. Pearce, T. Gamblin, B. R. de Supinski, M. Schulz, and N. M. Amato, "Quantifying the effectiveness of load balance algorithms," in *Proceedings of the 26th ACM International Conference on Supercomputing*, ser. ICS '12. New York, NY, USA: ACM, 2012, pp. 185–194. [Online]. Available: http://doi.acm.org/10.1145/2304576.2304601

[15] R. Ayoub, S. Sharifi, and T. S. Rosing, "Gentlecool: Cooling aware proactive workload scheduling in multi-machine systems," in *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2010, pp. 295–298.

[16] A. K. Coşkun, K. Whisnant, K. C. Gross *et al.*, "Static and dynamic temperature-aware scheduling for multiprocessor SoCs," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 16, no. 9, pp. 1127–1140, 2008.

[17] H. Khdr, S. Pagani, M. Shafique, and J. Henkel, "Thermal constrained resource management for mixed ilp-tlp workloads in dark silicon chips," in *Proceedings of the 52nd Annual Design Automation Conference*. ACM, 2015, p. 179.

[18] H. Khdr, S. Pagani, E. Sousa, V. Lari, A. Pathania, F. Hannig, M. Shafique, J. Teich, and J. Henkel, "Power density-aware resource management for heterogeneous tiled multicores," *IEEE Transactions on Computers*, vol. 66, no. 3, pp. 488–501, 2017.

[19] F. Fraternali, A. Bartolini, C. Cavazzoni, G. Tecchiolli, and L. Benini, "Quantifying the impact of variability on the energy efficiency for a next-generation ultra-green supercomputer," in *Proceedings of the 2014 international symposium on Low power electronics and design*. ACM, 2014, pp. 295–298.

[20] A. Auweter, A. Bode, M. Brehm, L. Brochard, N. Hammer, H. Huber, R. Panda, F. Thomas, and T. Wilde, "A case study of energy aware scheduling on supermuc," in *International Supercomputing conference*. Springer, 2014, pp. 394–409.

[21] C.-h. Hsu and W.-c. Feng, "A power-aware run-time system for high-performance computing," in *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*. IEEE Computer Society, 2005, p. 1.

[22] J. Eastep, S. Sylvester, C. Cantalupo, F. Ardanaz, B. Geltz, A. Al-Rawi, F. Keceli, and K. Livingston, "Global extensible open power manager: a vehicle for hpc community collaboration toward co-designed energy management solutions," *Supercomputing PMBS*, 2016.

[23] B. Rountree, D. K. Lownenthal, B. R. De Supinski, M. Schulz, V. W. Freeh, and T. Bletsch, "Adagio: making dvs practical for complex hpc applications," in *Proceedings of the 23rd international conference on Supercomputing*. ACM, 2009, pp. 460–469.

[24] D. Li, B. R. de Supinski, M. Schulz, K. Cameron, and D. S. Nikolopoulos, "Hybrid mpi/openmp power-aware computing," in *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 1–12.

[25] F. Fraternali, A. Bartolini, C. Cavazzoni, and L. Benini, "Quantifying the impact of variability and heterogeneity on the energy efficiency for a next-generation ultra-green supercomputer," *IEEE Transactions on Parallel and Distributed Systems*, pp. 1–1, 2017.

[26] V. W. Freeh, N. Kappiah, D. K. Lowenthal, and T. K. Bletsch, "Just-in-time dynamic voltage scaling: Exploiting inter-node slack to save energy in mpi programs," *Journal of Parallel and Distributed Computing*, vol. 68, no. 9, pp. 1175–1185, 2008.

[27] D. Hackenberg, R. Schone, T. Ilsche, D. Molka, J. Schuchart, and R. Geyer, "An energy efficiency feature survey of the intel haswell processor," in *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*. IEEE, may 2015. [Online]. Available: https://doi.org/10.1109/ipdpsw.2015.70

[28] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, and L. Benini, "Predictive modeling for job power consumption in hpc systems," in *International Conference on High Performance Computing*. Springer, 2016, pp. 181–199.

[29] A. Sîrbu and O. Babaoglu, "Predicting system-level power for a hybrid supercomputer," in *High Performance Computing & Simulation (HPCS), 2016 International Conference on*. IEEE, 2016, pp. 826–833.

[30] A. Borghesi, C. Conficoni, M. Lombardi, and A. Bartolini, "Ms3: a mediterranean-stile job scheduler for supercomputers-do less when it's too hot!" in *High Performance Computing & Simulation (HPCS), 2015 International Conference on*. IEEE, 2015, pp. 88–95.

[31] A. Marathe, P. E. Bailey, D. K. Lowenthal, B. Rountree, M. Schulz, and B. R. de Supinski, "A run-time system for power-constrained hpc applications," in *International Conference on High Performance Computing*. Springer, 2015, pp. 394–408.

[32] O. Sarood, A. Langer, A. Gupta, and L. Kale, "Maximizing throughput of overprovisioned hpc data centers under a strict power budget," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 2014, pp. 807–818.

[33] N. Gholkar, F. Mueller, and B. Rountree, "Power tuning hpc jobs on power-constrained systems," in *Proceedings of the 2016 International Conference on Parallel Architectures and Compilation*. ACM, 2016, pp. 179–191.

[34] H. Zhang and H. Hoffman, "A quantitative evaluation of the rapl power control system," *Feedback Computing*, 2015.

[35] A. K. Coskun, T. S. Rosing, and K. Whisnant, "Temperature aware task scheduling in mpsocs," in *Proceedings of the conference on Design, automation and test in Europe*. EDA Consortium, 2007, pp. 1659–1664.

[36] A. Bartolini, M. Cacciari, A. Tilli, and L. Benini, "A distributed and self-calibrating model-predictive controller for energy and thermal management of high-performance multicores," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, March 2011, pp. 1–6.

[37] A. K. Coskun, T. S. Rosing, and K. C. Gross, "Utilizing predictors for efficient thermal management in multiprocessor socs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 10, pp. 1503–1516, 2009.

[38] F. Zanini, D. Atienza, L. Benini, and G. D. Micheli, "Thermal-aware system-level modeling and management for multi-processor systems-on-chip," in *Circuits and Systems (ISCAS), 2011 IEEE International Symposium on*, May 2011, pp. 2481–2484.

[39] D. Puschini, F. Clermidy, P. Benoit, G. Sassatelli, and L. Torres, "Temperature-aware distributed run-time optimization on mp-soc using game theory," in *Symposium on VLSI, 2008. ISVLSI'08. IEEE Computer Society Annual.* IEEE, 2008, pp. 375–380.

[40] S. Murali, A. Mutapcic, D. Atienza, R. Gupta, S. Boyd, and G. D. Micheli, "Temperature-aware processor frequency assignment for mpsocs using convex optimization," in *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2007 5th IEEE/ACM/IFIP International Conference on*, Sept 2007, pp. 111–116.

[41] Y. Xie and W.-L. Hung, "Temperature-aware task allocation and scheduling for embedded multiprocessor systems-on-chip (mpsoc) design," *The Journal of VLSI Signal Processing*, vol. 45, no. 3, pp. 177–189, 2006.

[42] Q. Xie, M. J. Dousti, and M. Pedram, "Therminator: A thermal simulator for smartphones producing accurate chip and skin temperature maps," in *Low Power Electronics and Design (ISLPED), 2014 IEEE/ACM International Symposium on*, Aug 2014, pp. 117–122.

[43] C. Conficoni, A. Bartolini, A. Tilli, G. Tecchiolli, and L. Benini, "Energy-aware cooling for hot-water cooled supercomputers," in *Proceedings of the 2015 Design,*

Automation & Test in Europe Conference & Exhibition, ser. DATE '15. San Jose, CA, USA: EDA Consortium, 2015, pp. 1353–1358. [Online]. Available: http://dl.acm.org/citation.cfm?id=2757012.2757127

[44] M. Maiterth, T. Wilde, D. Lowenthal, B. Rountree, M. Schulz, J. Eastep, and D. Kranzlmiiller, "Power aware high performance computing: Challenges and opportunities for application and system developers x2014; survey tutorial," in *2017 International Conference on High Performance Computing Simulation (HPCS)*, July 2017, pp. 3–10.

[45] D. Cesarini, A. Bartolini, and L. Benini, "Benefits in relaxing the power capping constraint," in *Proceedings of the 1st Workshop on AutotuniNg and aDaptivity AppRoaches for Energy Efficient HPC Systems*, ser. ANDARE '17. New York, NY, USA: ACM, 2017, pp. 3:1–3:6. [Online]. Available: http://doi.acm.org/10.1145/3152821.3152878

[46] Z. Wang, C. Bash, N. Tolia, M. Marwah, X. Zhu, and P. Ranganathan, "Optimal fan speed control for thermal management of servers," in *ASME 2009 InterPACK Conference collocated with the ASME 2009 Summer Heat Transfer Conference and the ASME 2009 3rd International Conference on Energy Sustainability*. American Society of Mechanical Engineers, 2009, pp. 709–719.

[47] F. Beneventi, A. Bartolini, A. Tilli, and L. Benini, "An effective gray-box identification procedure for multicore thermal modeling," *IEEE Transactions on Computers*, vol. 63, no. 5, pp. 1097–1110, May 2014.

[48] F. Beneventi, A. Bartolini, C. Cavazzoni, and L. Benini, "Cooling-aware node-level task allocation for next-generation green hpc systems," *management*, vol. 1, p. 6, 2016.

[49] V. Hanumaiah, S. Vrudhula, and K. S. Chatha, "Performance optimal speed control of multi-core processors under thermal constraints," in *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, April 2009, pp. 1548–1551.

[50] A. Rudi, A. Bartolini, A. Lodi, and L. Benini, "Optimum: Thermal-aware task

allocation for heterogeneous many-core devices," in *High Performance Computing Simulation (HPCS), 2014 International Conference on*, July 2014, pp. 82–87.

[51] A. Borghesi, A. Bartolini, M. Milano, and L. Benini, "Pricing schemes for energy-efficient hpc systems: Design and exploration," *arXiv preprint arXiv:1806.05135*, 2018.

[52] V. Sundriyal, M. Sosonkina, and A. Gaenko, "Energy efficient communications in quantum chemistry applications," *Computer Science-Research and Development*, vol. 29, no. 2, pp. 149–158, 2014.

[53] V. Sundriyal and M. Sosonkina, "Per-call energy saving strategies in all-to-all communications," in *European MPI Users' Group Meeting*. Springer, 2011, pp. 188–197.

[54] V. Sundriyal, M. Sosonkina, and Z. Zhang, "Achieving energy efficiency during collective communications," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 15, pp. 2140–2156, 2013.

[55] V. Sundriyal, M. Sosonkina, and A. Gaenko, "Runtime procedure for energy savings in applications with point-to-point communications," in *Computer Architecture and High Performance Computing (SBAC-PAD), 2012 IEEE 24th International Symposium on*. IEEE, 2012, pp. 155–162.

[56] B. Rountree, D. K. Lowenthal, S. Funk, V. W. Freeh, B. R. de Supinski, and M. Schulz, "Bounding energy consumption in large-scale mpi programs," in *Supercomputing, 2007. SC '07. Proceedings of the 2007 ACM/IEEE Conference on*, Nov 2007, pp. 1–9.

[57] C. Liu, A. Sivasubramaniam, M. Kandemir, and M. J. Irwin, "Exploiting barriers to optimize power consumption of cmps," in *19th IEEE International Parallel and Distributed Processing Symposium*, April 2005, pp. 5a–5a.

[58] S. Bhalachandra, A. Porterfield, S. L. Olivier, and J. F. Prins, "An adaptive core-specific runtime for energy efficiency," in *Parallel and Distributed Processing Symposium (IPDPS), 2017 IEEE International*. IEEE, 2017, pp. 947–956.

[59] M. Y. Lim, V. W. Freeh, and D. K. Lowenthal, "Adaptive, transparent frequency and voltage scaling of communication phases in mpi programs," in *SC 2006 conference, proceedings of the ACM/IEEE*. IEEE, 2006, pp. 14–14.

[60] J. Li, J. F. Martinez, and M. C. Huang, "The thrifty barrier: Energy-aware synchronization in shared-memory multiprocessors," in *null*. IEEE, 2004, p. 14.

[61] A. Venkatesh, A. Vishnu, K. Hamidouche, N. Tallent, D. Panda, D. Kerbyson, and A. Hoisie, "A case for application-oblivious energy-efficient mpi runtime," in *High Performance Computing, Networking, Storage and Analysis, 2015 SC-International Conference for*. IEEE, 2015, pp. 1–12.

[62] D. J. Kerbyson, A. Vishnu, and K. J. Barker, "Energy templates: Exploiting application information to save energy," in *2011 IEEE International Conference on Cluster Computing*. IEEE, 2011, pp. 225–233.

[63] L. Benini, A. Bogliolo, and G. D. Micheli, "A survey of design techniques for system-level dynamic power management," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 8, no. 3, pp. 299–316, June 2000.

[64] V. Pallipadi and A. Starikovskiy, "The ondemand governor," in *Proceedings of the Linux Symposium*, vol. 2, no. 00216. sn, 2006, pp. 215–230.

[65] P. Giannozzi, S. Baroni, N. Bonini, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, G. L. Chiarotti, M. Cococcioni, I. Dabo *et al.*, "Quantum espresso: a modular and open-source software project for quantum simulations of materials," *Journal of physics: Condensed matter*, vol. 21, no. 39, p. 395502, 2009.

[66] F. Beneventi, A. Bartolini, C. Cavazzoni, and L. Benini, "Continuous learning of hpc infrastructure models using big data analytics and in-memory processing tools," in *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2017, pp. 1038–1043.

[67] P. Mercati, A. Bartolini, F. Paterna, T. S. Rosing, and L. Benini, "A linux-governor based dynamic reliability manager for android mobile devices," in *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*. IEEE, 2014, pp. 1–4.

[68] P. Mercati, F. Paterna, A. Bartolini, L. Benini, and T. Š. Rosing, "Warm: Workload-aware reliability management in linux/android," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 9, pp. 1557–1570, 2017.

[69] D. Cesarini, A. Bartolini, and L. Benini, "Prediction horizon vs. efficiency of optimal dynamic thermal control policies in hpc nodes," in *2017 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, Oct 2017, pp. 1–6.

[70] B. A. CESARINI, Daniele and L. BENINI, "Energy saving and thermal management opportunities in a workload-aware mpi runtime for a scientific hpc computing node," *Parallel Computing is Everywhere*, vol. 32, p. 277, 2018.

[71] C. Silvano, G. Palermo, G. Agosta, A. H. Ashouri, D. Gadioli, S. Cherubin, E. Vitali, L. Benini, A. Bartolini, D. Cesarini *et al.*, "Autotuning and adaptivity in energy efficient hpc systems: The antarex toolbox," in *Proceedings of the 15th ACM International Conference on Computing Frontiers*. ACM, 2018, pp. 270–275.

[72] A. Bartolini, R. Diversi, D. Cesarini, and F. Beneventi, "Self-aware thermal management for high performance computing processors," *IEEE Design & Test*, 2017.

[73] A. Bartolini, M. Cacciari, C. Cavazzoni, G. Tecchiolli, and L. Benini, "Unveiling eurora - thermal and power characterization of the most energy-efficient supercomputer in the world," in *Proceedings of the Conference on Design, Automation & Test in Europe*, ser. DATE '14. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2014, pp. 277:1–277:6. [Online]. Available: http://dl.acm.org/citation.cfm?id=2616606.2617013

[74] Intel, "Intel trace analyzer and collector," *ITAC User's Manual*, 2017.

[75] IBM, "Ibm ilog cplex 12.6," *CPLEX User's Manual*, 2014.

[76] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber *et al.*, "The nas parallel benchmarks-summary and preliminary results," in *Proceedings of the 1991 ACM/IEEE conference on Supercomputing*. ACM, 1991, pp. 158–165.

[77] D. Cesarini, A. Bartolini, P. Bonfà, C. Cavazzoni, and L. Benini, "Countdown-three, two, one, low power! a run-time library for energy saving in mpi communication primitives," *arXiv preprint arXiv:1806.07258*, 2018.

[78] S. Bhalachandra, A. Porterfield, and J. F. Prins, "Using dynamic duty cycle modulation to improve energy efficiency in high performance computing," in *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*. IEEE, may 2015. [Online]. Available: https://doi.org/10.1109/ipdpsw.2015.144

[79] K. SHOGA and B. ROUNTREE, "Github scalability-llnl/msr-safe, 2014."

[80] G. Avvisati, S. Lisi, P. Gargiani, A. Della Pia, O. De Luca, D. Pacilé, C. Cardoso, D. Varsano, D. Prezzi, A. Ferretti *et al.*, "Fepc adsorption on the moiré superstructure of graphene intercalated with a cobalt layer," *The Journal of Physical Chemistry C*, vol. 121, no. 3, pp. 1639–1647, 2017.

I hereby confirm that all of the above accepted publications and patents have been already included in the database (IRIS) of the University of Bologna and that I have a corresponding ORCID.