Resource Management for Multicores to Optimize Performance under Temperature and Aging Constraints

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Heba Khdr

aus Aleppo, Syrien

Tag der mündlichen Prüfung:	04.07.2018
Erster Gutachter:	Prof. Dr. Jörg Henkel Karlsruher Instituts für Technologie (KIT)
Zweiter Gutachter:	Prof. Dr. Jürgen Teich Friedrich-Alexander Universität (FAU)

Hiermit erkläre ich an Eides statt, dass ich die von mir vorgelegte Arbeit selbstständig verfasst habe, dass ich die verwendeten Quellen, Internet-Quellen und Hilfsmittel vollständig angegeben haben und dass ich die Stellen der Arbeit - einschlielich Tabellen, Karten und Abbildungen - die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

_____

Heba Khdr

Acknowledgements

First and foremost, I would like to thank God for giving me the strength, knowledge, ability and opportunity to undertake this doctoral research and complete it successfully. I could never have done this without the faith I have in God.

I have great pleasure to express my sincere and deep sense of gratitude to my advisor Prof. Henkel, for his advice, support and exemplary guidance that have led me here. I am thankful for his invaluable experience that he shared with me to help me overcome many obstacles towards accomplishing this work. Many thanks go out to my friends and my colleagues that have accompanied me through the previous years. I really appreciate all the help they have given me. I would also like to extend my gratitude to my co-advisor Prof. Teich for agreeing to co-advise my dissertation and for his valuable input and support.

This achievement would never be done without the continued support of my family. I would like to express my utmost heartfelt gratitude to my parents. My mother, the inexhaustible source of love and tenderness, who has been continuously supporting me, taking care of me, and praying for me since ever. My affectionate father, whose dream for me to be Dr. Heba Khdr is my strongest motivation to start, continue, and complete my PhD. I am deeply indebted to my beloved husband, Ahmad, who has been by my side encouraging me through this journey, helping me during sleepless nights. Without his caring and his love, this would not have been possible. My warm thanks to my dear sister and brother who helped me a lot taking care of my sons, bringing them to and from their kindergarten and school to let me work peacefully. I won't forget to thank my lovely sons, Mohamad und Abdullah, for their patience with me being far away from them for long nights working on my research. Their smiles was accompanying me during my work, brightening up my day, motivating me to complete my PhD successfully. Last but not least, I would like to thank my late grandfather and my late mother in-law, whose prayers for me gave me the strength to achieve my goal. My dear family! I will remain grateful to all of you forever. Thank you!

List of First-Author Publications

[**1**] Heba Khdr, Hussam Amrouch and Jörg Henkel, "Aging-Aware Boosting," *IEEE Transactions on Computers (TC)*, 67(9), 1217-1230, September 2018.

[**2**] Heba Khdr, Hussam Amrouch and Jörg Henkel, "Dynamic Guardband Selection: Thermal-Aware Optimization for Unreliable Multi-Core Systems," *IEEE Transactions on Computers (TC)*, 68(1). 53-66, June 2018.

[**3**] Heba Khdr, Santiago Pagani, Muhammad Shafique, Jörg Henkel, "Dark-Silicon-Aware Resource Management for Many-Core Systems," *Advances in Computers: Dark Silicon and Future of On-chip Systems*, Elsevier, Vol. 110, pp. 127-170, January 2018.

[**4**] Heba Khdr, Hussam Amrouch and Jörg Henkel, "Aging-Constrained Performance Optimization for Multi Cores," *ACM/EDAC/IEEE 55th Design Automation Conference (DAC)*, (pp. 1-6), June 2018. [**HiPEAC Paper Award**]

[**5**] Heba Khdr, Santiago Pagani, Éricles Sousa, Vahid Lari, Anuj Pathania, Frank Hannig, Muhammad Shafique, Jürgen Teich, Jörg Henkel, "Power-Density-Aware Resource Management for Heterogeneous Tiled Multicores," *IEEE Transactions on Computers (TC)*, 66(3). 488 - 501, March 2017.

[**6**] Heba Khdr, Santiago Pagani, Muhammad Shafique, Jörg Henkel, "Thermal-Constrained Resource Management for Mixed ILP-TLP Workloads in Dark Silicon Chips," *ACM/EDAC/IEEE 52th Design Automation Conference (DAC)*, p. 179, June 2015. [**HiPEAC Paper Award**]

[**7**] Heba Khdr, Thomas Ebi, Muhammad Shafique, Hussam Amrouch, Jörg Henkel, "mDTM: Multi-Objective Dynamic Thermal Management for On-Chip Systems," *IEEE/ACM Design Automation and Test in Europe Conference (DATE)*, March 2014.

List of Co-Author Publications

[8] Anuj Pathania, Heba Khdr, Muhammad Shafique, Tulika Mitra, and Jörg Henkel, "QoS-Aware Stochastic Power Management for Many-Cores," *ACM/EDAC/IEEE 55th Design Automation Conference (DAC)*, pp. 1-6, June 2018. **[HiPEAC Paper Award]**

[9] Santiago Pagani, Heba Khdr, Jian-Jia Chen, Muhammad Shafique, Minming Li, Jörg Henkel, "Thermal Safe Power (TSP): Efficient Power Budgeting for Heterogeneous Manycore Systems in Dark Silicon," *IEEE Transactions on Computers (TC)*, 66(1). 147-62, January 2017.

[10] Anuj Pathania, Heba Khdr, Muhammad Shafique, Tulika Mitra, and Jörg Henkel, "Scalable Probabilistic Power Budgeting for Many-Cores", *IEEE/ACM Design, Automation and Test in Europe Conference (DATE)*, pp. 864 - 869, March 2017.

[11] Santiago Pagani, Heba Khdr, Jian-Jia Chen, Muhammad Shafique, Minming Li, Jörg Henkel, "Thermal safe power: Efficient Thermal-Aware Power Budgeting for Manycore Systems in Dark Silicon," *The Dark Side of Silicon*, Springer, pp. 125-158, January 2017.

[12] Santiago Pagani, Lars Bauer, Qingqing Chen, Elisabeth Glocker, Frank Hannig, Andreas Herkersdorf, Heba Khdr, Anuj Pathania, Ulf Schlichtmann, Doris Schmitt-Landsiedel, Mark Sagi, ricles Sousa, Philipp Wagner, Volker Wenzel, Thomas Wild, Jörg Henkel, "Dark silicon management: An Integrated and Coordinated Cross-Layer Approach," *Special Issue of (De Gruyter Oldenbourg) Information Technology on Invasive Computing*, 58(6). 297-307, December 2016.

[13] Jörg Henkel, Santiago Pagani, Heba Khdr, Florian Kriebel, Semeen Rehman, Muhammad Shafique "Towards Performance and Reliability-Efficient Computing in the Dark Silicon Era", *IEEE/ACM 19th Design, Automation and Test in Europe Conference (DATE)*, pp. 1-6, March 2016.

[14] Santiago Pagani, Muhammad Shafique, Heba Khdr, Jian-Jia Chen, Jörg Henkel "seBoost: Selective Boosting for Heterogeneous Manycores", *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pp. 104-113, October 2015.

[15] Jörg Henkel, Haseeb Bukhari, Siddharth Garg, Muhammad Usman Karim Khan, Heba Khdr, Florian Kriebel, Umit Ogras, Sri Parameswaran, Muhammad Shafique "Dark Silicon - From Computation to Communication", *9th International Symposium on Networks-on-Chip (NOCS)*, p. 23, September 2015.

[**16**] Jörg Henkel, Heba Khdr, Santiago Pagani, Muhammad Shafique "New Trends in Dark Silicon", *ACM/EDAC/IEEE 52nd Design Automation Conference (DAC)*, pp. 1-6 ,June 2015. [**HiPEAC Paper Award**]

[**17**] Waqaas Munawar, Heba Khdr, Santiago Pagani, Muhammad Shafique, Jian-Jia Chen, Jörg Henkel "Peak Power Management for Scheduling Real-time Tasks on Heterogeneous Many-Core Systems", 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS), pp. 200-209, December 2014.

[**18**] Santiago Pagani, Heba Khdr, Waqaas Munawar, Jian-Jia Chen, Muhammad Shafique, Minming Li, Jörg Henkel "TSP: Thermal Safe Power - Efficient Power Budgeting for Many-Core Systems in Dark Silicon", IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), p. 10, October 2014. [**Best Paper Award**]

[**19**] Jörg Henkel, Thomas Ebi, Hussam Amrouch, Heba Khdr "Thermal Management for Dependable on-chip System", 18th Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 113-118, January 2013.

Abstract

Driven by the ever-increasing performance demand, multicore processors have emerged enabling concurrent computations on a single chip. A multicore processor can be exploited by executing multiple applications simultaneously on the chip. Furthermore, multiple threads of each application can be also executed in parallel on multiple cores. To optimize for performance in multicores, it is necessary to employ resource management techniques that manage the resources of the chip considering the running applications. Specifically, resource management can allocate more cores to applications to exploit their parallelism and upscale the voltage and frequency levels (V/f levels) of the cores to increase their speeds.

While such potential decisions of resource management improve the performance, they may, unfortunately, elevate the on-chip temperatures, which have negative impacts on reliability. One of the major temperature-dependent reliability concerns is the circuit aging. In particular, aging leads to an increase in the threshold voltage, which, in turn, may increase the delay of the processor's paths and eventually induce timing errors. Due to the adverse impacts of temperature and aging on reliability, they have been considered as dire design constraints. Therefore, this dissertation focuses on the relevant problem of performance optimization under temperature and aging constraints. To achieve this goal, several resource management techniques are introduced in this dissertation to maximize the overall system performance, while satisfying temperature and aging constraints within the decision making process.

Firstly, temperature-constrained resource management techniques are proposed for homogenous and heterogeneous multicores. These techniques allocate cores to multithreaded applications, indicating the number of parallel threads of each application, and determine the V/f level of the cores that execute each application. These decisions are conducted while taking into account the Thread Level Parallelism (TLP) and the Instruction Level Parallelism (ILP) of the applications, aiming at maximizing the overall system performance under a temperature constraint. Moreover, the positioning of inactive (power-gated) cores throughout the chip is selected such that they facilitate dissipating the generated heat from active cores. That, in turn, reduces the temperature of active cores and allows upscaling their V/f levels leading to further performance improvement.

Afterwards, an aging-constrained resource management technique is introduced with the goal of maximizing the overall system performance under an aging constraint. Towards achieving this goal, an aging-aware design space is explored, which translates temperature and voltage into the resulting amount of aging quantified as an increase

in the threshold voltage. This aging-aware design space exploration enables the aging-constrained resource management technique to fully exploit the available space for performance optimization by selecting the numbers of parallel threads of the applications and the V/f levels of the cores. In order to optimize for performance at runtime using a light-weight approach, a boosting technique is presented which aims at maximizing the performance while at the same time reducing aging effects. Finally, a thermal-aware guardbanding technique is proposed with a new concept of dynamic guardband selection. This technique highlights the potentials of further performance improvements when the guardband type is dynamically selected at the system level with regard to the running workload.

The evaluation of the presented techniques is conducted using state-of-the-art simulation tools. The experimental results show significant performance improvements compared to several state-of-the-art techniques, while temperature and aging constraints are satisfied.

Big Picture

The Chair of Embedded System (CES) at the Karlsruhe Institute of Technology (KIT) has been a productive research environment for more than two decades. Its early research focused on hardware/software partitioning in system on chips (SoCs). Hardware/software (hw/sw) partitioning is a well-established design methodology with the goal to optimize for one design metric, e.g., system performance, while at the same time satisfying design constraints (e.g. total chip area). CES was the first to propose a HW/SW partitioning technique that minimizes the power consumption of the system [CES1]. In [CES2], a hw/sw portioning technique has been presented to minimize the energy consumption, and a case study on an MPEG-2 encoder has been demonstrated in [CES3]. To achieve the target optimization, it is necessary to accurately estimate several hardware and software parameters as proposed in [CES4, CES5]. In [CES6, CES7], performance optimization has been proposed for ASIP processors, by providing flexibility through runtime adaptation.

Starting from 2004, the focus in the CES has shifted to multicore systems, to keep pace with the trend of chip design, which predicted a continuous increase in the number of cores on a single ship. In multicores, network on chip (NoC) has been employed to enable communication between the cores. In [CES8], different NoC models with various configurations have been studied to analyze their impact on the performance. This analysis shows how optimizing NoC configurations can significantly increase the performance of multicores. An adaptive on-chip communication scheme has been proposed in [CES9] to satisfy a required level of quality of service (QoS). Besides optimizing for communication, the CES has proposed several schemes to optimize for the performance of computation in multicores. In [CES10] an agent-based application mapping scheme, named ADAM, has been presented to maximize the performance of the applications considering the communications between the cores. ADAM, however, simplifies the problem by splitting the research space into virtual clusters and considers only static applications that requires a constant number of cores. In contrast, DistRM has been proposed in [CES11] to map at runtime dynamic applications to the cores aiming at maximizing the overall system performance, while considering dynamic core requirements of applications. Another branch of techniques for multicores in the CES has focused on optimizing for power and temperature. [CES12][CES13] have proposed thermal management techniques to minimize the temperature in order to increase the reliability and dependability of multicores. Nevertheless, none of these aforementioned works has targeted the problem of maximizing performance under temperature constraints.

This dissertation presents resource management techniques that aim at maximizing the performance of multicores while at the same time satisfying temperature and aging constraints. The proposed techniques in this dissertation contribute to two projects funded by the German Research Foundation (DFG). These projects are (1) the DFG

Transregional Collaborative Research Center [SFB/TR 89] \ Invasive Computing and (2) the DFG SPP 1500 \ Dependable Embedded Systems. An overview of these projects and their aspects that this dissertation contribute to are explained below.

Invasive Computing

The Transregional Collaborative Research Center [SFB/TR 89] \ Invasive Computing project referred to as *InvasIC* is funded by the DFG in its second funding phase from July 2014 until July 2018. Its research association aggregates researchers from three universities in Germany (Friedrich-Alexander-Universität Erlangen-Nürnberg, Karlsruher Institut für Technologie, Technische Universität München).

The main idea of *InvasIC* is to investigate and develop a novel paradigm of invasive computing [20, 21] for designing and programming future parallel computing systems, in which a thousand or more cores are expected to be integrated on a single chip [22]. *InvasIC* is composed of different sub-projects to cover different aspects such as resource-aware programming, hardware requirements to enable invasive computing, software requirements like compilers and operating systems.

This dissertation contributes to the sub-project B3, which is responsible for developing methodologies to maximize the overall system performance and energy efficiency under power and temperature constraints on loosely-coupled MPSoCs. One of the new concepts developed in CES for B3 is the Thermal Safe Power (TSP) [18], which is an abstraction that provides thermally-safe power constraint as a function of the number of active cores. TSP presents the maximum allowed power on the active cores that keep the temperature below the predefined thermal constraint. TSP, however, is uniform for all cores on the chip and does not consider the diverse power characteristics of the different applications. Therefore, applying TSP does not guarantee exploiting all available safe thermal margins on the cores. Therefore, the technique in Chapter 5 is proposed to manage the resources so that thermal violations are avoided and all available thermal margins are exploited. The work in Chapter 6 presents a resource management technique to maximize the performance for tiled heterogeneous multicore architecture similar to the InvasIC architecture. This work is achieved through a collaboration with the sub-project B2, which provides the integration of TCPA [23] with the employed heterogeneous architecture. The approaches presented in chapters 7, 8, 9 also contribute to B3, since their goal is also maximizing the performance under temperature and aging constraints. Thus, this dissertation contributes to B3 through maximizing the performance under temperature constraints. To maximize performance under power constraints and to improve energy efficiency as well, other approaches outlined in the CES are presented (e.g., [8, 10, 24]).

Dependable Embedded System in SPP 1500

The goal of the SPP 1500 project is to develop new methods and architectures at system level to eliminate the effects associated when migrating to new technology nodes like malfunctioning, performance degradation, and increased power consumption. This project is composed of five research areas, these are: (1) Technology Abstraction, (2) Dependable Hardware Architectures, (3) Dependable Embedded System, (4) Design Methods, (5) Operation, Observation, Adaptation.

This dissertation contributes to the third research area of dependable embedded System [25]. Specifically, aging is a serious degradation for reliability and hence dependability. Aging issues are tackled by the techniques proposed in Chapters 7, 8, 9 and thus

these techniques contribute to the core of SPP. The work in Chapter 7 presents a resource management technique that maximizes performance under an aging constraint. Satisfying the predetermined aging constraint for a processor helps to improve the dependability of the processor during its projected lifetime. In Chapter 8, an aging-aware boosting technique is presented to maximize performance, while at the same time minimize aging effects. This, in turn, improves the dependability. The work in Chapter 9 selects the guardband types with the goal to maximize performance while ensuring reliable operation of the processor. These techniques and many others from CES (e.g., [26–28]) have covered the mains aspects of the SPP 1500 \ Dependable Embedded System.

[CES1] Jörg Henkel, "A low power hardware/software partitioning approach for core-based embedded systems," *Design Automation Conference (DAC)*, 1999.

[CES2] Yanbing Li, and Jörg Henkel, "A framework for estimation and minimizing energy dissipation of embedded HW/SW systems," *Design Automation Conference (DAC)*, 1998.

[CES3] Jörg Henkel and Yanbing Li, "Energy-conscious HW/SW-partitioning of embedded systems: A Case Study on an MPEG-2 Encoder," *International Workshop on Hardware/Software Codesign*, 1998.

[CES4] Jörg Henkel and Rolf Ernst, "A path-based technique for estimating hardware runtime in HW/SW-cosynthesis," *International Symposium on System Synthesis*, 1995.

[CES5] Jörg Henkel, Thomas Benner, Rolf Ernst, Wei Ye, Nikola Serafimov, Gernot Glawe, "COSYMA: a software-oriented approach to hardware/software codesign," *Journal of Computer and Software Engineering - Special issue: hardware-software codesign*, 1994.

[CES6] Lars Bauer, Muhammad Shafique, Simon Kramer, Jörg Henkel, "RISPP: rotating instruction set processing platform," *Design Automation Conference (DAC)*, 2007.

[CES7] Lars Bauer, Muhammad Shafique, Jörg Henkel, "Run-time instruction set selection in a transmutable embedded processor," *Design Automation Conference (DAC)*, 2008.

[CES8] Jiang Xu, Wayne Wolf, Jörg Henkel, Srimat Chakradhar, Tiehan Lv, "A case study in networks-on-chip design for embedded video," *Design Automation and Test in Europe Conference (DATE)*, 2004.

[CES9] Mohammad Abdullah Al Faruque, Thomas Ebi, Jörg Henkel, "Run-time adaptive on-chip communication scheme," *International Conference on Computer-Aided Design (ICCAD)*, 2007.

[CES10] Mohammad Abdullah Al Faruque, Rudolf Krist, Jörg Henkel, "ADAM: run-time agent-based distributed application mapping for on-chip communication," *Design Automation Conference (DAC)*, 2008.

[CES11] Sebastian Kobbe, Lars Bauer, Daniel Lohmann, Wolfgang Schrder-Preikschat, Jörg Henkel, "DistRM: Distributed Resource Management for On-Chip Many-Core Systems," *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2011.

[CES12] Thomas Ebi, Mohammad Abdullah Al Faruque, Jörg Henkel, "TAPE: thermal-aware agent-based power economy for multi/many-core architectures ," *International Conference on Computer-Aided Design (ICCAD)*, 2009.

[CES13] Thomas Ebi, David Kramer, Wolfgang Karl, Jörg Henkel, "- Economic learning for thermal-aware power budgeting in many-core architectures ," *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2011.

Contents

Chapter 1

Introduction

One of the guiding principles of processor design that enables fulfilling the ever increasing performance demand is known as Moore's Law [29], which states that the number of transistors on a chip will roughly double each two years. This law remains valid until the present day as technology feature sizes have steadily scaled down over the past decades. Technology scaling has been guided by Dennard scaling [30], which states that as transistors become smaller, their power density stays constant. Hence, both supply voltage and current must be downscaled, so that power consumption is reduced by the same scaling factor as area. The combination of Moore's law and Dennard scaling has provided over thirty years many generations of higher performance processors, in which increasing clock frequency has been the most important contributor to performance gains.

Since around 2002 it has become difficult to further follow Dennard scaling. In particular, supply voltage cannot be downscaled with the same factor as feature size, because it has run into lower limits imposed by threshold voltage scaling limits [31]. Hence, increasing clock frequency along with technology scaling must be limited to avoid high power densities. As an alternative way for improving performance, chip designer started integrating multiple cores on a single chip. Thus, the shift to multicore chips was a natural evolutionary step to keep fulfilling the ever increasing performance demand. According to ITRS [32] predictions shown in Figure 1.1, hundreds of cores will exist in mobile devices over the next decade, while in servers thousands of cores are predicted to exist. Today, an example of a commercial multicore chip is Intel Xeon Phi [33] with 64 cores.

Unfortunately, the problem does not end here; although Dennard scaling has discontinued, technology node sizes are continuously shrinking, and power densities are increasing. Figure 1.2 shows the estimated increase in the power density of the chip along with technology scaling, which has been estimated based on technological data from ITRS [32] and Intel [34]. High power densities lead to increasing temperature on the chip. To avoid thermal emergencies, not all cores on the chip can run simultaneously at full speed, but a fraction of cores needs to be power-gated (*dark*). This problem is known as *dark silicon*, and it is expected to be dominant in the upcoming technology nodes. Such a problem prevents obtaining the expected performance gain of multicore chips and technology scaling.

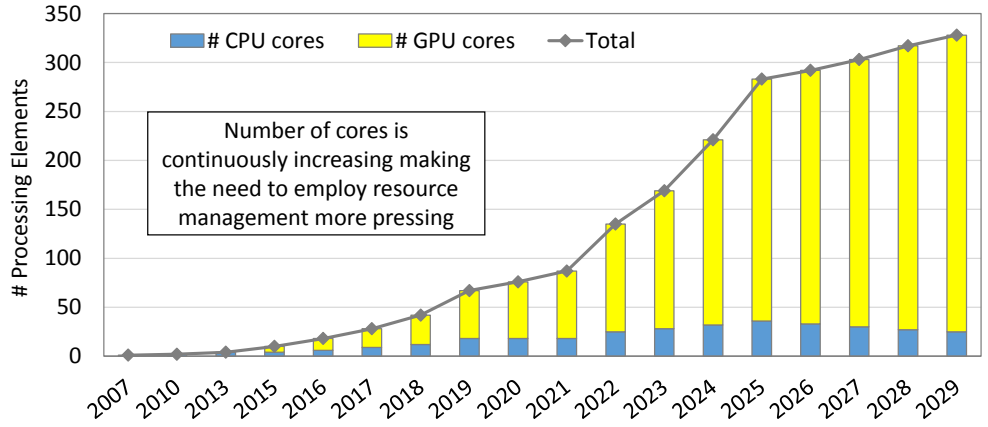
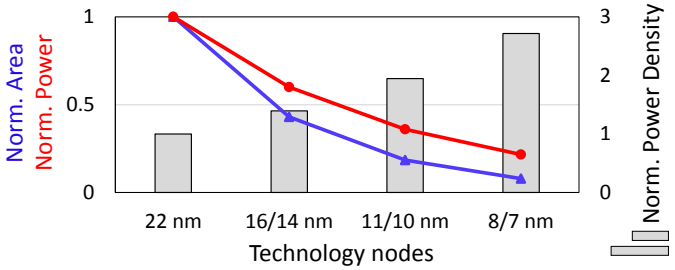


Figure 1.1: ITRS predictions for the number of cores in mobile devices.

Figure 1.2: The estimated increase in the power density along with technology scaling based on technological data from ITRS [32] and Intel [34]. Area, power, and power density values are normalized to the corresponding values at 22 nm.



High temperature not only leads to *dark silicon* problem, but also has severe negative impacts on reliability. Among others, it is a main contributor to accelerated aging process. Circuits have always aged, but this aging has only become significant when high-K oxides and metal gate have been applied in transistors at recent technology nodes [35]. Circuit aging may lead to an increase in the delay of processors' critical path over time which may eventually result in timing violations and errors.

To limit the adverse impacts stemming from elevated temperatures and accelerated aging, it is necessary to enforce temperature and aging constraints, under which it is deemed to be a safe range. Enforcing such constraints, however, will limit performance improvement. That makes the problem of performance optimization under temperature and aging constraints of paramount importance, and therefore this dissertation has the aim of solving this problem.

1.1 Performance Optimization

The emergence of multicores required a shift of programming paradigm to parallelization. First, multiple applications can be executed simultaneously on different cores on the chip. Furthermore, multi-threaded applications are introduced providing fine-grained parallelization to further exploit the available concurrent execution capabilities on multicores by running application threads in parallel on different cores.

As computing systems are continuously becoming more complex with increasing number of cores, number of simultaneous applications, and number of parallel threads per

application, it becomes indispensable to employ resource management techniques that allocate cores to applications with the goal of maximizing the overall system performance. To achieve this goal, resource management techniques must first consider the diverse TLP of the applications. In particular, each application has a sequential part and a parallelizable part. The dominance of the parallelizable part of the application indicates its TLP. The speedup in execution time of an application when running parallel threads is limited by its sequential part. The more dominant the sequential part, the less speedup is. Thus, allocating more cores and thereby more parallel threads to applications with higher TLP will improve the overall system performance.

An additional means can be employed by resource management techniques to improve the overall system performance is Dynamic Voltage and Frequency Scaling (DVFS) [36]. DVFS is a feature implemented in most of recent processors [37] that allows setting different V/f levels to the cores in order to tune their speed and power. Intuitively, when high V/f level is selected, the application performance will be increased, as more instructions per second will be executed. The increase in application performance when V/f level is increased is related to its ILP, as ILP indicates how many instructions of the application can be executed simultaneously. Analogous to the previous discussion, the overall system performance can be improved, when setting high V/f levels to the cores that execute applications with high ILP. An example of two applications that expose different TLP and ILP is shown in Figure 1.3. Evidently, for the first application more performance gain is obtained when allocating more cores to it rather than increasing the V/f level of its cores. Contrarily, for the second application, more performance can be obtained when the V/f levels of cores are increased compared to the obtained performance when allocating more cores to the application.

Thus, an efficient resource management must consider the TLP and ILP of the applications to maximize the overall system performance.

Unfortunately, the utilized means by resource management to maximize the performance can lead to increasing both temperature and aging. Specifically, as seen in Figure 1.4, increasing the number of parallel threads of the applications and upscaling the V/f levels of the cores lead to increasing the number of active cores and the switching activity in the circuits. That, in turn, elevates the temperature on the chip. Elevated temperature and increased voltage accelerate aging mechanisms. This impact of resource management means on temperature and aging emphasizes more the importance of enforcing temperature and aging constraints at the system level within the decision making process of resource management. In the following sections, the current practices to cope with temperature and aging are investigated and the key challenges are highlighted.

1.2 Temperature

The front line of defense against elevated temperatures on the chip is the cooling system. However, the cost of the cooling system is proportionally increasing per watt of the heat dissipated [38]. Therefore, it is typically designed to dissipate a specific power amount, referred to as Thermal Design Power (TDP). By definition, TDP is not the maximum power consumption on the chip; it might be exceeded to satisfy performance surges, and in this case the chip might overheat.

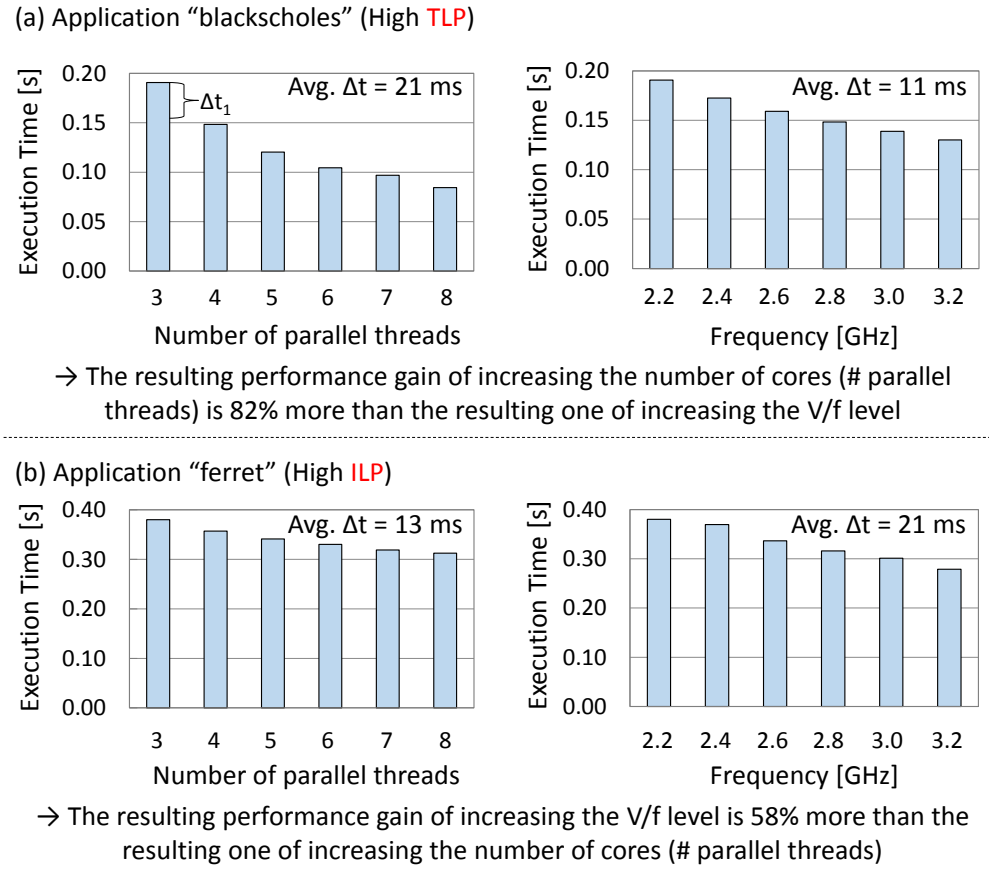


Figure 1.3: The impact of increasing number of cores or V/f level on the performance of two applications with different TLP and ILP.

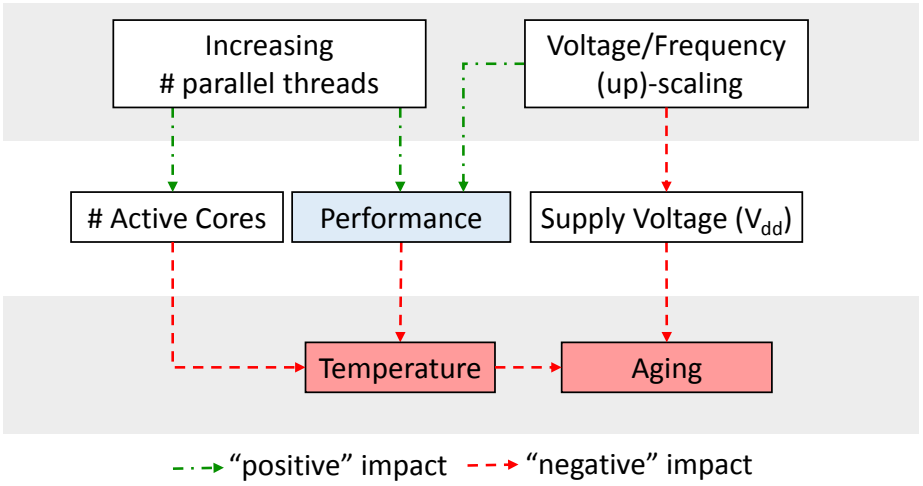


Figure 1.4: The impact of resource management means on temperature and aging.

The current practice to protect the chip from potential overheating is to employ Dynamic Thermal Management (DTM). Typically, DTM is a reactive technique implemented on the hardware to quickly react to thermal violations. Particularly, if some part of the chip heats up above a predefined critical temperature (T_{crit}), DTM is triggered in order to reduce the temperature by increasing fan speed, downscaling V/f levels of the cores, and power-gating some cores.

By downscaling V/f levels and/or power-gating cores, DTM contradicts resource management decisions and ultimately hinders it from achieving its goal of performance maximization.

With the purpose of avoiding thermal violations and thereby DTM consequences, state-of-the-art techniques employ TDP as a chip-level power constraint on the chip. Hence, the total power consumption of the cores is enforced to be below TDP. This might potentially reduce the temperature on the chip but it is not sufficient to avoid thermal violations, as will be discussed in the following case study.

1.2.1 Case Study

This case study shows the resulting thermal maps of a homogeneous 64-core chip[1], considering various scenarios. The employed chip has a critical temperature (T_{crit}) of 80 °C and a TDP of 220 W, which is enforced as a chip-level power constraint. This case study consists of two parts explained below.

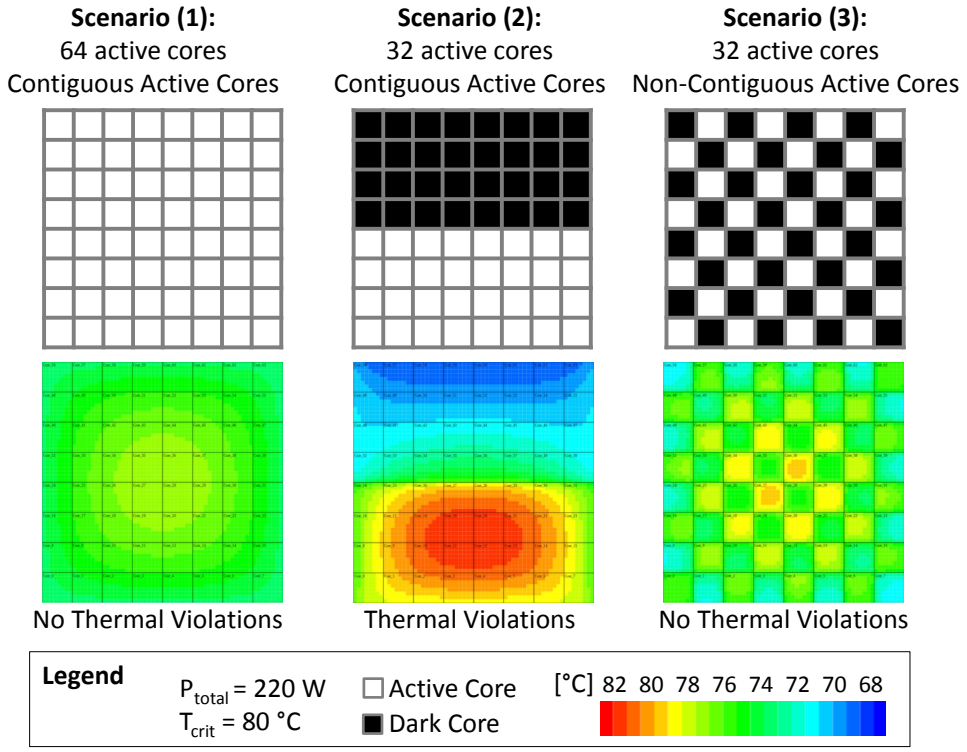


Figure 1.5: Different thermal maps using the same total power consumption and different positioning of active and dark cores.

The first part (illustrated in Figure 1.5) shows three scenarios differing from each other by the number of active cores and the positioning of active and dark cores throughout the chip. In these three scenarios, the TDP is distributed evenly among the active cores. In scenario (1), all cores on the chip are active, and hence each core consumes 3.4 W. As it can be noticed from Figure 1.5-(1), no thermal violation occurs. In scenario (2),

[1] Alpha cores in 22 nm, simulated with gem5 [39] and McPAT [40]. Further details are presented in Chapter 4.

only 32 cores are active, and since the same TDP is used, each active core consumes 6.8 W. As Figure 1.5-(2) shows, thermal violations occur. Hence, consuming TDP by a smaller number of cores on the chip results in a higher power density and thereby thermal violations. Scenario (3) has the same number of active cores and per-core power consumption as scenario (2), but it has different distribution of active and dark cores on the chip. Particularly, the active cores are contiguous in scenario (2), while they are non-contiguous in scenario (3). This non-contiguous positioning of active cores in scenario (3) enables higher heat dissipation. Specifically, being dark cores in close vicinity to active cores facilitates dissipating more heat from the active cores, and thereby, their temperatures are decreased.

These scenarios reveal that considering TDP as a power constraint is not enough to avoid thermal violations. Moreover, different positioning of active and dark cores can lead to diverse thermal maps, even if the active cores have the same power consumption.

The second part (illustrated in Figure 1.6) shows two scenarios differing from the previous ones (Figure 1.5) by considering realistic power consumptions resulting from executing applications on the cores. Specifically, two instances of two applications from the PARSEC benchmark [41] (i.e., "x264" and "canneal") are executed simultaneously on the chip. Executing "x264" leads to higher power consumption than executing "canneal". The resulting total power consumption on the chip is 217 W. These two scenarios share the same number of active cores and the same positioning of active and dark cores, but they differ in the application-to-core mapping.

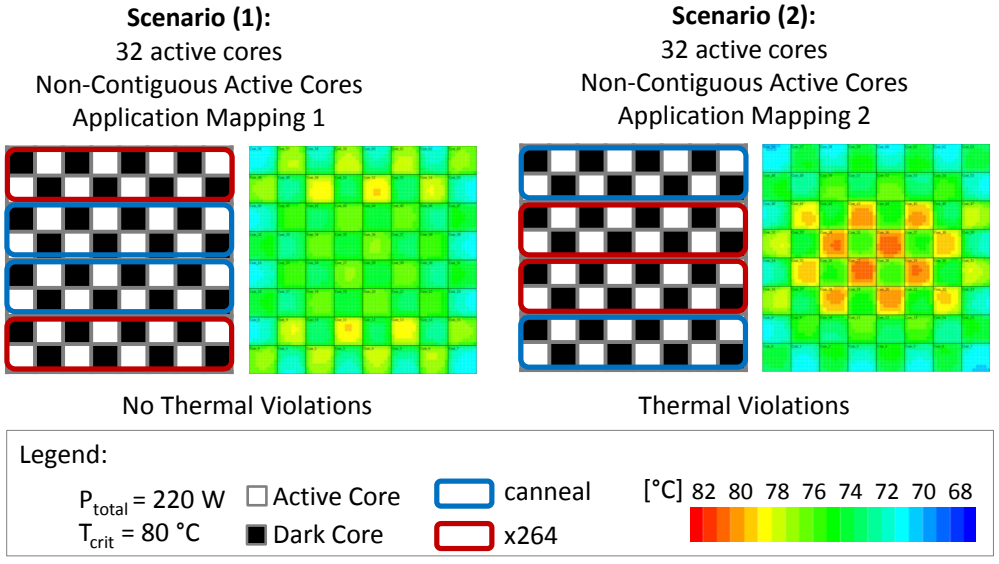


Figure 1.6: Scenarios with different application-to-core mapping result in different thermal maps, although the total power consumption and the positioning of active and dark cores are the same in these scenarios.

As shown in Figure 1.6, although both scenarios have the same total power consumption, i.e., 217 W, which is below TDP and the dark cores are in close vicinity to the active cores, thermal violations occur only in scenario (2). This is because in scenario (2) the instances of high-power application ("x264") are mapped close to each other and allocated to the interior cores, which have high susceptibility to temperature increase. In contrast, in scenario (1), the instances of "x264" are mapped to the outer cores near the

chip edges, where more heat can be dissipated.

According to this case-study, the power properties of applications and application-to-core mapping have a significant impact on the generated temperature, and this impact cannot be considered using power constraints.

It is noteworthy that many thermal management techniques have attempted to maximize performance while directly considering a temperature constraint. In particular, they allow the system to operate at the maximum allowable power that keeps core temperatures as close to a specified temperature constraint as possible. This, however, cannot be translated into maximizing the overall system performance, since these techniques do not determine the number of cores that should be allocated to each multi-threaded application, and also they do not consider application properties, such as TLP and ILP to optimize their performance. Having these parameters to be determined or considered calls for resource management solutions.

1.3 Aging Effects

As previously discussed, technology scaling and elevated temperature accelerate aging mechanisms on transistors [42]. Among different aging mechanisms, Bias Temperature Instability (BTI) has become one of the major reliability concerns due to its considerable ability to degrade the electrical characteristics of pMOS and nMOS transistors [43]. In particular, applied BTI leads to charge buildup within the gate dielectric and hence it weakens the electric field. That increases the threshold voltage (V_{th}) of transistors [43] over time. In effect, transistors with higher V_{th} switch more slowly due to the smaller drain current (I_d) as it is seen from the first order approximation of I_d in Equation (1.1)[2].

$$I_d \approx (V_{dd} - V_{th} - \underbrace{\Delta V_{th}}_{\text{due to aging}}) \tag{1.1}$$

$$\text{Transistor delay} \propto \frac{1}{I_d} \Rightarrow \text{ aged transistors are slower}$$

This slowdown in transistor speed leads to prolonging the critical path delay of the processor ($t_{critpath}$) during its lifetime. However, the processor's clock delay (t_{clk}) is specified at design time according to the nominal specifications of the processor (i.e., without any increase in V_{th}). Hence, $t_{critpath}$ might become longer than t_{clk} during lifetime resulting in timing violations and errors, as Figure 1.7 illustrates. This well-known phenomenon of the gradual increase in V_{th} and hence in the delay of the processor's paths is referred to as *long-term aging effects*. The common practice to avoid aging-induced timing errors is to add a time slack (guardband), referred to as t_{GB}, to the clock delay, as Figure 1.7-(c) shows.

Determining the width of t_{GB} necessitates specifying an upper bound (constraint) for the increase in the threshold voltage during the targeted lifetime, denoted as ΔV_{th}^m. To keep the *amount of aging* (i.e., V_{th}) below a predetermined aging upper bound ΔV_{th}^m,

[2]It is to be noted that only for the sake of simplicity, Equation (1.1) uses the simplified relation between the transistor delay and the drain current of transistor which does not consider the MOSFET short channel effects. However, the aging modeling in this dissertation employs a detailed modeling of I_d using the state-of-the-art industrial standard compact modeling for MOSFET (Berkeley Short-channel IGFET Model (BSIM)) [44, 45].

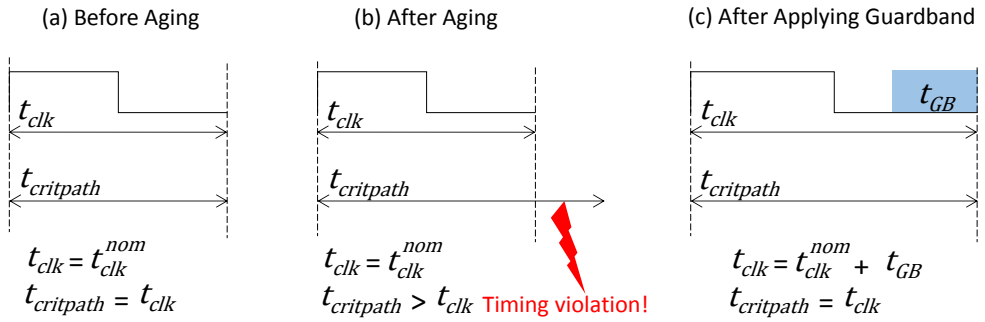


Figure 1.7: A preliminary example illustrating the impact of aging on the critical path delay of the processor during its lifetime. To avoid timing violations induced by aging, a timing guardband t_{GB} can be added to the processor's clock delay.

state-of-the-art techniques impose conservative temperature and V_{dd} bounds.

The drawback of this practice is that the resource management's goal of maximizing performance is then limited by such conservative bounds.

Although the aging constraint can be enforced, the problem, however, does not end here as the work proposed in [46] has (through measurements) discovered that a "sudden" increase in the critical path delay of the processor temporally arises during applying DVFS, and particularly when the voltage of the processor (V_{dd}) is downscaled. This phenomenon has been also studied in [28], where it is demonstrated that the conjunction between voltage scaling and the increase in the threshold voltage (ΔV_{th}) results in increases in the critical path delay that are observable in short time scale (i.e., in order of microseconds). These effects are referred to as *short-term aging*. To compensate for this additional temporal increase in the critical path delay, a guardband wider than t_{GB} is required. Intuitively, a wider guardband leads to higher performance losses.

As a result, these recently discovered aging effects will challenge the efficacy of DVFS, which is an essential means to optimize for performance.

1.3.1 Guardband Types

As illustrated in Figure 1.7, timing violations can be avoided by adding a guardband to the clock delay. That implies that the operating frequency of the processor needs to be reduced as shown in the following equation:

$$f^{GB} = \frac{1}{t_{clk}^{nom} + t_{GB}} < f^{nom} = \frac{1}{t_{clk}^{nom}} \tag{1.2}$$

There, f^{nom} refers to the nominal frequency without any reduction. This type of guardbanding is referred to as *frequency guardband* (F_GB). The disadvantage of F_GB is that it prohibits the system from utilizing the possible full speed of the processor, and hence limiting the performance.

As an alternative to frequency guardband, some state-of-the-art guardbanding techniques suggest applying a voltage guardband. Particularly, to compensate for the reduction in the I_d shown in Equation (1.1), a voltage guardband can be added to the supply voltage V_{dd}, as Equation (1.3) shows, ensuring that any increase in V_{th} due to aging will

always be compensated for. Thus, the processor can still be clocked with the nominal frequency without exhibiting timing errors, even though aging effects may occur during its lifetime.

$$V_{dd}^{GB} = V_{dd}^{nom} + \Delta V_{th} > V_{dd}^{nom} \tag{1.3}$$

This type of guardbanding is referred to as *voltage guardband* (V_GB). The disadvantage of V_GB is that the power consumption will be increased, since V_{dd} has been increased.

State-of-the-art techniques determine at design time of the processor (circuit level) to employ either frequency guardband (F_GB) (e.g., [47]) or voltage guardband (V_GB) (e.g., [48]) with respect to the system requirements. For instance, when the performance of the targeted system cannot be scarified, V_GB is employed as it enables using the nominal frequency without any reduction. For low-power systems, F_GB is employed since it does not lead to a power increase. However, this traditional view of the impact of the guardband type is only valid at the circuit level. Traversing towards the system level, V_GB is not necessarily able to sustain the performance (as revealed by our case study that will be presented in Chapter 9). This is because V_GB increases the power consumption which may lead to thermal violations. Consequently, DTM will be triggered leading to performance losses. Hence, the performance would decrease unlike what is expected.

Thus, selecting a fixed guardband type at the circuit level might not be efficient at the system level.

1.4 Key Challenges

This dissertation proposes to solve the problem of performance optimization under temperature and aging constraints through resource management decisions at the system level. Such resource management needs to address the following challenges:

The current practice to cope with thermal issues within resource management is to employ the TDP as a chip-level power constraint. However, adopting a chip-level power constraint is not enough to avoid thermal violations. Thermal violations, in turn, trigger DTM on the chip, which downscales V/f levels and/or power-gates cores with the purpose of reducing the temperature. That, however, contradicts resource management decisions and ultimately hinders it from achieving its goal of performance maximization. Additionally, there are several factors that affect the core temperatures and must be considered by resource management to solve the problem of performance optimization under temperature constraint. These factors are the positioning of active and dark cores within the chip, the diverse power properties of the applications and the application mapping to the cores.

Significantly, the problem will be even more complicated for heterogeneous multicore chips, where the power and performance characteristics of applications significantly differ from a core type to another. Moreover, thermal characteristics of heterogeneous multicores are also different. Even if two heterogeneous cores have the same power value, the resulting temperatures, the two exhibit, might be different if they have different areas. Thus, these different characteristics need to be taken into account when optimizing the performance of heterogeneous multicore chips under temperature constraint.

The current approach to prevent aging-induced timing errors is to add a timing guardband to the clock delay. Determining the necessary guardband width necessitates specifying an upper bound for ΔV_{th}. This upper bound or so-called aging constraint is typically satisfied by enforcing conservative temperature and V_{dd} bounds. That, however, limits the system performance. Nevertheless, the determined guardband width might still not be enough to avoid timing errors, due to short-term aging effects that might be stimulated by DVFS. Consequently, a wider guardband is required to compensate for the delays induced by long and short-term aging effects, or using DVFS should be limited. Besides the guardband width, the guardband type has a significant impact on the system performance. Typically, the guardband type is selected at the circuit level. However, its impact on the system performance cannot be accurately estimated irrespective of the running workload at the system level.

1.5 Dissertation Contributions

To cope with the aforementioned challenges, this dissertation makes the following contributions with the goal to maximize the performance under temperature and aging constraints. An overview of the dissertation contributions is shown in Figure 1.8.

Dark Silicon-Aware Resource Management
First a novel dark silicon-aware resource management, named *DsRM*, is presented. *DsRM* aims at maximizing the overall system performance in multicore chips under temperature constraint. To achieve its goal, *DsRM* determines the number of cores that should be allocated to each application and the V/f level of these cores, while taking into account the the ILP and TLP of the applications. Furthermore, *DsRM* maps the applications to the chip considering their power consumptions. Particularly, high-power applications are mapped to the cores that have high susceptibility to dissipate heat (e.g., the cores near the chip edges). Moreover, the positioning of dark cores is selected to be in close vicinity to the active cores that execute high-power applications, to facilitate dissipating the generated heat on the active cores, and hence reducing the temperatures of the active cores. That might allow to increase the V/f levels leading to further performance improvement.

Power Density-Aware Resource Management
For heterogeneous tiled multicores, where the problem of performance optimization under temperature constraint becomes more complicated, we introduce power density as a novel system-level constraint to abstract from thermal issues. Nevertheless, the adopted power density constraint is derived based on the thermal model of the chip such that it guarantees avoiding thermal violations. Then, a resource management *PdRM* technique is developed to assign applications to tiles by choosing the number of parallel threads of each application and the V/f level of each tile, such that the power density constraint is satisfied. Moreover, *PdRM* adapts the power density constraint at runtime to consider the resulting power consumptions of the running applications on the heterogeneous cores, and to react to the workload changes. This adaptation enables exploiting any available thermal headroom on the cores, and that, in turn, helps maximizing performance.

Aging-Constrained Resource Management
To enable maximizing performance under an aging constraint, a new aging-aware design space is explored which translates temperature and V_{dd} into the *amount of aging*, quantified by ΔV_{th}, using an accurate physics-based aging model. Then, an aging-constrained

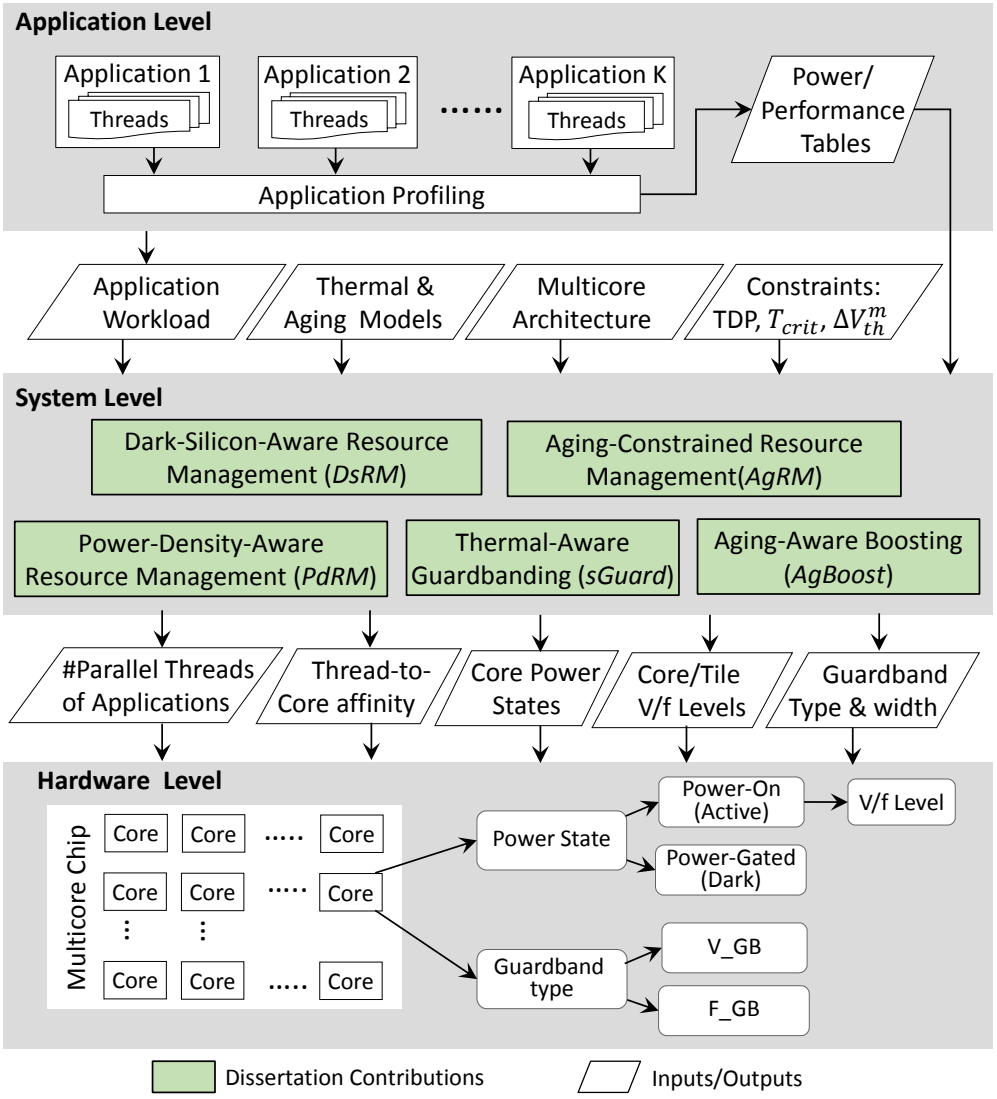


Figure 1.8: An overview of the contributions presented in this dissertation.

resource management technique *AgRM* is tailored to fully exploit the performance/aging trade-off within this design space without the restrictions inherent to current state-of-the-art. More specifically, *AgRM* adjusts the system level parameters, i.e. the number of parallel threads of the applications and the V/f levels of the cores, such that the performance is maximized under an aging constraint.

Aging-Aware Boosting

To provide a deeper insight about the recently-discovered short-term aging effects, a timing analysis is conducted at the circuit level to demonstrate how the conjunction between ΔV_{th} and DVFS influences the processor's critical path delay. This analysis enables estimating the induced delays by both long-term and short-term aging effects as well as the required guardband to compensate for these delays. Furthermore, the impacts of a traditional DVFS-based boosting mechanism on long-term and short-term aging effects are analyzed. Such analyses help to derive a boosting mechanism that limits the stimuli of these effects, and hence reducing aging-induced delays. As a result, the required guardbands to compensate for these delays are reduced. On that basis, a comprehensive, yet efficient, boosting technique is proposed in this dissertation that is

able to maximize the performance, while at the same time minimizing the long-term and short-term aging effects.

Thermal-Aware Guardbanding
The impact of the different guardband types (i.e., V_GB and F_GB) on the system performance is investigated. This investigation reveals that the generated temperatures by running the workload have the potential to play a key role in determining the appropriate guardband type with respect to the system performance. Therefore, a thermal-aware guardbanding technique, named *sGuard* is proposed to select the guardband types at runtime according to the the workload-induced temperatures aiming at optimizing for performance under temperature and reliability constraints. Moreover, different guardband types for different cores can be selected simultaneously when multiple applications with diverse properties suggest this to be useful. Our dynamic guardband selection, which can be considered as a paradigm shift in designing guardbands, allows for a higher performance compared to techniques that employ a fixed (at design time) guardband type throughout.

1.6 Dissertation Outlines

The remainder of this dissertation is organized as follows:

- Chapter 2 discusses the related work that either concerns the problem of performance optimization under temperature and aging constraints or the dual problem.
- Chapter 3 presents the adopted system model in this dissertation, which consists of power, thermal, aging, and application models.
- Chapter 4 explains the utilized simulation tools throughout the dissertation.
- Chapter 5 presents the details of the dark silicon-aware resource management technique.
- Chapter 6 presents the power density-aware resource management technique for heterogeneous multicores.
- Chapter 7 introduces a new aging-aware design space exploration and a resource management technique that exploits this design space to maximize the performance under aging constraint.
- Chapter 8 presents the proposed aging-aware boosting technique.
- Chapter 9 presents the thermal-aware guardbanding technique.
- Chapter 10 concludes the dissertation and discusses future work.

Chapter 2

Related Work

This chapter summarizes the related work to the problem of performance optimization under temperature and aging constraints. We categorize the related work into three main categories which are resource management techniques, thermal management techniques and aging management techniques. Under each of them, sub-categories might be defined.

2.1 Resource Management Techniques

The need to employ resource management techniques is becoming more and more pressing due to the continuous increase in the number of cores on chip. In response to this need, many resource management techniques have emerged using different policies and considering different constraints. The main task of resource management is to distribute the cores among multi-threaded applications that run simultaneously on the chip. The objective is to maximize the overall system performance. The work proposed in [49] distributes the cores among the applications based on the speedup curve of each application. More cores will be assigned to the applications which have higher speedup gain. The speedup model is based on an enhanced version of Downey's speedup models [50] for parallel applications. Similar work is presented in [51], but additionally the heterogeneity of the cores is considered. In [52], an adaptive clustering policy is proposed to distribute the resources among the applications. Each application is assigned to one virtual cluster, so that the cluster adapts and changes its size at runtime to increase the performance of its application. The proposed policy in [53] maps the applications to the chip, so that the fragmentation of the chip is reduced as much as possible. Reducing the fragmentation allows better mapping for new arriving applications, which, in turn, improves the performance of the system. The work of [54] has proposed an incremental mapping algorithm that maps the applications to the cores so that the required communication between the application tasks over NoC is reduced. None of these techniques [49, 51–54] considers power or temperature constraints.

As a response to the challenges stemming from elevated temperatures on the chip, like dark silicon, researchers have started to enforce power constraints by resource management as an abstraction from thermal issues. The reason for avoiding to directly deal with temperature constraints is that the complexity of resource management that allocates cores to multi-threaded applications is known to be NP-hard problem [49]. Considering temperature will further increase the complexity of the problem. Therefore, to avoid

further complexity, chip-level power constraints are considered instead. To enforce a chip-level power constraint, resource management techniques consider it as a power budget for the chip and distribute it among the applications similar to the cores. For example, the presented work in [55] distributes a power budget among the applications, and then each application distributes its power quota among its threads according to their criticality. Based on their determined power quota, the V/f levels of the cores are selected. An auction-based multi-agent system is introduced in [56] which distributes a chip-level power budget among the chip clusters and determines the number of active cores of each cluster based on their assigned power budgets. In [57], a hierarchical resource management technique for tiled heterogeneous architecture is presented aiming at maximizing the overall system performance without exceeding the TDP. This is achieved by employing PID controllers that allocate cores to applications and adjust the V/f levels of the tiles, considering the given goal and constraint. A resource management technique presented in [58] employs linear programming to determine the number of cores allocated to each application considering the power and performance characteristics of the applications. A new concept for power budgeting, named TSP, is presented in [18] which provides a thermally-safe per-core power constraint based on the thermal model of the chip and the number of active cores. However, as shown in Figure 1.6, applications can lead to diverse power consumptions on the cores, and thus considering the same power constraint (e.g.,TSP) for all cores hinders potential performance gains.

As a result, none of the resource management techniques that distribute the cores among the applications directly consider temperature constraints. In contrast, this dissertation presents resource management techniques that distribute the cores among the applications aiming at maximizing the overall system performance while at the same time satisfying temperature constraints.

2.2 Thermal Management Techniques

A large body of system-level techniques has emerged to control the temperature on the chip. A class of these techniques targets the problem of performance optimization under temperature constraints. Another class aims at optimizing the thermal profile of the chip in order to mitigate the severe impacts of temperature on reliability. These two classes are explained in Section 2.2.1 and Section 2.2.2, respectively.

2.2.1 Performance Optimization under Temperature Constraints

Several system-level approaches aimed at optimizing performance under temperature constraints. Some of them attempted to achieve this goal by mitigating dark silicon problem and thereby enabling the activation of more cores on the chip. In particular, the technique proposed in [59] allows exceeding the TDP constraint for a very short period of time, in order to activate some extra cores and thereby increase the performance of the multi-threaded applications. To dissipate the additional heat generated during this period, a phase-change material is utilized in the thermal packaging. In [60], an application mapping technique exploits dark silicon to balance the temperature of active cores. As a result, a higher power budget can be used to increase performance without leading to thermal violations. In particular, the applications are greedily mapped to the

coolest regions on the chip and then the mapping inside the regions are determined to distribute the heat as much as possible. The technique in [61] determines the mapping of application threads to the cores and hence the locations of dark cores, so that the core temperatures are reduced. As a result, more cores can be activated to run more threads and thereby the performance is increased while the core temperatures are kept below a predetermined temperature constraint. This thread-to-core mapping also account for the impact of process variations on the speed and the power of the cores. Additionally, to reduce the overhead of making such decisions at runtime, a lightweight temperature prediction mechanism is utilized to estimate the resulting temperature distribution for candidate solutions. The drawback of these techniques (i.e., [59–61]) is that they do not consider DVFS within their decisions, although DVFS has been widely adopted by multicore processors, because it provides fine-grain control with the power and the speed of the cores.

Therefore, several techniques can be found on the literature that employ DVFS to achieve the goal of performance optimization under temperature constraint. An early approach is presented in [62], which introduced a closed-loop thermal controller that iteratively increase the V/f levels of the cores to maximize the performance without leading to thermal violations. A similar approach is presented in [63], but distributed thermal controllers are employed on the cores in order to provide scalability with the increasing number of cores. A thermal management technique is proposed in [64] to assign tasks to cores, besides adjusting the V/f levels of the cores, aiming at maximizing performance under a predetermined temperature constraint. Similarly, the technique proposed in [65] assigns tasks to cores and adjusting the V/f levels of the cores, but additionally heterogeneous multicores are considered. The main drawback of these two techniques, i.e., [64, 65], is that they ignore the lateral heat flow between the cores in order to reduce the time complexity of their algorithms at runtime. Ignoring the lateral heat flow might be acceptable in the adopted architectures by those techniques. Specifically, in [64] the floorplan of the adopted multicores has four cores separated by four large L2 caches. That in turn reduces the lateral heat flow between the cores. The proposed technique in [65] is implemented on Exynos 5 Octa (5410) chip [66]. This platform has two tiles with two cores each. These tiles are separated by a substantial distance, and thereby the heat flow between the cores of these tiles is found to be negligible. However, in general ignoring the heat flow between cores is unrealistic assumption in the recent multicores, where more cores are integrated on the chip near each other, e.g. Intel Xeon Phi [33]. Moreover, our case study presented in Section 1.2.1 demonstrates the influence of the heat flow between the cores on the temperatures. A selective boosting technique (*seBoost*) is presented in [14] based on an Resistance-Capacitance (RC) thermal network that does consider the lateral heat flow between the cores. *seBoost* upscales the V/f levels of the cores that execute high-priority applications to satisfy their performance demands. Meanwhile, it downscales the V/f levels of the cores that execute low-priority applications, so that no thermal violations occur. To be able to select these V/f levels that satisfy the performance demand and avoid thermal violations, *seBoost* requires prior knowledge about the applications, i.e., their performance and power consumptions at different V/f levels.

The state of the art for maximizing performance under temperature constraints without depending on prior knowledge of the applications is the DVFS-based turbo boosting technique, which is widely adopted by several processor manufacturers like Intel Turbo Boost [67, 68] and AMD Turbo Core [69]. The Intel Turbo Boost technique has been first implemented in Nehalem architecture at 2008 [67], but it is still adopted in the recent Intel processors, like Intel Core i7 [68]. In order to maximize the performance,

boosting techniques allow processor cores to run faster than the base operating frequency and hence they can exceed TDP. To keep the processor operating within safe margins, a simple closed-loop control system is employed to upscale the V/f level of the cores when the power and temperature are below specific limits, and downscale it otherwise. These techniques have shown superiority in improving the performance [70], and therefore, processor manufacturers keep improving and implementing these techniques with the new generations of the processors [68].

All of these techniques presented under this category attempted to optimize performance under temperature constraints by allowing the system to run at the maximum power level that keeps core temperatures as close to a specified temperature constraint as possible. This is, however, cannot be translated into maximizing the overall system performance (the target of this dissertation), since the problem of resource allocation (distributing the cores among application) has not been addressed by these techniques. Moreover, the application characteristics; TLP and ILP, which have a significant impact on the system performance, have not been considered.

2.2.2 Temperature Optimization

Optimizing for temperature means minimizing temperature and/or balancing it among the cores on the chip. That allows mitigating reliability concerns induced by high temperatures and high variations.

Several techniques optimize for temperature by mitigating hot spots on the chip and balancing the temperature between the cores. This problem has been solved optimally at design time using integer linear programming for a known set of tasks as presented in [71]. Contrarily, the work in [72] presents a runtime task allocation policy that balances the workload among the cores based on the predicted temperatures. If a thermal imbalance is predicted between the cores, the tasks will be migrated from the hottest core to the coldest one. The number of task re-allocations is bounded to reduce the performance overhead of such a technique. The temperature is predicted using auto-regressive moving average model. Another technique in [73] proposes a distributed thermal management that aims at balancing the temperature and the workload throughout the chip. For this purpose, it migrates tasks between only neighboring cores, so that the overhead of task migration is reduced. Similarly, the technique proposed in [74] migrates tasks among the cores to balance the temperature throughout the chip. To achieve their goal, these thermal management techniques rely solely on task migration which comes with a significant performance cost [75], while using DVFS leads to a cubic power reduction with an expense of linear performance degradation [64].

Therefore, several techniques (e.g., [76–80]) employ DVFS to optimize temperature. The work in [78] presents a distributed thermal management technique, referred to as *TAPE*. The *TAPE* approach focuses on balancing the power consumptions and thereby the resulting temperatures over a multicore architecture while at the same time meeting the performance constraints of the running tasks. This is achieved by incorporating an agent system, in which each agent is responsible for adjusting the V/f level of one core. In particular, based on the temperature of its own core, an agent can trade the so-called power units with its neighboring agents, based on the current temperatures of their cores. When an agent receives additional power units, it upscales the V/f level of its core, and downscales it otherwise. During runtime, if an agent does not have enough

power units to meet the performance constraint of the currently running task, the task will be migrated. A similar goal is targeted by the work in [79], but with considering a tiled architecture, in which each tile has its own domain of V/f levels and the cores inside each tile share the same V/f level. Therefore, a hierarchical thermal management approach is used by employing high-level agents at the tile level, and low-level agents at the core level.

As mentioned above, the techniques in this category do not have the same goal as this dissertation. Specifically, they optimize for temperature, while this dissertation optimizes for performance under temperature constraints.

2.3 Aging Management Techniques

This section discusses the state-of-the-art techniques that optimize for performance under an aging constraint (Section 2.3.1) and the techniques that optimize for aging at the system level (Section 2.3.2). Additionally, Section 2.3.3 summarizes guardbanding-aware techniques.

2.3.1 Performance Optimization under Aging Constraints

Few recent works share with this dissertation the goal of improving the performance under aging constraints, like those proposed in [81], [82], [83]. In [81], a resource management technique is presented to control the aging of the chip at runtime. If the aging constraint is violated at any point of time, the presented technique downscales the V/f levels of the cores proportional to the amount of the aging constraint violation. The purpose of the V/f level downscaling is decreasing the temperature of the cores and hence improving the aging. The drawback of this technique, however, is that it does not consider the positive impact of reducing the V_{dd} on the aging, and that, in turn, leads to dispensable V/f level downscaling and thereby performance losses. In [83], a runtime application mapping scheme is proposed aiming at maximizing the performance under an aging constraint, while considering applications with various communication characteristics. The proposed scheme meets the aging constraint in a long-term scale. That is, if the application is communication-intensive, it is mapped to a near-square area to reduce communication overhead. In contrast, the computation-intensive applications are mapped to an area with a higher dispersion to avoid increasing temperature and accelerating aging. The limitation of this technique is that it does not consider DVFS.

In contrast to these techniques, the work proposed in [82] considers the impact of both, the temperature and V_{dd}, on the aging. In particular, it estimates the maximum values of the voltage and the temperature that should not be exceeded in order to meet the targeted aging constraint. These values are used as references by a resource management technique that assigns the tasks to the cores and determines the V/f levels of the cores in order to improve the performance without exceeding the reference values of the voltage and temperature. This is, however, conservative, because it enforces one temperature constraint suitable to the selected reference value of the voltage rather than determining the temperature constraint based on the selected V/f level as proposed in Chapter 7.

In summary, state-of-the-art techniques aim at keeping the aging below a specific constraint, by enforcing conservative bounds on the temperature and V_{dd}. These conservative bounds hinder resource management at the system level from maximizing the performance.

2.3.2 Aging Optimization

Optimizing for aging means minimizing it or balancing it among the cores. That allows increasing processor lifetime. Several techniques have emerged aiming at minimizing aging effects by reducing the delays induced by aging mechanisms in multicores. In [84], a task mapping technique is proposed in order to minimize NBTI-induced aging while satisfying performance demands. In particular, the aging (i.e., ΔV_{th}) on the cores is monitored using aging sensors as described in [85]. When a core becomes weak according to a predetermined weakness-criterion, the technique checks the availability of a healthier core. If there is a healthier core available, the task will be migrated to it allowing to reduce the temperature of the weak core and hence it can recover. Otherwise, the task will not be migrated in order to keep satisfying performance demands. The work presented in [86] aims at minimizing and balancing the NBTI-induced aging as much as possible without considering a performance constraint. Specifically, it balances the workload and thus the temperatures among healthy cores only, while keeping weak cores inactive in order to recover. An aging aware register file allocator (*ARGO*) is proposed in [87] for general-purpose graphic processing units (GPGPUs). This allocator uniformly distributes the stress and hence the generated heat throughout the register file without any performance penalty. That allows minimizing the aging on the register file, which is the most highly stressed GPGPU components. The technique presented in [88] assigns the tasks to the cores so that the peak temperature is minimized and the thermal cycles on the cores are mitigated. The goal is to reduce the aging effects induced by both NBTI and electro migration. All of these aforementioned techniques consider the temperature as the only means to decelerate aging. However, the voltage of the processor has even a more significant impact on aging mechanisms [89] and it should be considered.

An early approach proposed in [90] considers the impact of reducing voltage on aging, and therefore it statically reduces the maximum allowed voltage level of the processor at specific points during its lifetime, in order to decelerate the aging. A dynamic reliability management is presented in [91] that employs DVFS, in order to mitigate NBTI- and TDDP-induced aging effects. A comprehensive aging management (*ARTEMIS*) is proposed in [92] that aims at minimizing the aging in both the cores and the NoC routers. This is achieved through balancing both computations and communications among cores and routers by means of application-to-core mapping, NoC routing path allocation, and DVFS.

As can be noticed, some of these techniques (e.g., [87]) decrease the temperature of the cores to reduce aging, while others (e.g., [92]) decrease both the voltage and the temperature, resulting in a more significant reduction in aging. Regardless of their utilized means to minimize the aging, these techniques target a different goal from the targeted one in this dissertation, which is optimizing performance under temperature and aging constraints.

2.3.3 Guardbanding Techniques

Typically, the required guardband width is estimated at the design time of the processor (circuit level) either by means of aging models [47, 93] or through measuring the increase in the processor delay after testing a prototyping chip under accelerated aging conditions [94]. Hence, the guardband width is set to compensate for worst-case delays. This is, however, pessimistic and prevents exploiting the current capabilities of the processor. Motivated by that, guardbanding techniques (e.g., [95–98]) have emerged to adapt the guardband width to compensate for the actual delay increase in the processor's paths at runtime. The technique of [95] proposes to adapt the width of the guardband that compensates for the delays induced by long-term aging effects in order to maintain the optimal performance of an aged circuit. A more recent technique [28] presents an adaptive guardbanding that compensates for the delays induced by both long-term and short-term aging effects. There is another class of techniques that focuses on reducing the guardband that compensates for voltage noises and fluctuations, e.g., [97, 98]. In particular, in [97] a thread scheduler is proposed in order to control the activity within the multicores so that the voltage is smoothed and thereby a reduced voltage guardband will be enough to avoid timing errors. In [98], a synchronization mechanism between threads is used in order to eliminate voltage fluctuations, thereby reducing the voltage guardband. A more comprehensive guardbanding technique is presented in [96] determines the width of the necessary guardband to sustain reliability according to the monitored critical path delay of the processor at runtime, which is influenced by both voltage fluctuations and aging effects.

More precisely, although these guardbanding techniques adapt the guardband width at the system level considering the workload, they still employ one fixed guardband type throughout the lifetime of the processor. In contrast, this dissertation suggests selecting the guardband type dynamically with regard to the running workloads, which allows additional potentials for performance optimization.

Chapter 3

System Model

This dissertation considers both homogeneous and heterogeneous multicore systems. This chapter illustrates the adopted hardware model, power model, thermal model and application model that are used for these multicore systems.

3.1 Hadware Model

The considered multicore systems are tiled architectures with L tiles. Every tile ℓ has N_ℓ cores. $N_\ell \in [1, N]$, where N is the total number of cores, i.e., $N = \sum_{\ell=1}^{L} N_\ell$. In homogenous multicore systems, the cores are identical and N_ℓ is identical for all tiles. While in heterogeneous systems, the tiles might have different types of cores $\tau_1, \ldots \tau_{max}$ and also different N_ℓ. Each tile has an L2 cache memory shared between its cores. Moreover, the L2 caches of the tiles are fully coherent.

This dissertation focuses on two DVFS policies; per-core DVFS and per-tile DVFS. In per-core DVFS policy, each core may operate at different voltage V_{dd} and frequency f levels. A minimum voltage V_{dd} is necessary for a core to stably run at a given frequency f. In order to avoid consuming unnecessary power, the voltage V_{dd} of a core is always set to the minimum value so that a stable execution can be achieved on the core for the desired frequency f. These V/f levels are enumerated as $\{VF_1, VF_2, \ldots, VF_Y\}$, where Y is the number of the V/f levels available on the chip. Hence, VF_Y is the maximum V/f level of the cores. The selected V/f level for core i is denoted as vf_i^{core}. In per-tile DVFS policy, all cores inside a tile share the same V/f level. In heterogeneous systems, each tile has its own range of V/f levels. We enumerate the available V/f levels of tile ℓ as $\{VF_{\ell,1}, VF_{\ell,2}, \ldots, VF_{\ell,y}\}$, where $VF_{\ell,y}$ is the maximum V/f level of tile ℓ. The selected V/f level for tile ℓ is denoted as vf_ℓ^{tile}.

Each core has two power states: active (i.e., power-on) or dark (i.e., power-gated). Active cores can run at different speeds according to their V/f levels. The total power consumption of core i is denoted as p_i and comprises dynamic and leakage power. Dynamic power relies on workload activities on the core, i.e, the running application on the core and the selected V/f level of the core as shown later in the application model. However, the leakage power relies on only the voltage V_{dd} and the core's temperature. The dynamic and leakage power consumptions of the cores are obtained using the state-of-the-art power model McPAT [40] as elaborated later in the next chapter.

3.2 Application Model

The application model consists of K multi-threaded applications, of which each can be mapped to a set of cores so that application threads can run in parallel on these cores. Each core can execute one thread at a time; which is a suitable model for multicore systems [99]. These applications are *malleable*, i.e., they can adapt the number of threads dynamically at runtime according to the number of cores assigned to them [100]. The number of threads of application k at a given time point is denoted as h_k, such that $1 \leq h_k \leq H_k$, where H_k is the maximum number of threads of application k. To identify application-to-tile assignment, a binary matrix $\mathbf{A} = [a_{\ell,k}]_{L\times K}$ is defined. If application k is assigned to tile ℓ then $a_{\ell,k} = 1$, and $a_{\ell,k} = 0$ otherwise. We assume that multiple applications can be assigned to one tile, but each application is executed on only one tile at any given time, such that $\sum_{\ell=1}^{L} a_{\ell,k} \leq 1$ for all $k = 1, 2, \ldots, K$. Similarly, to represent application-to-core mapping, a binary matrix $\mathbf{M} = [m_{i,k}]_{N\times K}$ is defined. If application k is mapped to core i then $m_{i,k} = 1$, and $m_{i,k} = 0$ otherwise. Since only one application thread can be executed at one core at a time, that implies $\sum_{k=1}^{K} m_{i,k} \leq 1$ for all $i = 1, 2, \ldots, N$. Moreover, it holds that $\sum_{i=1}^{N} m_{i,k} = h_k$ for all $k = 1, 2, \ldots, K$.

The set of cores that application k is mapped to is referred to as map_k. To avoid synchronization stalls between the threads of an application, the cores of each application (map_k) must be executed at the same speed. Thus, at a given time point, the cores that execute application k must share the same V/f level denoted as vf_k.

As aforementioned, the dynamic power of the cores is related to the running application. The resulting power consumption of running the application is quantified according to its number of threads h_k and the V/f level vf_k that its cores run at. Therefore, a table P_k for each application is defined to store the average power consumption for all possible values of the number of threads, $h = 1, 2, \ldots, H_k$, and all available V/f levels, $vf = VF_1, VF_2, \ldots, VF_Y$. When an application thread is mapped to a core, the average power consumption on that core will be equal to $P_k(h_k, vf_k)/h_k$, assuming that the resulting average power consumptions on all application cores are similar.

Throughout this dissertation, two application performance metrics are used. The first metric (used in Chapters 5, 6) defines the application performance as the inverse function of the execution time of the application. This performance metric is shown in Equation (3.1), in which $E_k(h, vf)$ indicates the execution time of application k, when running h parallel threads and the V/f level of the cores is set to vf, and $R_k(h, vf)$ represents the corresponding performance.

$$R_k(h, vf) = \frac{1}{E_k(h, vf)} \tag{3.1}$$

As seen in Equation (3.1), the execution time and thus the performance is related to the number of threads the V/f level of the application. When h_k and/or vf_k are increased, the execution time of the application will be reduced, and hence the application performance becomes higher.

The second metric (used in Chapters 7, 8, 9) defines the application performance as a ratio of the Instructions Per Second (IPS) of the application to its maximum IPS that can be obtained at the maximum number of threads H_k and the maximum V/f level.

This performance metric is shown in Equation (3.2).

$$R_k(h, vf) = \frac{IPS_k(h, vf)}{IPS_k(H_k, VF_Y)} \in [0, 1] \tag{3.2}$$

Hence, the maximum possible performance for all applications is 1. This performance metric enables direct and fair comparisons between diverse applications, because the IPS of different applications might have a different order of magnitude and cannot be directly compared without this normalization.

Similar to the power table P_k, the table R_k is defined to store the performance of application k for all possible number of threads $h = 1 \dots H_k$, and all available V/f levels, $vf = VF_1, VF_2, \dots, VF_Y$.

In the case of heterogeneous cores, a new dimension representing the core type must be added to these tables, in order to store the resulting power and the performance of application k considering all available core types $\tau_1, \dots \tau_{max}$, as shown in Figure 3.1. It is important to note that these tables are obtained by profiling the applications at design time as explained later in Chapter 4 and considered as inputs to the presented techniques in Chapters 5, 6, 7, 8, and 9. Nevertheless, these techniques are not limited to how these profiles are obtained; they can be obtained using other profiling tools like [101, 102] either at design time or at runtime (similar to the technique proposed in [103]).

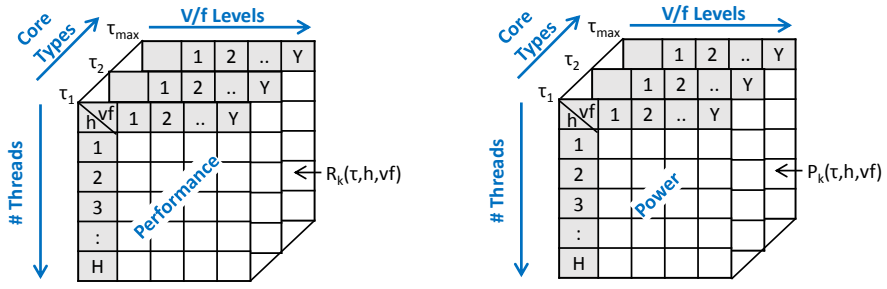


Figure 3.1: The performance and power tables for the applications.

3.3 Thermal Model

The most widely thermal model adopted in architectural-level thermal analysis is the RC thermal model [104], which is based on the well-known duality between thermal and electrical circuits [105]. In particular, the heat flow can be described as a current. The passing of this heat flow through a thermal resistance leads to a temperature difference equivalent to the voltage drop resulting from the passing of the current through the electrical resistance. Although the power flow changes instantaneously , there is a delay before the temperature changes and reaches a steady state. Describing the heat flow during time requires thermal capacitance. The combination of thermal resistance and capacitance leads to exponential rise and fall times characterized by a thermal RC time constant similar to the electrical RC constant.

This dissertation adopts an RC thermal network similar to [104], which models the heat dissipation from the die to the ambient temperature. Figure 3.2-a shows the stacked layers between the die and the ambient temperature that are considered by this thermal

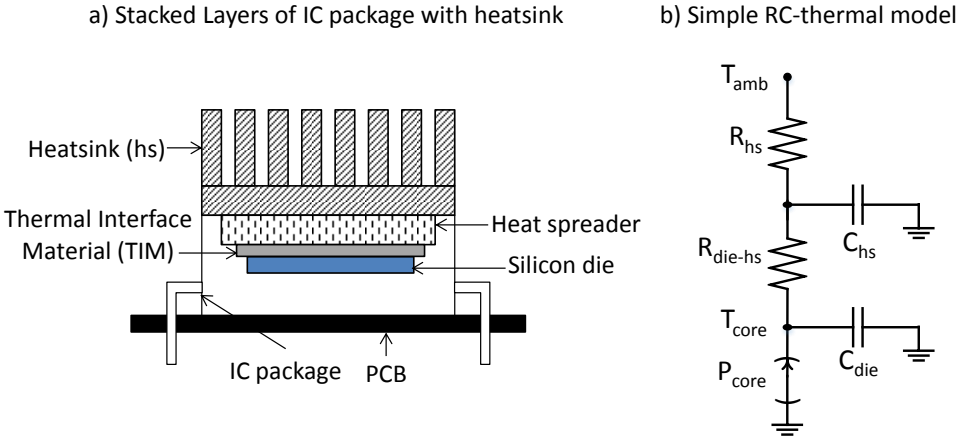


Figure 3.2: Sub-figure (a) illustrates the IC package layers considered by the employed RC-thermal model in this dissertation. Sub-figure (b) shows a simplified RC-thermal model.

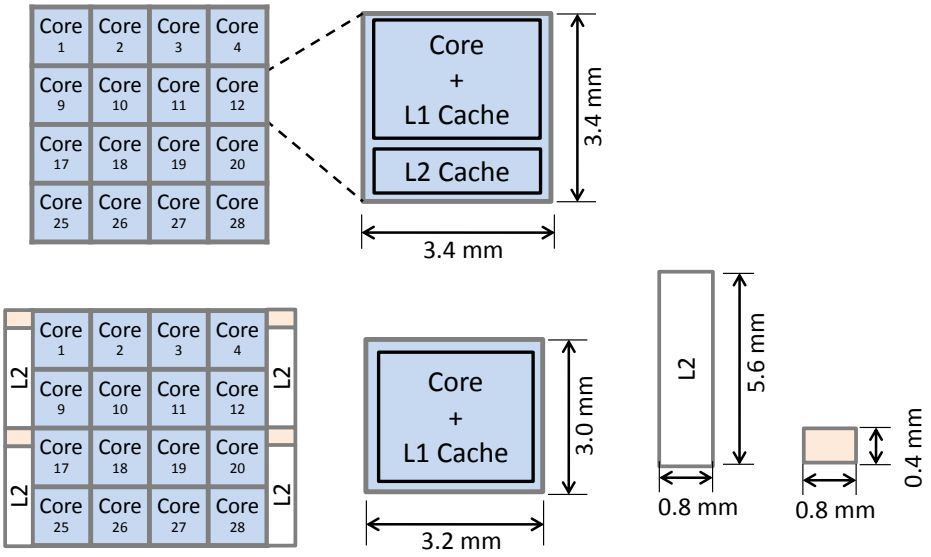


Figure 3.3: Examples of floorplans that can be considered by the RC-thermal model.

network. Figure 3.2-b illustrates a simplified RC thermal network that consists of only three thermal nodes which represent the die, the heatsink, and the ambient temperature T_{amb}. The temperature associated with each thermal node is represented by the voltage on the node. Thermal nodes are interconnected with each other through thermal resistances. There is a thermal capacitance associated with every thermal node, except the thermal node of the ambient temperature, because the latter is considered to be constant for a long time. Only the thermal nodes that represent the die layer will be connected with current sources to represent the power consumptions on the die.

The adopted thermal network in this dissertation is composed of Z thermal nodes to represent the die, Thermal Interface Material (TIM), the heat spreader and the heatsink. There are at least N thermal nodes, which belong to the die layer, so that each core on the chip is represented by one thermal node. Besides the cores, there might be other blocks on the chip, like L2 caches and memory controllers, which can be represented by additional thermal nodes.

Figure 3.3 shows two examples of 4x4 homogenous multicore floorplans, where each block on a floorplan corresponds to a thermal node in the RC thermal network. In the first floorplan, each block represents a core, an L2 cache, or a memory controller. In the second floorplan, the blocks have higher granularity, so that each block represents a core with its L2 cache. The power consumptions of the cores and the other blocks on the floorplan are the heat sources on the chip, and thereby the corresponding thermal nodes in the RC thermal network are associated with current sources, as seen in Figure 3.2.

The temperatures of the thermal nodes can be calculated using the following system of differential equations[1]:

$$\mathbf{\Lambda T'} + \widetilde{\mathbf{B}}\mathbf{T} = \mathbf{P} + T_{\text{amb}}\mathbf{G} \tag{3.3}$$

There, matrix $\mathbf{\Lambda}$ is a diagonal matrix that contains the thermal capacitance values of the thermal nodes. Matrix $\widetilde{\mathbf{B}}$ contains the thermal conductance values between the nodes, the values in column vector $\mathbf{T} = [T_i(t)]_{Z\times 1}$ are the temperatures on the thermal nodes, where the first-order derivative of these temperatures with respect to time are stored in column vector $\mathbf{T'}$. The power consumptions of the thermal nodes are stored in column vector $\mathbf{P} = [p_i]_{Z\times 1}$. Column vector $\mathbf{G} = [g_i]_{Z\times 1}$ contains the values of the thermal conductance between each node and the ambient temperature T_{amb}.

When only considering the steady-state temperatures, the first-order derivative of the temperature becomes zero, and thus (3.3) can be re-written as follows:

$$\mathbf{T}_{steady} = \mathbf{BP} + T_{\text{amb}}\mathbf{BG} \tag{3.4}$$

The values in column vector $\mathbf{T}_{steady} = [T_i]_{Z\times 1}$ are the steady-state temperatures on the thermal nodes. Matrix $\mathbf{B} = [b_{i,j}]_{Z\times Z}$ is the inverse of Matrix $\widetilde{\mathbf{B}}$ and it contains the amount of the heat contribution of the thermal nodes.

Thus, the steady-state temperature of core i, can be calculated as follows:

$$T_i = \sum_{j=1}^{Z} b_{i,j} \cdot p_j + T_{\text{amb}} \sum_{j=1}^{Z} b_{i,j} \cdot g_j \tag{3.5}$$

There, $b_{i,j}$ represents the heat contribution of node j into the temperature of node i, p_j is the power consumption of node j, and g_j corresponds to the thermal conductance between node j and T_{amb}. For each core i,a constant c_i is defined, such that $c_i = \sum_{j=1}^{Z} b_{i,j} \cdot g_j$, since this term is a constant for each core. By substituting c_i in Equation (3.5), the following equation is obtained:

$$T_i = \sum_{j=1}^{Z} b_{i,j} \cdot p_j + T_{\text{amb}} \cdot c_i \tag{3.6}$$

Additionally, we can estimate the change in the steady-state temperature resulting due to a change in the core power Δp_j, as follows:

$$\Delta T_i = \sum_{j=1}^{N} b_{i,j} \cdot \Delta p_j \tag{3.7}$$

[1]The details of mathematical steps to reach to (3.3) are elaborated in details in [104].

For the considered multicore systems, the critical temperature (under which it is deemed to be a safe range) is denoted as T_{crit}.

3.4 Aging Model

In order to model the impact of aging, different abstraction levels must be jointly considered starting from the level where aging originates (i.e. the physical level) all the way up to the level where they manifest themselves as delay increases (i.e., the circuit level).

Physical Level: In this dissertation, state-of-the-art physics-based aging model [43, 106] is employed. It precisely estimates the resulting number of *interface* and *oxide* traps (i.e. ΔN_{IT} and ΔN_{OT}) due to Negative Bias Temperature Instability (NBTI) and Positive Bias Temperature Instability (PBTI) effects for various *aging stress conditions*. Each of these conditions corresponds to a combination of specific voltage (V_{dd}) and temperature values that are applied to pMOS and nMOS transistors. The worst-case aging stress corresponds to the combination of the maximum voltage V_{dd}^{max} and the worst-case temperature, T_{crit}. The model has been validated against semiconductor measurements for wide ranges of V_{dd} and temperature and the detailed equations and validations are presented in [106].

Transistor Level: After estimating the number of *interface* and *oxide* traps (i.e. ΔN_{IT} and ΔN_{OT}), the resulting threshold voltage increase ΔV_{th} can be estimated using Equation (3.8) similar to [43].

$$\Delta V_{th} = \frac{q}{C_{ox}} \cdot (\Delta N_{IT} + \Delta N_{OT}) \tag{3.8}$$

Figure 3.4 shows the estimated ΔV_{th} for various aging stress conditions that correspond to the combinations of various temperature and V_{dd} values in the ranges $[0\,°\mathrm{C}, \cdots, 100\,°\mathrm{C}]$ and $[0.6\,\mathrm{V}, \cdots, 1.4\,\mathrm{V}]$, respectively.

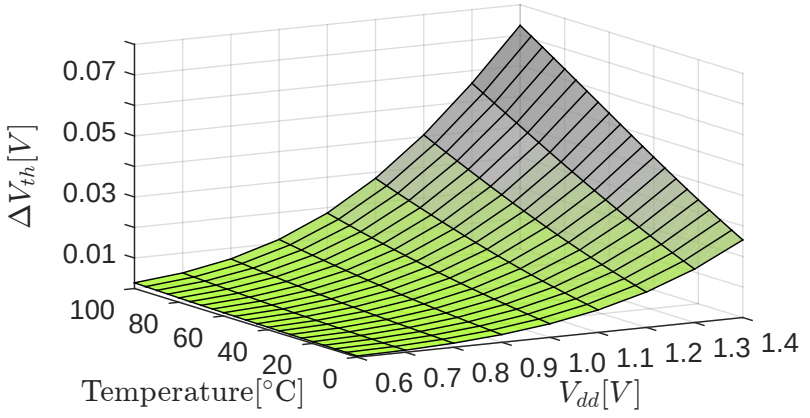


Figure 3.4: The increase in V_{th} (i.e. aging) as a function of V_{dd} and temperature obtained with a physics-based aging model.

Gate Level: Afterwards, any estimated ΔV_{th} value can be translated into the corresponding delay increase at the gate level. To achieve that, varied combinational and sequential cells from the open-source "NanGate" standard cell library [107, 108] have been re-characterized using SPICE [109] to create an *aging-aware* standard cell library in which the delay of every gate has been estimated under the effect that ΔV_{th} has on pMOS and nMOS transistors. This is achieved by applying similar methods to what

proposed in [27, 110], For every gate, the SPICE simulation employs the Berkeley Short-channel IGFET Model (BSIM) [44] along with the Predictive Technology Model (PTM) of pMOS/nMOS transistors [111] after setting the threshold voltage based on the estimated increase (ΔV_{th}) at the transistor level.

Circuit Level: To analyze the impact of aging on increasing the delay of paths of a circuit, a Static Timing Analysis (STA) available in standard EDA tool flows (e.g. Synopsys [109]) is conducted for the circuit's netlist using the *aging-aware* cell library created in the previous step. This enables an accurately estimation of the total delay increase due to aging in a circuit. This analysis is conducted for the open-source Berkeley out-of-order Machine (BOOM) [112].

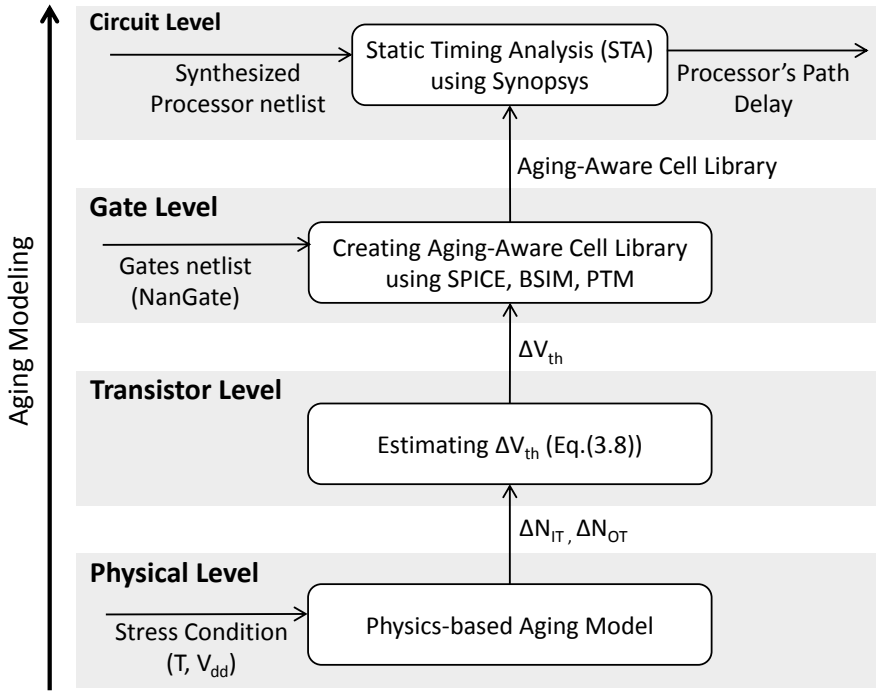


Figure 3.5: The adopted flow of modeling aging from physical level to circuit level [27, 110, 113].

3.5 Guardband Estimation

As aforementioned, the amount of aging, i.e., ΔV_{th}, can be estimated for any given stress condition which corresponds to a combination of specific temperature and V_{dd}. For estimating the required timing guardband (t_{GB}) to compensate a specific ΔV_{th}, it is necessary to estimate first the delay increase in the processor's paths caused by ΔV_{th} (aging-induced delay) through conducting the steps shown in Figure 3.5.

Since the adopted system model in this dissertation supports several V/f levels, and hence various V_{dd} values, we estimate the aging-induced delay resulting from each of the available V_{dd} values along with the maximum allowed temperature (i.e., T_{crit}). Thus, the difference between each of these delays and the corresponding processor's clock delay (without aging) indicates the required timing guardband. Figure 3.6 illustrates the process of estimating the timing guardbands.

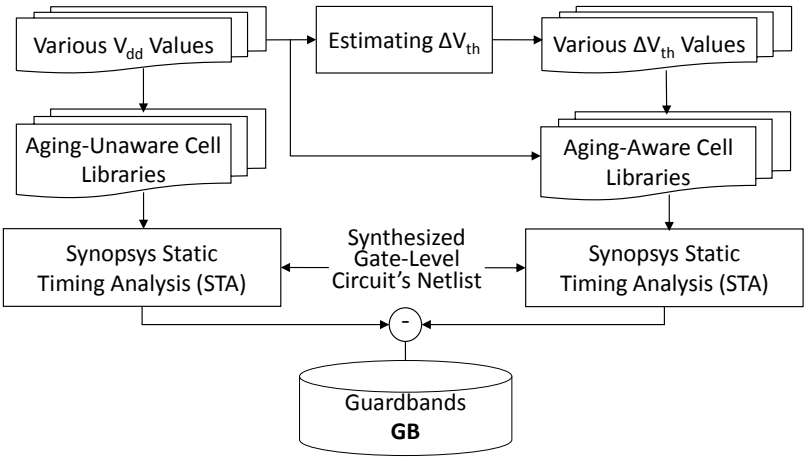


Figure 3.6: Estimation of timing guardbands considering considering various V_{dd} values.

In Chapter 9, different guardband types are utilized; which are voltage guardband (V_GB), and frequency guardband (F_GB), but only the maximum V/f level, i.e., VF_Y is used. Thus, estimating the width of voltage guardband is straightforward; it is equal to ΔV_{th} at the maximum V_{dd} as shown in Equation (1.3). While estimating the width of frequency guardband requires first estimating the timing guardband t_{GB} similar to what explained above but only for the maximum V_{dd} value. After estimating t_{GB}, the corresponding frequency guardband is calculated as shown in Equation (1.2).

Chapter 4

Experimental Framework

For the evaluation of the resource management techniques that will be presented in the following chapters, a common experimental setup is utilized, while different homogenous and heterogeneous multicore architectures are used. In homogeneous architectures, out-of-order (O3) Alpha cores [114] are used. While in heterogeneous architectures, four types of cores are used, which are O3 Alpha cores, simple Alpha cores, in-order ARM-A7 cores, and O3 ARM-A15 cores. In addition to these cores, the Tightly-Coupled Processor Arrays (TCPAs) are used as accelerators, further details about it are explained in Section 4.1.1. In this chapter, the common experimental setup and the various multicore architectures are illustrated.

4.1 Setup

An overview of the experimental framework is shown in Figure 4.1. This experimental framework integrates the gem5 [39] and McPAT [40] simulators to simulate Alpha cores using full-system mode. Additionally (for heterogeneous architecture), it integrates real measurements for ARM cores from an Exynos 5 Octa (5422) chip [66] with ARM's "big.LITTLE" architecture and the post-synthesis results for TCPA from the Cadence Encounter RTL Compiler.

For benchmarks, realistic multi-threaded applications from the PARSEC benchmark suite [41] are utilized. This suit was designed to be representative of next-generation shared-memory applications for multicore architectures. It contains different applications from many different areas such as computer vision, video encoding, and financial analytics, etc. These applications can be executed on Alpha and ARM cores. Besides PARSEC applications, additional computationally-intensive applications are used, which are "SAD", "Edge Detection", "FIR", "Optical Flow", "Matrix Multiplication", and "Harris Corner" that can be executed either on Alpha and ARM cores as single-threaded applications, or on the adopted TCPA accelerator, which provides the ability of parallelizing these applications to speedup the execution.

For the simulations with gem5 and McPAT, each application is first executed in gem5 on the different types of Alpha cores (O3 and simple) and for all possible V/f levels and for different numbers of parallel threads. From the output statistics of gem5, the performance tables of the applications are built as illustrated in Section 3.2 and also

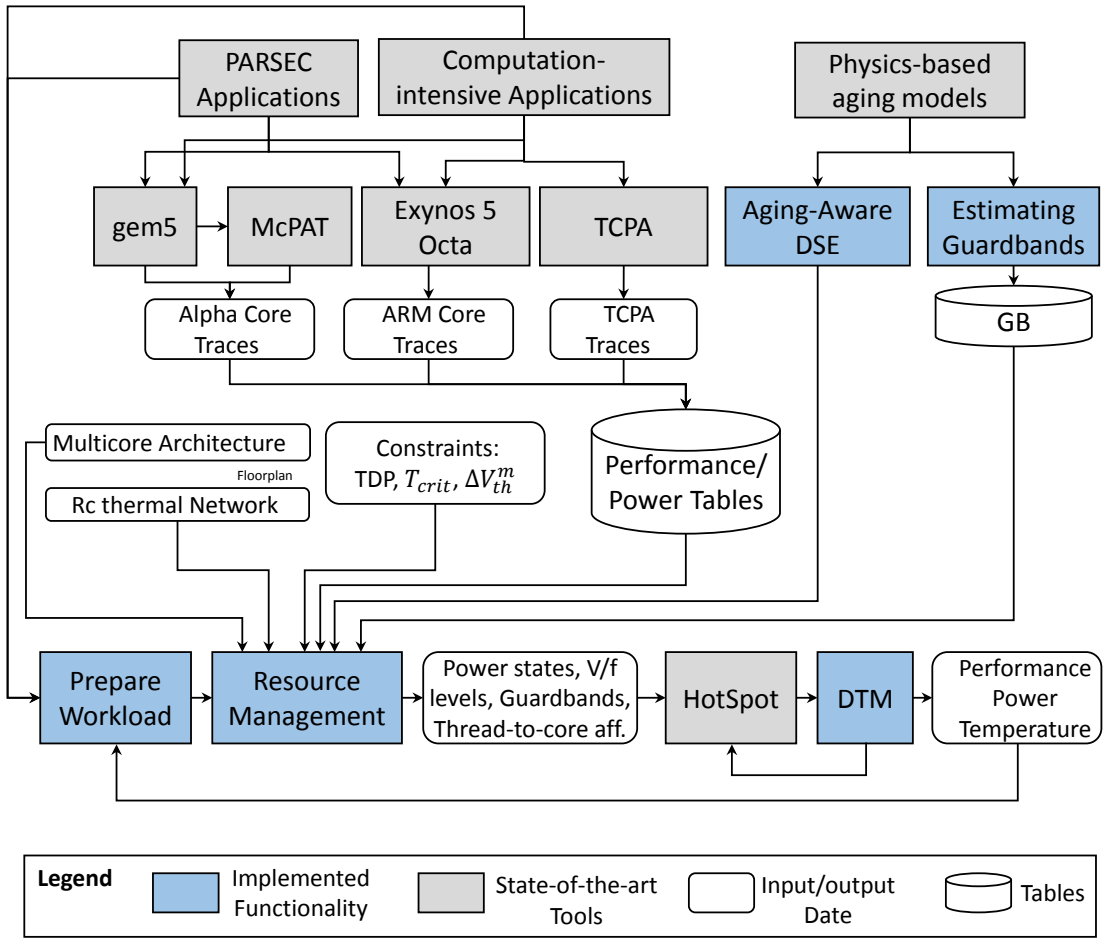


Figure 4.1: Overview of the experimental framework employed to evaluate the resource management techniques presented in this dissertation.

the relevant statistics required by McPAT are extracted. Then, McPAT is executed in order to estimate the areas and the power consumptions of the cores and other blocks in the architecture, like L2 caches and memory controllers. These power statistics are used to build the power tables of the applications, while the estimated areas of the different blocks are utilized to build the floorplans of the employed mutlicore architectures as will be explained in Section 4.2. For the measurements of ARM cores on the Odroid-XU3 platform, each application is executed on the different types of ARM cores (O3 and in-order) for all possible V/f levels and for different number of parallel threads. During execution, the power sensors of the cores are read and at the end the total execution time is measured. For running the applications on TCPA, the codes are generated using PARO [115], where the problem of resource constrained scheduling and binding is solved by mixed integer linear programming (MILP) as described in [116]. PARO contains a back end targeting programmable hardware accelerators. Here, for mapping loop programs, symbolic parallelization techniques [117] are implemented as a compiler transformation to partition and schedule loop nests using parametrized tile sizes. The power and performance traces for the TCPA are derived based on post-synthesis results from the Cadence Encounter RTL Compiler. Similar to the traces of Alpha cores, the measurements for ARM cores and the traces for TCPA will be also used to build the corresponding power and performance tables for the applications.

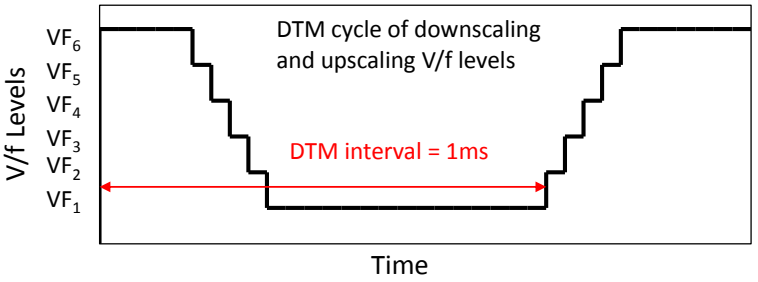


Figure 4.2: Illustrating a control step of a standard DTM that typically implemented in commercial processors. The figure is adapted from the data sheet of Intel Xeon Phi [118].

The proposed resource management techniques need to estimate the temperatures within their decision making process. Therefore, a thermal model is employed based on RC-thermal network as explained in Section 3.3. To obtain the required metrics of the adopted RC-thermal network, such as **B** and **G** in Equation (3.4), the HotSpot simulator is utilized with its default configurations, which are: Chip thickness of 0.15 mm, silicon thermal conductivity of 100 $\frac{\mathrm{W}}{\mathrm{m}\cdot\mathrm{K}}$, silicon specific heat of $1.75 \cdot 10^6 \frac{\mathrm{J}}{\mathrm{m}^3\cdot\mathrm{K}}$, a heat sink of 6×6 cm and 6.9 mm thick, heat sink convection capacitance of 140.4 $\frac{\mathrm{J}}{\mathrm{K}}$, heat sink convection resistance of 0.1 $\frac{\mathrm{K}}{\mathrm{W}}$, heat sink and heat spreader thermal conductivity of 400 $\frac{\mathrm{W}}{\mathrm{m}\cdot\mathrm{K}}$, heat sink and heat spreader specific heat of $3.55{\cdot}10^6$ $\frac{\mathrm{J}}{\mathrm{m}^3\cdot\mathrm{K}}$, a heat spreader of 3×3 cm and 1 mm thick, interface material thickness of 20 um, interface material thermal conductivity of 4 $\frac{\mathrm{W}}{\mathrm{m}\cdot\mathrm{K}}$, and interface material specific heat of $4 \cdot 10^6$ $\frac{\mathrm{J}}{\mathrm{m}^3\cdot\mathrm{K}}$. The ambient temperature is 45 °C.

Moreover, this framework incorporates the aging modeling explained in Figure 3.5 in order to enable the aging-ware design space exploration that will be presented in Chapter 7. Also, the guardband estimation process (Figure 3.6) is incorporated to enable resource management techniques to apply the required guardbands for sustaining reliability.

All of the previous steps are required to be prepared for one time (at design time) before running the management loop. In particular, this framework employs a management loop that runs at each time interval of 10 ms, similar to the scheduling epoch in Linux. In this management loop, the proposed techniques and the comparison candidates (both represented in this framework as "Resource Management") will first make their decisions of allocating the resources to the applications of the current workload and mapping these applications to the cores. The resulting transient temperature will be calculated using the HotSpot simulator. We assume that the standard DTM technique is implemented on the chip similar to commercial processors [33]. Thus, if the transient temperature of any core exceeds the critical temperature T_{crit}, DTM will be triggered to cool down the cores. DTM has a control interval of 1 ms, since it is supposed to be implemented on the hardware to provide a quick reaction to thermal violations. Thus, at each control interval, DTM examines the transient temperatures and downscales the V/f levels to the minimum level, if the temperature exceeds T_{crit}. Otherwise, if it drops back below T_{crit}, DTM will upscale the V/f levels again to the original V/f level determined by the resource management technique. Figure 4.2 shows details of the DTM technique implemented in Intel Xeon Phi according to its data sheet. It is important to be noted that not all of the presented resource management techniques will necessarily be executed at each time interval. Some of them will be executed only when the workload changes, some of them accept only a static workload and therefore they will be executed only one time at the beginning of the management loop. The execution details of each resource management technique will be clarified in the respective chapters.

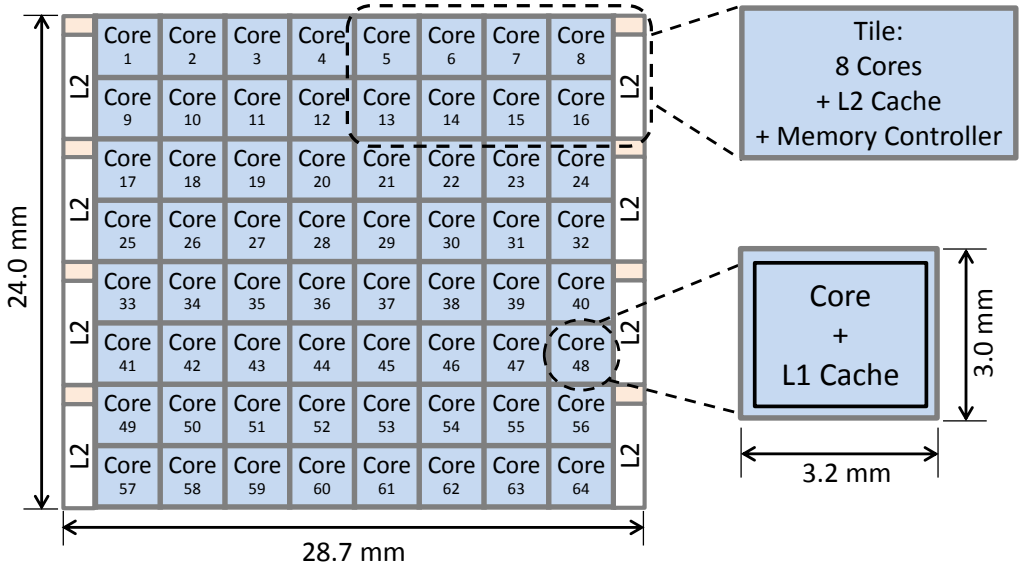


Figure 4.3: The floorplan of a 64-core chip adopted in Chapter 5.

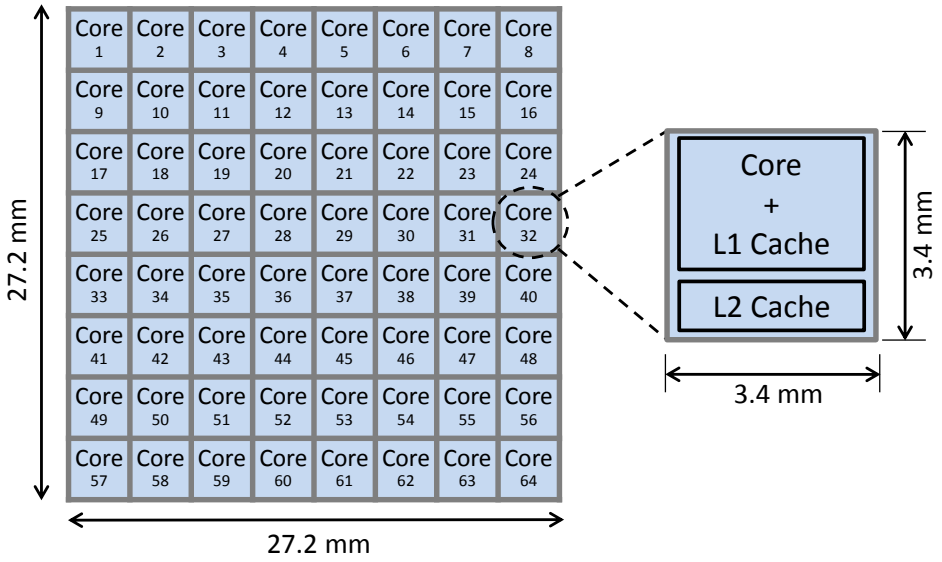


Figure 4.4: The floorplan of a 64-core homogeneous chip adopted in Chapters 7, 8, and 9.

4.1.1 Tightly-Coupled Processor Arrays (TCPA)

The TCPA tile used in our experiments [119, 120] is a highly parameterizable architecture template, and thus offers a high degree of flexibility. The heart of this accelerator tile comprises a massively parallel array of tightly coupled Very Long Instruction Word (VLIW) Processing Elements (PEs) complemented by peripheral components such as I/O buffers as well as several control, configuration, and communication companions. This TCPA can exploit a parallel and direct PE-to-PE communication, where data is streaming from the surrounding buffers through the array. Through the VLIW nature of each PE and the parallel and synchronous execution of loop iterations assigned by the compiler to each PE, the hardware accelerator nicely exploits both instruction- and

loop-level parallelism while achieving a better energy efficiency compared with general purpose commercial off-the-shelf embedded processors [121]. Some parameters, such as the number of PEs, basic interconnect topology, the number of functional units as well as the register organization within the PEs, have to be defined at synthesis time, whereas other parameters such as programmable delays between neighbor processors and inter-PE interconnect selection can be reconfigured at runtime. Each PE at the boundary can read/write data directly from/to a local buffer connected to it and each PE can exchange data with its neighbor PE in a single cycle. In our experiments each TCPA consists of the same number of 16 PEs each.

Since TCPAs are envisioned to be used as programmable accelerators in MPSoCs, there is often the question how they differ from other accelerators, for example, embedded general purpose graphics processing units (GPGPU) such as those used in smart phones or tablets. From the architectural point of view, TCPAs and GPGPUs both have a huge number of light-weight processing units. However, there exist also a few fundamental differences. The programming model of GPGPU is limited to perfect SIMD parallelism and therefore may not handle any loop nests with loop-carried data dependencies, whereas the TCPA used in this paper supports well the general case of kernels for streaming applications including (multidimensional) loop-carried data dependencies. For that reason, a TCPA tile is adopted in the proposed heterogeneous chip that it is used as programmable massively parallel processor array to speedup a myriad of computationally intensive applications like linear algebra and image processing algorithms.

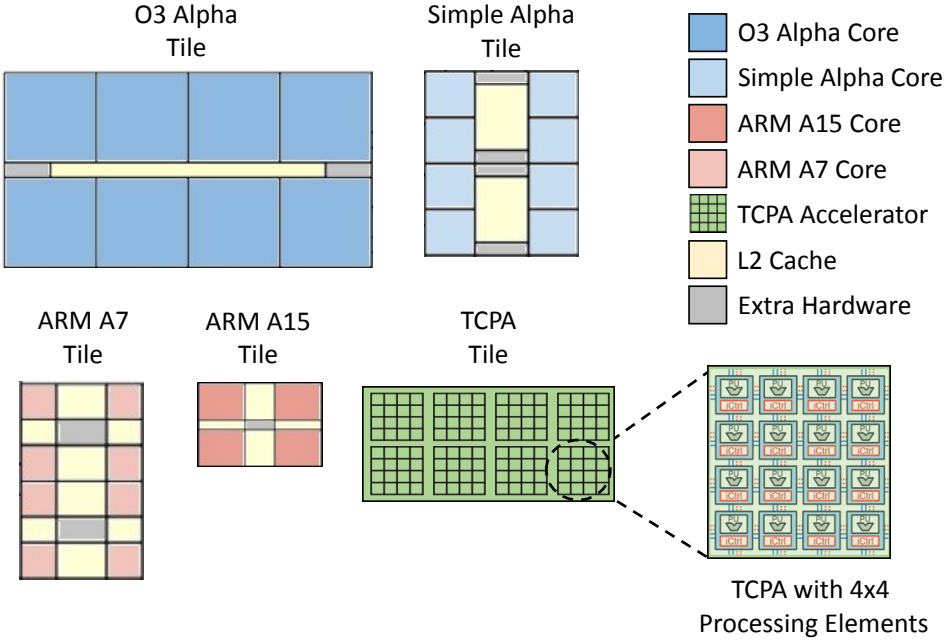


Figure 4.5: Five different types of tiles are considered in the heterogeneous architecture employed in Chapter 6.

4.2 Multicore Architectures

This section explains the homogenous and heterogeneous multicore architectures that are used for the experimental evaluations of the presented techniques in this dissertation.

4.2.1 Homogeneous Architecture

For our experiments in Chapter 5, a simulated 64-core chip is employed. This chip is tiled-architecture similar to Intel's Single Chip Cloud computer (SCC) [122], but with more cores and more tiles. The number of the tiles in the system is 8. Each tile has 8 cores and one shared L2 cache of 2 MB, and a memory controller. Each core is an out-of-order Alpha and composed by several units: an instruction fetch unit, an execution unit, a load and store unit, an out-of-order issue/dispatch, and a private L1 cache.

The floorplan of the chip that will be used by HotSpot consists of 80 thermal blocks, one for each core, and the rest of the blocks correspond to the L2 caches and the memory controllers. Figure 4.3 shows the adopted floorplan, with its dimensions, where each core has an area of 9.6 mm^2. The combined area of the L2 cache and the memory controller of each tile is 9.4 mm^2.

For the experiments of Chapters 7, 8, 9, a chip composed of 64 out-of-order Alpha cores is employed, and this chip has a 16 MB shared L2 cache physically distributed on the cores. The tile in this chip contains only one core and its L2 cache. The floorplan of the chip consists of 64 thermal blocks, one for each core and its L2 cache resulting in an area of 11.96 mm^2 for each block, as shown in Figure 4.4.

In these homogeneous cores, each core has its own voltage and frequency domain. The minimum V/f level corresponds to a V_{dd} of 0.45 V and a frequency of 0.2 GHz, while the maximum one corresponds to a V_{dd} of 0.45 V and a frequency of 0.2 GHz. The voltage and frequency steps between the V/f levels are 0.05 V and 0.2 GHz, respectively.

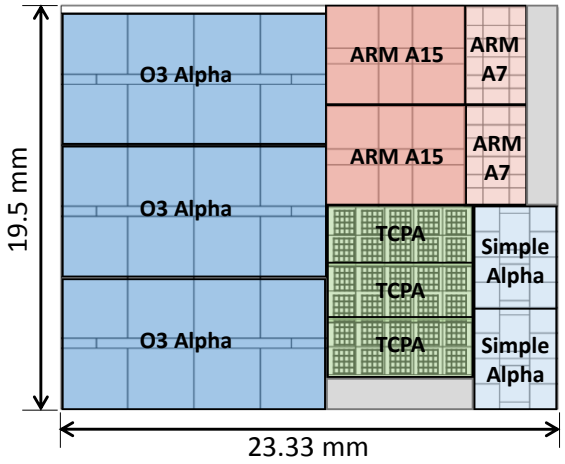


Figure 4.6: The floorplan of the adopted tiled heterogeneous multicore architecture in Chapter 6.

4.2.2 Heterogeneous Architecture

For the experiments in Chapter 6, a heterogeneous tiled multicore is considered with 94 heterogeneous cores, where 30 of them are TCPA accelerators (Figure 4.6). The adopted heterogeneous multicores consists of 12 tiles of 5 different types with respect to the type of the cores in the tile, as shown in Figure 4.5. Each tile has one domain of V/f levels,

Tile	**Core Type**	**#Cores / #Accelerators**	**Frequency Range of the supported V/f levels**
1, 2, 3	O3 Alpha	8	$\{0.2, 0.4, 0.6, \ldots, 4.0\}$ GHz
4, 5	Simple Alpha	16	$\{0.2, 0.4, 0.6, \ldots, 4.0\}$ GHz
6,7	ARM-A15	4	$\{1.2, 1.3, 1.4, \ldots, 2.0\}$ GHz
8,9	ARM-A7	16	$\{0.2, 0.3, 0.4, \ldots, 1.4\}$ GHz
10, 11, 12	TCPA	10	$\{50, 100, 150, \ldots, 650\}$ MHz

Table 4.1: Tile information for experimental evaluation.

such that all cores or accelerators inside a tile share the same V/f level at any given time point. The details of all tiles, including the type of cores, the number of cores, and the available V/f levels of the tiles, are shown in Table 4.1.

Chapter 5

Dark Silicon-Aware Resource Management

Traditional resource management techniques do not consider dark silicon problem. Specifically, they distribute the chip resources among the applications, considering that all cores can be active and run at the maximum speed. That, however, might lead to thermal violations, and hence DTM on the chip will be triggered. To cool down the cores, DTM downscales the V/f levels and/or power-gate some cores. As a result, DTM will lead to performance degradation, but it is necessary to save the chip from undesired overheating. To mitigate triggering DTM and its unpredictable performance losses, some resource management techniques (e.g., [57]) enforce TDP as a chip-level power constraint. Particularly, the number of active cores and their V/f levels are chosen so that TDP is not exceeded. However, restricting the system to run below TDP is not enough to avoid thermal violations as shown in the presented case study in Chapter 1. As a result, resource management techniques need to determine the number of active cores and the V/f levels considering a temperature constraint rather than considering only a power constraint.

In this chapter, we present a dark silicon-aware resource management technique, called *DsRM* [3, 6], that aims at maximizing the overall system performance under a temperature constraint. To achieve its goal, *DsRM* determines the number of active cores that should be allocated to each application and the V/f levels of these cores, and also maps the applications to the cores. Moreover, *DsRM* distributes the dark cores among the applications; that is for each application, some number of dark cores will be in close vicinity to its active cores to facilitate dissipating their generated heat. Importantly, these decisions are taken according to the TLP and the ILP of the applications. Specifically, to increase the performance of a high-ILP application, *DsRM* needs to upscale the V/f level of the active cores that execute that application. To enable this without violating the temperature constraint, *DsRM* sets more dark cores in close vicinity to the active cores of that application to dissipate the potential increase in the generated heat. Contrarily, to increase the performance of a high-TLP application, *DsRM* allocates more active cores to high-TLP application with lower V/f level.

In the following section,the problem of the proposed resource management technique is formulated. Section 5.2 presents details about our presented technique. The evaluation of *DsRM* is demonstrated in Section 5.3.

5.1 Problem Definition

For a given chip composed of N cores and a set of K applications, the focus of *DsRM* is to select for each application the number of active cores, which is equivalent to the number of its parallel threads, and the V/f level of the application cores, as well as the mapping of these applications to cores. The goal is to maximize the overall system performance under the temperature constraint T_{crit}. Mathematically, this problem can be expressed as finding h_k, vf_k, and map_k, for all $k = 1, 2, \ldots, K$, in order to:

$$\text{Maximize } \sum_{k=1}^{K} R_k(h_k, vf_k)$$

$$\text{Subject to: } \sum_{k=1}^{K} h_k \leq N \quad \text{and} \quad T_i < T_{crit} \qquad \text{for all} \quad i = 1, 2, \ldots, N$$

There, the application performance $R_k(h_k, vf_k)$ is estimated using Equation (3.1). T_i is the steady-state temperature of core i estimated using Equation (3.6). Finding the optimal solution by exhaustive search through all the possible combinations of map_k, h_k, vf_k for all applications requires exponential time complexity. Therefore, there is a need to decompose this problem into sub-problems. The first sub-problem is finding h_k and vf_k for all applications, so that the overall system performance, i.e., $\sum_{k=1}^{K} R_k(h_k, vf_k)$, is maximized, where the total number of allocated cores should not exceed N. However, the resulting solutions from solving this problem could lead to many thermal violations, because the highest V/f levels would be chosen, as the goal is to maximize the performance. Hence, in order to mitigate the potential thermal violations, TDP is considered as a power budget for the chip only in this sub-problem. Thus, the two constraints of the first sub-problem are as follows: $\sum_{k=1}^{K} h_k \leq N$ and $\sum_{k=1}^{K} P_k(h_k, vf_k) \leq TDP$. The second sub-problem is determining the mapping of the applications to the available cores, i.e., map_k, according to their allocated resources. The last sub-problem is adapting the application resources h_k, vf_k and the application mapping map_k to consider the temperature constraint T_{crit} and remove the TDP constraint. This resource adaptation should avoid thermal violations and exploit thermal headroom (if existing).

5.2 Dark Silicon-Aware Resource Management

The presented dark silicon-aware resource management technique, *DsRM*, manages the system resources and distributes them among multiple applications by performing the following three steps that correspond to the aforementioned sub-problems. At the first step, *DsRM* considers that TDP is the total power budget of the chip. Therefore, it optimally distributes both the cores and the TDP budget among the applications. To achieve this, *DsRM* employs a dynamic programing algorithm that jointly determines for each application the number of the cores (threads) and the V/f levels of these cores considering the TLP and ILP of the applications, so that the maximum overall system performance is obtained. At the second step, the applications are mapped to the cores according to their allocated resources in the first step. The goal of this step is to minimize the core temperatures, because this helps to reduce thermal violations on the chip. To achieve this, the available dark cores are distributed among the applications in proportional to their power consumptions. Then, the applications are mapped in such a way that the locations of active and dark cores of each application are close

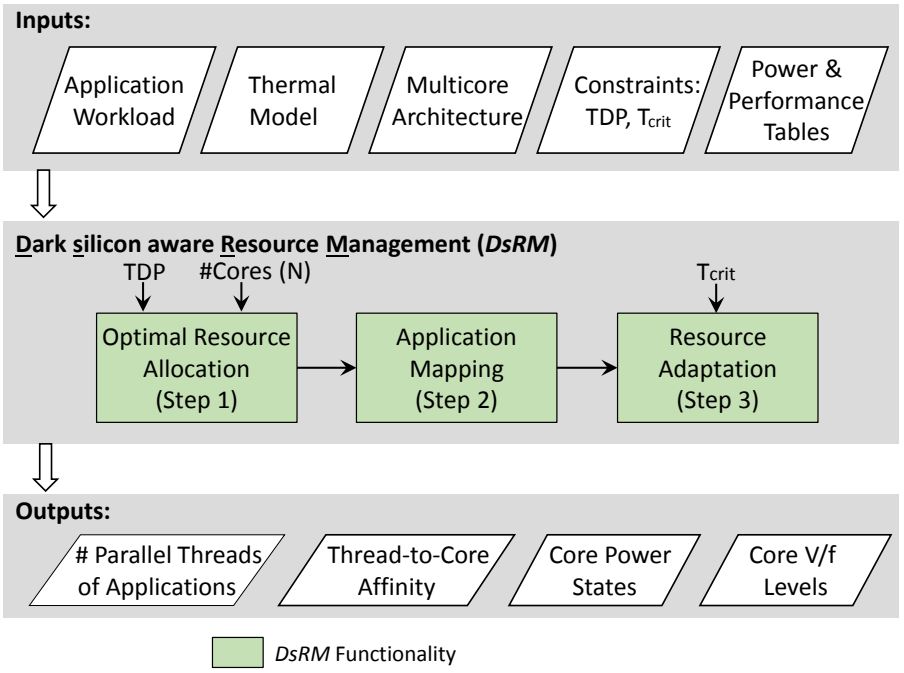


Figure 5.1: Overview of the presented dark silicon-aware resource management technique, *DsRM*.

to each other in order to allow the dark cores of each application to dissipate more heat from its active cores. The final step of *DsRM* considers the predefined temperature constraint and accordingly modifies the application resources obtained from the previous two steps in order to avoid thermal violations and exploit thermal headroom. In this step, TDP constraint is not considered and it might be exceeded if there is available thermal headroom and the resources are adapted to exploit it. Figure 5.1 shows an overview of the *DsRM* technique.

5.2.1 TDP-Constrained Optimal Resource Distribution

At the first step of *DsRM*, the system resources, i.e., the available cores and the power budget (TDP), will be distributed optimally among the applications in order to maximize the overall system performance, which is the first sub-problem defined in Section 5.1.

To solve this problem, *DsRM* employs an algorithm based on dynamic programming. Dynamic programming breaks down the problem into smaller sub-problems, and builds up the final solution gradually. The benefit here is that by storing the obtained sub-solutions at every step in tracking tables, the algorithm can reuse them in further steps, thus reducing the number of combinations that should be tested. That, in turn, reduces the complexity and the required time for finding the final optimal solution.

The inputs of this algorithm are the application set with their power and performance tables (see Section 3.2), besides the total number of cores N and the available power budget, referred to as p_{max}, which is equal to TDP. The output will be the optimal settings h_k, vf_k for all $k = 1, 2, \ldots, K$ applications, that result in the maximum overall system performance. To achieve this goal, the algorithm needs to divide the total available power budget p_{max} to smaller power budgets (portions) p to be distributed among

the applications similar to the cores. The final determined power quota for application k is denoted as q_k.

Before explaining how this algorithm works, it is necessary to define the following auxiliary function that the algorithm uses in finding the sub-solutions:
$MaxVF^{Budget}(k, n, p)$ returns the V/f level that results in the maximum performance for application k under a given power budget p, where the number of threads is equal to n. If there is no V/f level available for the given power budget p, this function returns 0. An example of the work of this function is illustrated in Figure 5.2.

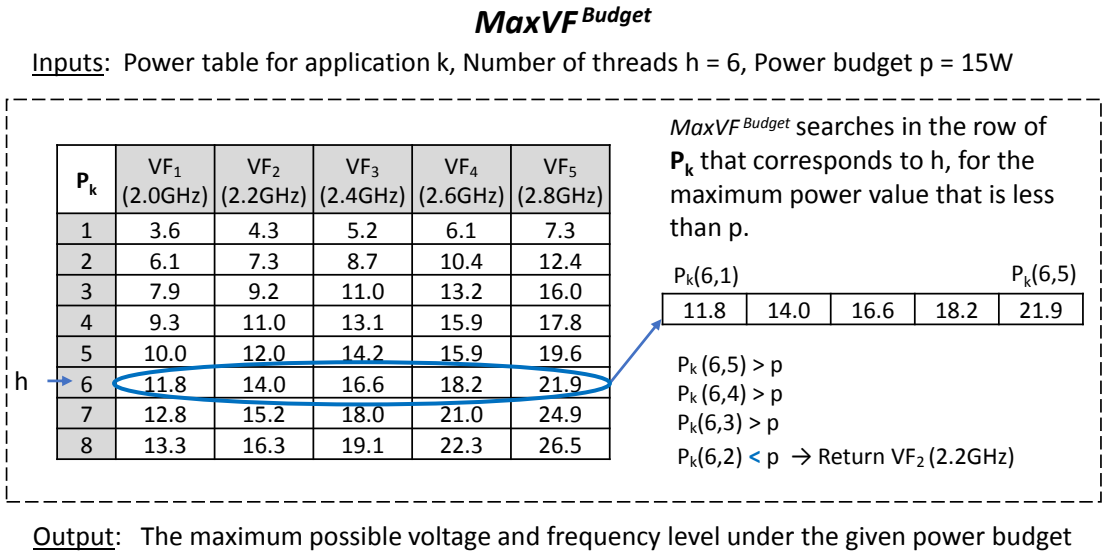


Figure 5.2: Example of the work of function $MaxVF^{Budget}(k, n, p)$.

To find the optimal solution, the algorithm builds a table $\mathbf{G^R}$, whose entries are represented by all possible unique triple (k, n, p), such that $k = 1, 2, \ldots, K$, $n = 1, 2, \ldots, N$, $p = p_1, p_2, \ldots, p_{max}$. Each cell of this table, $G^R(k, n, p)$, contains the maximum performance for the first k applications (assuming an arbitrary order for the applications) using n cores and a peak power consumption less than or equal to p. It starts from the first application $k = 1$ and calculates its maximum performance at all possible combinations between n and p, as shown in Equation (5.1). This equation represents the initial boundary condition of building $\mathbf{G^R}$ table.

$$G^R(1, n, p) = \begin{cases} -\infty & \text{if } MaxVF^{Budget}(1, n, p) = 0 \\ R_k(n, MaxVF^{Budget}(1, n, p)) & \text{otherwise} \end{cases} \tag{5.1}$$

Besides $\mathbf{G^R}$, the algorithm builds an auxiliary table called $\mathbf{G^P}$ to store the resulting power consumption from obtaining the maximum performance at each step. $\mathbf{G^P}$ has the same dimensions as $\mathbf{G^R}$ (see Figure 5.3). The initial boundary condition of building $\mathbf{G^P}$ is shown in Equation (5.2).

$$G^P(1, n, p) = \begin{cases} \infty & \text{if } MaxVF^{Budget}(1, n, p) = 0 \\ P_k(n, MaxVF^{Budget}(1, n, p)) & \text{otherwise} \end{cases} \tag{5.2}$$

To calculate $\mathbf{G^R}$ for $(k > 1)$ and $(k <= n)$, the following recursive function is used:

$$G^R(k,n,p) = \max_{\substack{k-1 \le n' < n \\ p_1 \le p' < p}} \left\{ \begin{cases} -\infty & \text{if } G^P\left(k-1,n',p'\right) + \\ & P_k\left(n-n', MaxVF^{Budget}(k,n-n',p-p')\right) > p \\ G^R\left(k-1,n',p'\right) + & \\ R_k\left(n-n', MaxVF^{Budget}(k,n-n',p-p')\right) & \text{otherwise} \end{cases} \right\} \quad (5.3)$$

Using Equation (5.3), the algorithm finds the maximum performance out of all the possible divisions of n and p between the application k and the previous $k-1$ applications. To represent these divisions, the parameters, n' and p' are used. n' will be set to the values from $k-1$ to n, because at least $k-1$ cores should be allocated to the previous $k-1$ applications. We assume that $K \le N$, and each application will get at least one core. p' will be set to the values from $p1$ to p, covering the whole range of the power budgets, because there is no constraint on the minimum power budget that each application should have. The resulting performance of all possible combinations of n' and p' will be calculated in order to choose the maximum one. As can be noticed from Equation (5.3), the previous computations of $G^R(k-1,n',p')$ are reused in order to reduce the number of required calculations. When the maximum performance out of these combinations is found, it is stored in $\mathbf{G^R}$, and the resulting power consumption from obtaining this performance is stored in $\mathbf{G^P}$, as follows:

$$G^P(k,n,p) = \quad G^P(k-1,n',p') + P_k(n-n', MaxVF^{Budget}(k,n-n',p-p')) \quad (5.4)$$

where n' and p' are the values that result in the maximum performance according to Equation (5.3). In case there is no feasible solution for the given power budget p or there are more applications than n cores, $G^R(k,n,p)$ is set to $-\infty$ and $G^P(k,n,p)$ is set to ∞.

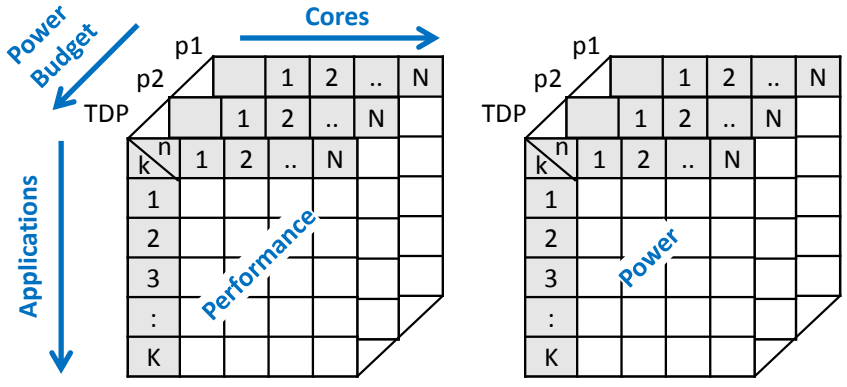


Figure 5.3: The employed dynamic programming tables, $\mathbf{G^R}$ and $\mathbf{G^P}$, that store the resulting performance and power values for different combinations.

At the same time of building $\mathbf{G^R}$ and $\mathbf{G^P}$, the values of the parameters $n-n'$, $p-p'$ and $MaxVF^{Budget}(k,n-n',p-p')$, which lead to the maximum performance that is stored in $G^R(k,n,p)$ at this step, are stored in tracking tables $TrackN$, $TrackP$, $TrackVF$,

Algorithm 1 Build $\mathbf{G^R}$ and $\mathbf{G^P}$ tables

Input: Application set with their power and performance tables P_k and R_k;
Output: Tables $\mathbf{G^R}, \mathbf{G^P}, TrackN, TrackP, TrackVF$;

1: **for** $k = 1$ **to** K **do**
2: **for** $n = 1$ **to** N **do**
3: **for** $p = p_1$ **to** p_{max} **do**
4: $vf = MaxVF^{Budget}(k, n, p)$;
5: **if** $vf = 0$ or $k > n$ **then**
6: $G^P(k, n, p) \leftarrow \infty$;
7: $G^R(k, n, p) \leftarrow -\infty$;
8: **else if** $vf > 0$ and $k \leq n$ **then**
9: **if** $k = 1$ **then**
10: Build $\mathbf{G^R}$ according to Equation (5.1);
11: Build $\mathbf{G^P}$ according to Equation (5.2);
12: **else if** $G^P(k-1, n', p') + P_k(n - n', MaxVF^{Budget}(k, n - n', p - p')) \leq p$ **then**
13: Build $\mathbf{G^R}$ according to Equation (5.3);
14: Build $\mathbf{G^R}$ according to Equation (5.4);
15: **end if**
16: Save the parameters $n - n', p - p', MaxVF^{Budget}(k, n - n', p - p')$ that result in the maximum performance at this step in the tracking tables $TrackN, TrackP, TrackVF$.
17: **end if**
18: **end for**
19: **end for**
20: **end for**
21: **return** Tables $\mathbf{G^R}, \mathbf{G^P}, TrackN, TrackP, TrackVF$;

respectively. The dimensions of these tracking tables are the same as the dimensions of $\mathbf{G^R}$ and $\mathbf{G^P}$ tables.

After building the tables, the final optimal solution, i.e., the maximum performance, can be found in $\mathbf{G^R}$ at the cell that holds the maximum value. Finding this cell is performed by searching through the cells whose entries hold (K, p_{max}), to find the one that holds the maximum value. When this value is extracted, the corresponding n will be the total number of active cores n_{max} that the applications can run on so that the maximum performance is obtained without exceeding the given power budget p_{max}, i.e., TDP. The optimal number of cores, the V/f level and the power quota for each application will be derived from the tracking tables, starting from the entry (K, n_{max}, p_{max}) (which holds the maximum performance). The values at this entry of the tracking tables $TrackN$, $TrackP$, and $TrackVF$ hold $n - n', p - p', MaxVF^{Budget}(k, n - n', p - p')$, respectively. These values represent the optimal settings (h_K, q_K, vf_K) of application K. For application $K - 1$, its optimal settings will be stored at the cell whose entry is $(K - 1, n', p')$, as can be observed from Equation (5.3). The values of n' and p' can be extracted by subtracting the found values of $n - n'$ and $p - p'$ from n and p, respectively. A similar process is repeated for further applications until reaching $k = 1$. The corresponding pseudo-code for building the tables $\mathbf{G^R}$ and $\mathbf{G^P}$ in a bottom-up manner is presented in Algorithm 1. The pseudo-code for extracting the solutions from the tracking tables is presented in Algorithm 2.

Algorithm 2 Extract optimal application settings h_k, q_k, vf_k, from dynamic programming tracking tables

Input: Tables $\mathbf{G^R}, \mathbf{G^P}, TrackN, TrackP, TrackVF$;
{Find the total number of active cores}
$n_{max} = 0$;
$R_{max} = 0$
for $n = N$ **to** 1 **do**
  if $G^R(K, N, p_{max}) > R_{max}$ **then**
    $R_{max} = G^R(K, N, p_{max})$;
    $n_{max} = n$;
  end if
end for
{Find the optimal settings of applications}
$n = n_{max}$;
$p = p_{max}$;
for $k = K$ **to** 1 **do**
  $h_k = Track\_N(k, n, p)$;
  $q_k = Track\_P(k, n, p)$;
  $vf_k = Track\_VH(k, n, p)$;
  {The entries for the optimal settings of the previous $k-1$ applications}
  $n = n - Track\_N(k, n, p)$;
  $p = p - Track\_P(k, n, p)$;
end for
return h_k, q_k, vf_k for all applications.

Hence, by the end of this step, the optimal settings of each application k are determined, namely, the number of threads (its active cores), h_k, the V/f level of its cores vf_k, and its power quota q_k.

Proof of optimality: We prove the optimality of the presented resource distribution algorithm by induction. Initially, $\forall n, p, G^R(1, n, p)$ is optimal because all the values of $R_1(n, p)$ and $P_1(n, p)$ are tested, and the maximum performance under the given power budget p is stored in $G^R(1, n, p)$. Now we need to prove that $G^R(k, n, p)$ is optimal assuming that $G^R(k-1, n, p)$ is optimal.
To achieve this, we first assume that Eq. (5.3) does not result in the maximum performance for $G^R(k, n, p)$. According to Eq. (5.3), the value of $G^R(k, n, p)$ is obtained by calculating the resulting performance for all possible combinations of dividing n and p between the previous $k-1$ applications and the current k application.
Thus, for each combination, different portions n', p' are given to the previous $k-1$ applications while $n'' = n - n'$ and $p'' = p - p'$ are given to the current application k. $\forall n, p$, the term $R_k(n'', maxVF^{Budget}(k, n'', p''))$ gives the performance of the application k at n'' cores and at V/f level equal to $maxVF^{Budget}(k, n'', p'')$. The function $maxVF^{Budget}(k, n'', p'')$ returns the maximum V/f level that leads to a power consumption less than p'' where the number of cores is n''. By definition the maximum V/f level (for n'' and p'') will give the maximum performance of application k (for n'' and p''). As a result, the term $R_k(n'', maxVF(n'', p''))$ will give the optimal solution for application k for a specific n'', p''. Consequently, $\exists n', p'$ that make the term $G^R(k-1, n', p')$ not optimal, and that, however, contradicts the assumption. As a result, by the mathematical induction hypothesis, we reach the conclusion that $G^R(k, n, p)$ is optimal $\forall n, p$.

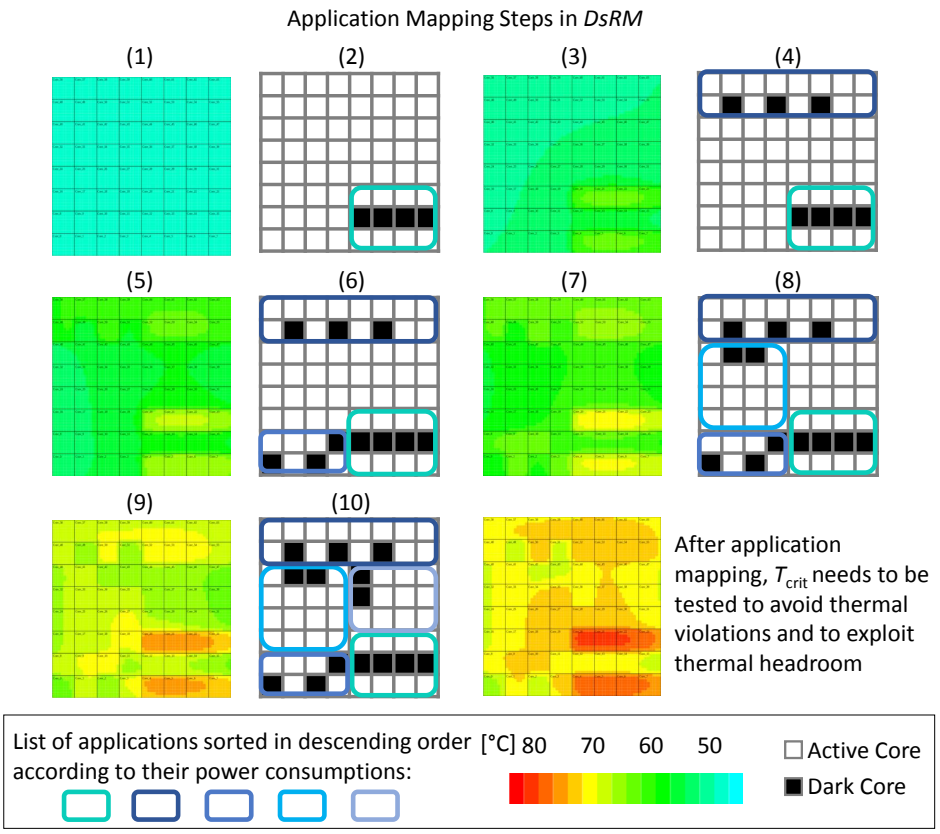


Figure 5.4: Example of the application mapping policy of *DsRM*, in which the applications are iteratively mapped to the chip. At each iteration, the core temperatures will be estimated, in order to map the application with the highest power consumption to the coldest region on the chip.

5.2.2 Thermal-aware Application Mapping

After determining the optimal number of h_k and vf_k of each application k in the previous step, *DsRM* employs a heuristic for application mapping that specifies which cores the each application will be mapped to. Since the previous dynamic programming algorithm considers the TDP constraint, some cores might be left dark and not allocated to any application. In this step, *DsRM* aims to exploit dark silicon to dissipate the generated heat on the active cores. Therefore, *DsRM* will first distribute the dark cores among the applications proportional to their power consumptions. Thus, the applications with higher power consumptions get more dark cores than the applications with lower power consumptions. The amount of dark cores specified for application k is denoted as d_k. After determining the number of dark cores of each application, the application mapping policy tries to specify close locations for the active and dark cores of each application, in order to allow the dark cores of each application to dissipate some of the generated heat on its active cores. Afterwards, the worst fit decreasing (WFD) algorithm is employed within the application mapping policy in order to map the applications with high power consumption to tiles with low average temperature. That also helps in reducing the resulting temperatures of the cores.

Thus, the application mapping policy of *DsRM* consists of the following steps: (1) Find the unmapped application k that has the highest power consumption. (2) Find the coldest tile ℓ, i.e., the average of its core temperatures is the lowest compared to the

other tiles. (3) If tile ℓ does not have h_k+d_k available cores, the policy reserves only the available cores of ℓ and searches for available cores in neighboring tiles until the whole h_k+d_k cores are reserved. Hence, a subset of cores are reserved for application k, namely, map_k. Afterwards, (4) the policy chooses the coldest core of map_k, then (5) maps one thread to it and re-estimates the core temperatures. The last two steps are repeated until all application threads are mapped. As a result of this policy, the active cores are distributed loosely through the region of the application cores map_k and the dark cores will be surrounding them. Moreover, the region of cores that will be allocated to the next application will be automatically far from the first application region, because the presented policy will re-estimate the core temperatures after mapping each application and then searches for the coldest tile. The entire steps are repeated until mapping all applications. Figure 5.4 illustrates an example of mapping a list of applications to the chip.

5.2.3 Thermal-Constrained Resource Adaptation

Although the presented mapping policy in the previous section aims at minimizing the potential temperature increase, it is still insufficient to avoid thermal violations. Therefore, a heuristic to adapt the resources considering the temperature constraint is proposed in order to avoid thermal violations and to exploit available thermal headroom. The flow chart of the proposed heuristic is shown in Figure 5.5. The first step of this heuristic is to compute the core temperatures using Equation (3.5) considering the application settings and mapping determined in the previous steps. In case any core of application k exceeds T_{crit}, the power quota q_k allocated to that application at step 1 (see Section 5.2.1) should be decreased. Contrarily, if there is available thermal headroom on some of the cores of an application, the allocated power quota of the application q_k will be increased to ultimately increase the performance. Note that *DsRM* does not increase the power quota of the application if the peak temperature of the application cores is slightly below the critical temperature, because this increase might lead to exceed T_{crit}, and therefore q_k must be reduced in the next iteration of the algorithm leading to unstable behavior. To prevent this, *DsRM* employs a parameter ε to determine if there is a *sufficient* thermal headroom to increase the power quota or not. This value needs to be empirically determined.

Algorithm 3 Find the settings of application k that maximize its performance under a power budget p

Input: Power and performance tables P_k and R_k (Section 3.2);
Output: h_k, vf_k;

```
1: R* = 0;   h* = 0;   vf* = 0;
2: for h = Min(H_k, h_k + d_k) to 1 do
3:    for vf = VF_Y to VF_1 do
4:       if P_k(h, vf) < p and R_k(h, vf) > R* then
5:          R* = R_k(h, vf);
6:          h* = h;
7:          vf* = vf;
8:       end if
9:    end for
10: end for
11: return h*, vf*;
```

After increasing or decreasing the power quota q_k, new application settings h_k, vf_k should be obtained, so that the performance of application k is maximized under the new power quota. Finding these new settings can be done by searching within the performance and power tables of that application, as shown in Algorithm 3. It is to be noted that the new h_k can be increased up to $\min\{H_k, h_k + d_k\}$. In other words, the application will adapt its own resources for maximizing its performance under the new power quota. In case this algorithm leads to increase h_k, that means dark cores of application k need to be activated to run threads. In this case, the coldest dark cores are activated. In contrast, if h_k is decreased, the hottest active cores of application k are darken.

After obtaining the new application settings, the steady-state temperatures of the cores will be re-estimated again. If there are still some cores that exceed T_{crit}, q_k will be decreased again. If q_k is decreased to a value, at which not even one thread at the lowest V/f level can run without exceeding T_{crit}, h_k is set to zero. In other words, all the cores of that application become dark. That means, running application k under the temperature constraint T_{crit} is not feasible. Figure 5.5 shows the flow chart of the resource adaptation heuristic.

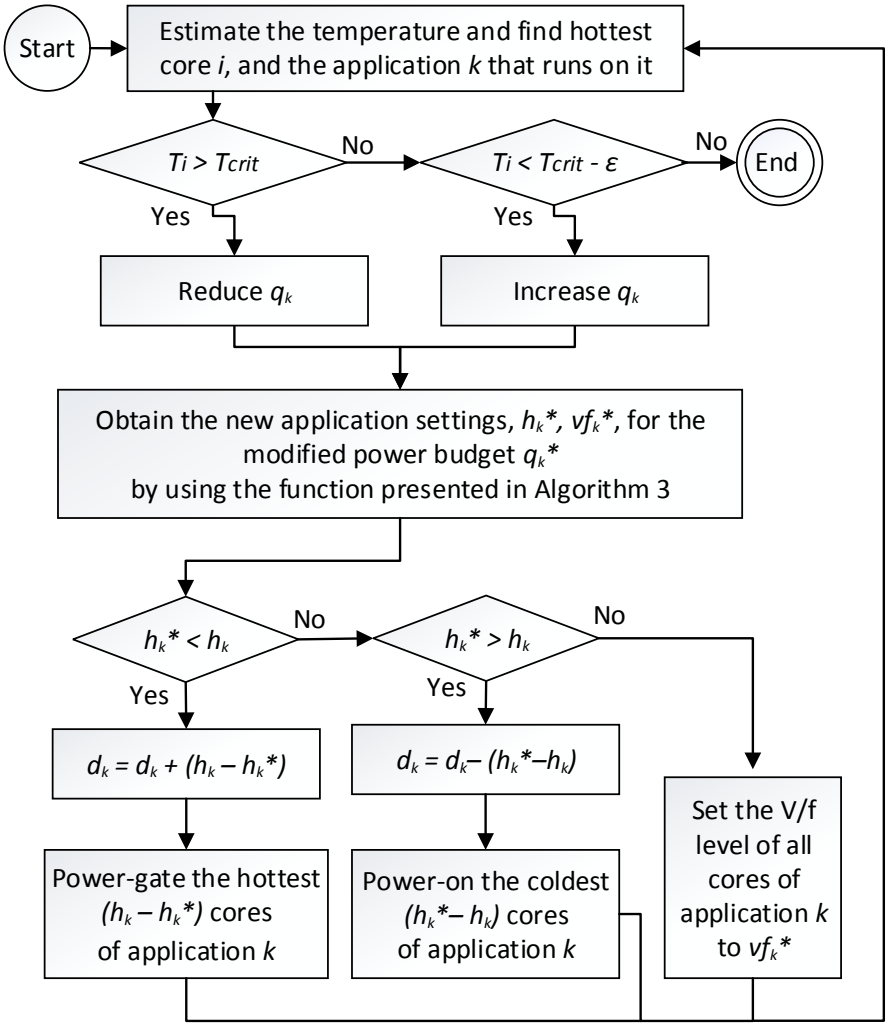


Figure 5.5: The flow chart of the third step of *DsRM*, i.e., resource adaptation.

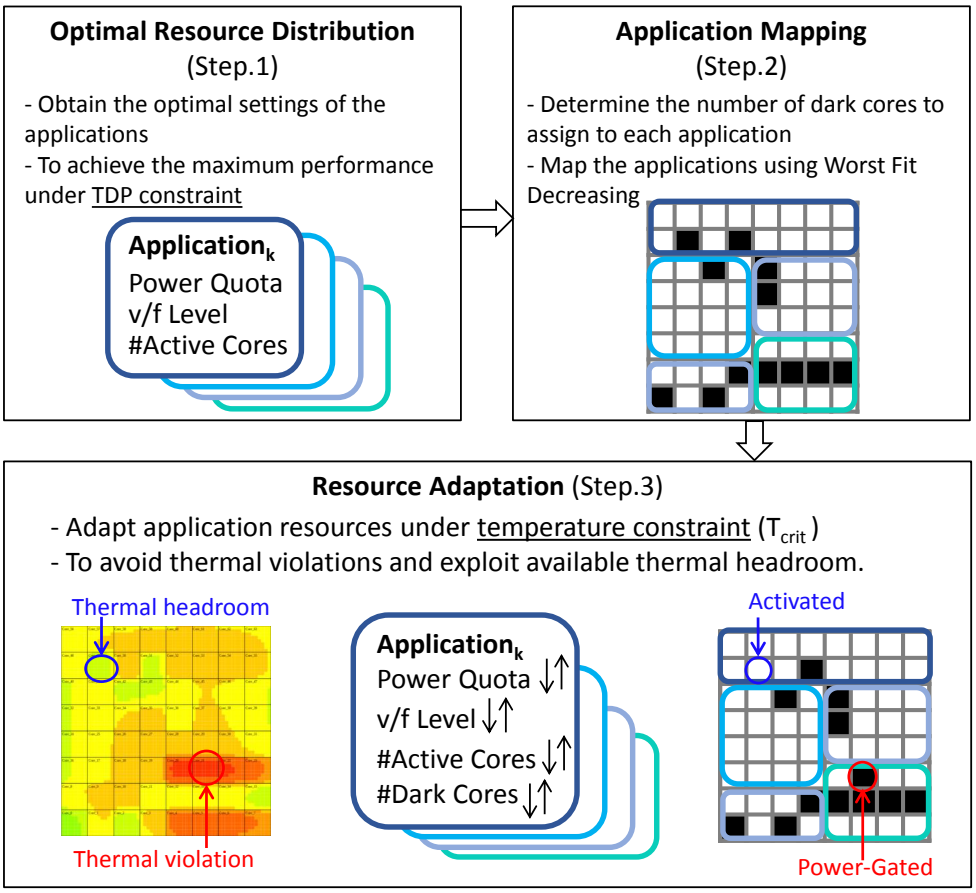


Figure 5.6: The steps of the presented dark silicon-aware resource management technique, *DsRM*.

In summary, applying the first and the second steps of *DsRM* results in a thermal-aware application mapping with optimal resource distribution under TDP constraint. It is referred to this combination as *TDPmap*. Nevertheless, having a TDP-constrained solution might still lead to thermal violations or to thermal headroom. Thus, the third step (resource adaptation) is applied and it might change the optimal resource distribution obtained by the first step, in order to consider the temperature constraint. Considering the temperature constraint, *DsRM* at the third step can adapt the resources in order to avoid thermal violations and to exploit available thermal headroom, while taking into account the performance and the power models of the applications, aiming at maximizing the overall system performance under the temperature constraint. An overview of the three steps of *DsRM* is shown in Figure 5.6.

5.3 Experimental Evaluations

5.3.1 Setup

To evaluate *DsRM*, the 64-core chip shown in Figure 4.3 is used. The critical temperature, T_{crit}, is set to 80 °C. The TDP is 220 W, which is near the practical numbers used in a real processor, i.e., Intel Xeon Phi [33], that consists of 64 cores.

Scenario	blackscholes	x264	bodytrack	ferret
Mix1	1	2	2	3
Mix2	-	3	1	4
Mix3	-	4	-	4
Mix4	4	-	-	4
Mix5	4	-	4	-

Table 5.1: The adopted scenarios of different applications. Each scenario consists of multiple instances of different applications. Each cell represents the number of instances of each application in the corresponding scenario.

For applications, "x264", "ferret", "blackscholes", and "bodytrack", from PARSEC benchmark suite [41] are utilized. Some of them have TLP more than ILP, like "blackscholes", and some have ILP more than TLP, like "ferret". Thus, these benchmarks are suitable examples to evaluate the efficiency of *DsRM*. As shown in Table Table 5.1, five scenarios (Mix1, Mix2, Mix3, Mix4, Mix5) of multiple instances of these applications are composed in order to provide various workloads with different ILP and TLP. Using gem5, each application is executed under a different number of threads $\{1, 2, \ldots, 8\}$ and different V/f levels. The complete tool flow of the experimental setup is illustrated in Figure 4.1.

It is assumed that *DsRM* will be implemented at design time, because deriving the optimal solution at step.1 requires a considerable time (2.36 s). However, runtime evaluation is performed as well to check the resulting transient temperatures, when the decisions of *DsRM* with regard to the application resources and their mappings are given as inputs to runtime simulation. In the following subsectionsvarious results of comparing *DsRM* with different state-of-the-art techniques are demonstrated.

5.3.2 Results

Before evaluating *DsRM* in general and comparing it to state-of-the-art techniques, it is important to evaluate its individual steps. The first step of *DsRM* results in the optimal solution, therefore, there is no need to evaluate it or compare it with other techniques. However, the second and the third steps of *DsRM* are heuristics. Therefore, they are evaluated in the following first two subsections.

(1) Evaluation of the presented mapping policy (the second step of *DsRM*): As aforementioned, at the second step of *DsRM*, i.e., application mapping, a heuristic based on WFD is employed and aims at reducing the temperature while mapping the applications. To evaluate this heuristic, it is compared to another application mapping policy, referred to as *ContiguousMap*, which tries to keep both the active cores, that applications are mapped to, and the unused (dark) cores contiguous. Several application mapping techniques, e.g., [53, 54], adopt the concept of contiguous mapping and map the applications close to each other in order to reduce the area fragmentation of the chip.

For fair comparison, the results of the first step of *DsRM*, i.e., the optimal application settings, are provided as inputs to *ContiguousMap*. As mentioned above, the version of *DsRM* that only executes the first and second steps is referred to as *TDPmap*. Hence, the comparison is conducted between *TDPmap* and *ContiguousMap*. Both of these

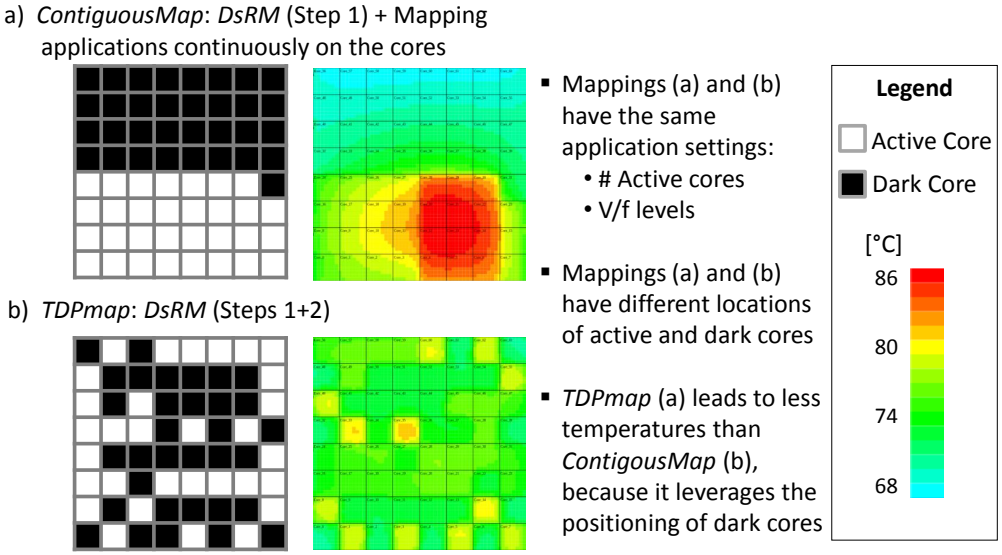


Figure 5.7: Comparison between the presented mapping policy, *TDPmap*, i.e., the second step of *DsRM*, and a contiguous mapping policy. Both policies utilize the same optimal application settings obtained by the first step of *DsRM*.

techniques consider the same application settings; the number of active cores (threads) of the applications, and the V/f levels of their cores, while each of them applies its own mapping policy. The resulting thermal maps of applying the *ContiguousMap* and *TDPmap* are compared as seen in Figure 5.7. As can be observed, the incurred temperatures on the cores when applying the mapping policy *TDPmap* are less than the incurred ones when applying *ContiguousMap*. Decreasing the core temperatures helps reduce the potential thermal violations and increase the available thermal headroom on the cores. That facilitates achieving the goal of the third step of *DsRM*, i.e, avoiding thermal violations and exploiting thermal headroom.

(2) Evaluation of the presented resource adaptation policy (the third step of *DsRM*):
The focus of this subsection is to evaluate the importance of the last step of *DsRM*, i.e., resource adaptation. For this purpose, several experiments for various workload scenarios are conducted to evaluate the resulting system performance when applying *DsRM* without resource adaptation, which means applying only the first and the second step of *DsRM*, i.e., *TDPmap*. Then, the resulting performance of *TDPmap* is compared with the resulting one when applying *DsRM* with all its three steps. Besides the system performance, the resulting amount of dark silicon is evaluated as well. Moreover, these experiments are conducted for two TDP values, i.e., 220 W and 180 W. The resulting performance of these experiments are shown in Figure 5.8.

The first observation of this figure is that the differences between the results of *DsRM* and *TDPmap* (w.r.t. the amount of dark silicon and the system performance) are more at the lower value of TDP. The reason is that setting TDP to a low value leads to less temperatures on the cores, and potentially larger thermal headroom exists on the cores. Hence, the last step of *DsRM* would have more possibilities to adapt the resources so that the available thermal headroom is exploited. The second observation is that the resource adaptation step either increases dark silicon amount or decreases it according to the TLP and ILP of the applications. For instance, at mix3 of scenario (b), the resource

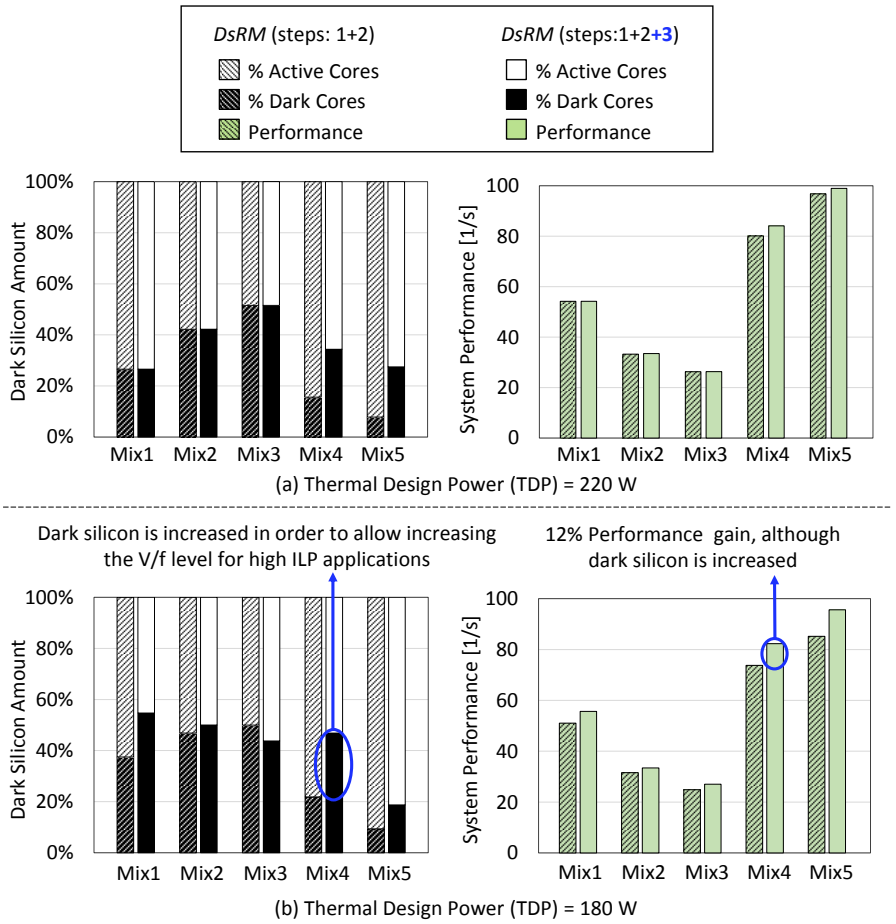


Figure 5.8: Evaluation of the presented resource adaptation, i.e., the third step of *DsRM*, by comparing *DsRM* with *TDPmap*, i.e., the combination of the first and the second steps of *DsRM*.

adaptation decreases the amount of dark silicon while the performance is increased. That means there was available thermal headroom at some cores, and it is exploited by increasing the number of threads of one or more applications from the running workload. On the other hand, at mix4 the amount of dark silicon is increased by the resource adaptation. Nevertheless, the performance is increased as well. Since the performance is increased, that means the adaptation in this case is also to exploit available thermal headroom, and not to avoid thermal violations. As a result, the resource adaptation in this case increases the performance by increasing the amount of dark cores of one or more applications from the workload and at the same time increasing the V/f levels of their active cores. In summary, the gain in the system performance resulting by applying the resource adaptation step is 12% compared to *TDPmap*.

(3) Comparison between *DsRM* and a state-of-the-art technique in performance maximization:
To achieve this comparison, the DVFS policy presented in [64] is implemented, and denoted as *MaxSpeed*. *MaxSpeed* aims at maximizing the system performance under the temperature constraint. Hence, *DsRM* and *MaxSpeed* share the same goal. However, the authors in *MaxSpeed* assume that the lateral heat flow between cores is neglected, because their adopted floorplan contains large caches around cores similar to a chess

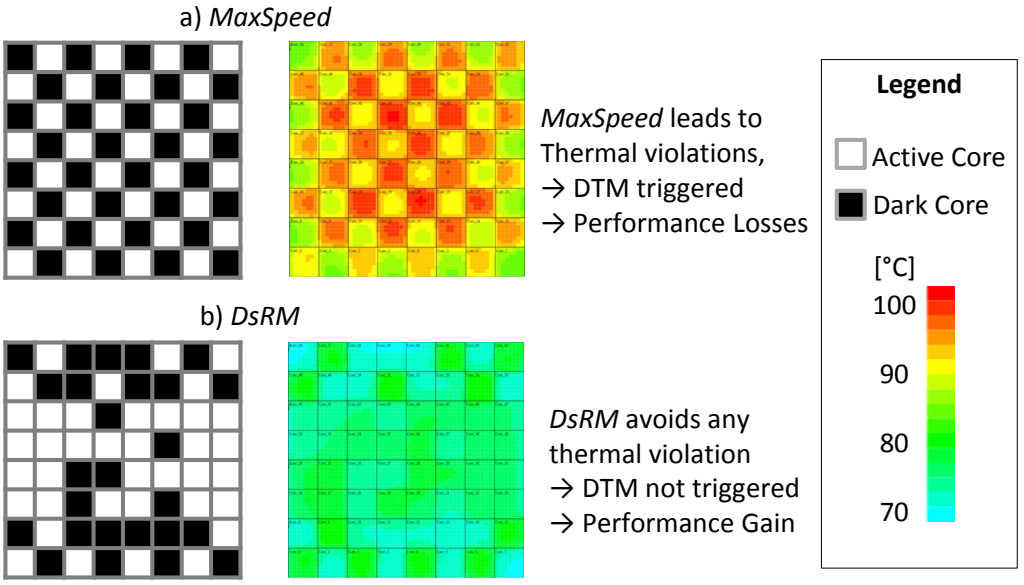


Figure 5.9: Comparison between *DsRM* and *MaxSpeed*, where the latter leads to thermal violations because it neglects the lateral heat transfer between the cores.

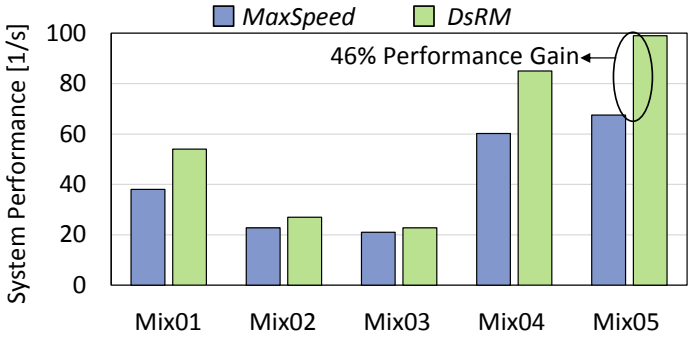


Figure 5.10: Comparison between the resulting system performance of *DsRM* and *MaxSpeed* [64]. The performance gain of *DsRM* is up to 46% compared to *MaxSpeed*.

board pattern. Therefore, to enable comparison with *MaxSpeed*, the chess board pattern is adopted for the active and dark cores, as shown in Figure 5.9, *MaxSpeed* is applied to the active cores. Furthermore, *MaxSpeed* does not target multi-threaded applications and thereby does not determine the number of cores to allocate to each application. Therefore, it is assumed that the active cores will be equally distributed to the coming applications in the case of *MaxSpeed*. Then, *MaxSpeed* determines the V/f levels of the active cores, so that the performance is maximized while the core temperatures are kept below T_{crit}, regarding their adopted thermal model that neglects the lateral heat transfer. *MaxSpeed* is applied to all scenarios (Mix1, ..., Mix5) of different applications. For example, in Figure 5.9, the resulting thermal maps are shown after applying *MaxSpeed* and *DsRM* on mix5. Note that the thermal maps are calculated using the adopted thermal model (Section 3.3), in which the lateral heat transfer between the cores are considered. As can be noticed from the shown thermal maps, *MaxSpeed* leads to a temperature increase more than 100 °C, because it neglects the heat contribution of the neighboring cores. This experiment demonstrates the importance of considering the heat transfer among the cores in the employed thermal model to estimate the temperatures more accurately.

In general, when the decisions of a resource management technique leads to thermal

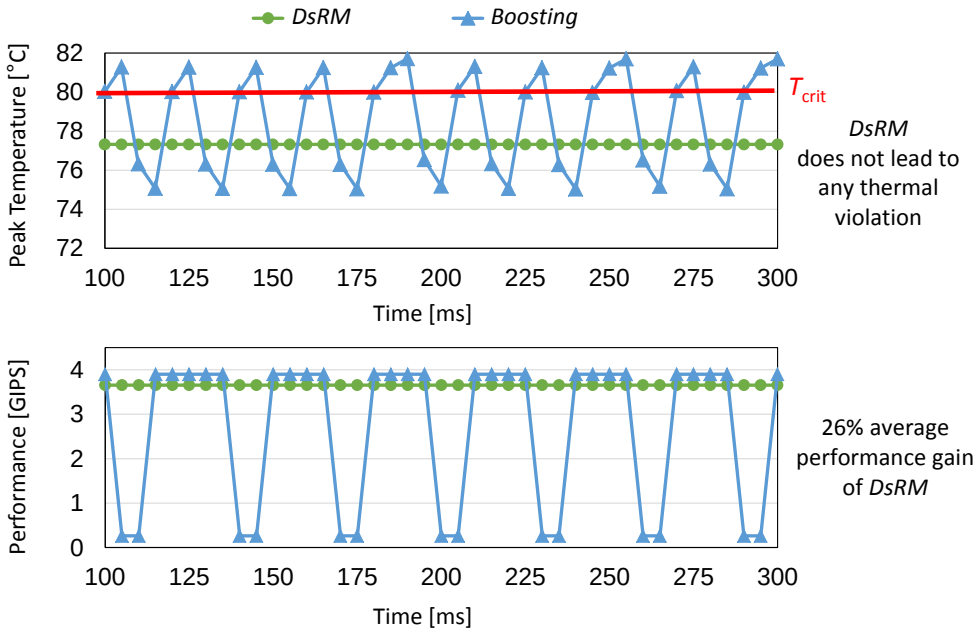


Figure 5.11: Runtime evaluation of *DsRM* and comparison with the boosting technique.

violations, DTM on the chip will be triggered to throttle down the cores and save the chip from burning out. Therefore, a simple thermal management technique is implemented in this experiment to check the core temperatures after the resource allocation decisions. If there is any thermal violation, the employed thermal management throttles down the V/f levels to keep the core temperatures below T_{crit}. Since *MaxSpeed* leads to thermal violations, the employed thermal management will throttling down the V/f levels, and consequently the system performance will be decreased. Contrarily, *DsRM* does not lead to any thermal violation. Figure 5.10 shows the resulting performance of *DsRM* compared to the resulting one of *MaxSpeed* after throttling down the V/f levels by the employed thermal management. As shown in the figure, the performance gain of *DsRM* compared to *MaxSpeed* reaches up to 46%. On average, the performance gain of *DsRM* is 34%.

(4) Comparison between *DsRM* and the state-of-the-art boosting technique: As noticed, the first comparison candidate results in severe thermal violations due to its employed thermal model that ignores the heat transfer among the cores. Therefore, we adopt another comparison candidate, which is a boosting technique that determines the V/f level of the cores based on measuring the temperatures of the cores at runtime within a control loop, similar to Intel Turbo Boost [67]. This boosting technique shares the same goal of *DsRM*, i.e., maximizing the performance under the temperature constraint. To achieve its goal, the boosting technique checks the peak temperature of the cores at each control period; if the peak temperature is below T_{crit}, the boosting technique increases the V/f levels of the cores by one step. Otherwise, if the peak temperature is higher than T_{crit}, the V/f levels are decreased by one step. This process will be repeated over the execution time. Hence, the boosting technique exploits any available thermal headroom at runtime.

However, the boosting technique does not determine the number of active and dark cores. Moreover, it does not determine the number of the cores that should be allocated to each

application. Therefore, we adopt the chess board pattern for the active and dark cores, and consider that the active cores will be equally distributed to the coming applications. After that, *Boosting* will boost the V/f levels of the cores at runtime. Therefore, a simulation loop is executed for 500 ms, and the boosting technique is activated to take its decision at each control period (5 ms). To compare *DsRM* with *Boosting*, *DsRM* is applied at design-time to determine the application mapping with their settings. Then, a simulation loop is executed for 500 ms, and at each control period (5 ms), the transient temperatures of the cores are checked. If *DsRM* leads to thermal violations, the DTM technique will be triggered to throttle down the V/f level in order to reduce the core temperatures.

Figure 5.11 shows first the resulting peak temperature over execution time when applying *DsRM* and *Boosting*. It can be observed, *DsRM* does not lead to any thermal violation and thereby DTM has not been triggered. However, *Boosting* leads to an increase in the peak temperature (due to the increase in the V/f levels) until exceeding T_{crit}. Once exceeded, *Boosting* throttles down the V/f levels. This process will be repeated over the simulation time. The second curve of Figure 5.11 shows the resulting IPS of the running workload in the case of *DsRM* and *Boosting*. As noticed, *Boosting* can lead to higher IPS at some points of the execution time, but on average, *DsRM* has 26% higher IPS than *Boosting*.

As a summary of the evaluation, *DsRM* efficiently manages the resources of the chip considering dark silicon. The performance gain that *DsRM* achieved is up to 46%, and an average of 34%. Moreover, the evaluation shows that maximizing the overall performance over time (*DsRM*) results in better performance gain on average, compared to maximizing the performance during short time intervals (*Boosting*).

5.4 Summary

Although dark silicon phenomenon can be considered as a problem, it can also be leveraged by resource management techniques in order to deliver more performance. In this chapter, we have discussed a resource management technique, called *DsRM*, that leverages the positioning of dark silicon to dissipate more heat from the active cores, which ultimately allows increasing the V/f levels of the active cores, and thereby improving the overall system performance.

Considering dark silicon, *DsRM* efficiently distributes the resources among applications by jointly deciding the number of the cores that should be allocated to each application and the V/f level of these cores, so that the overall system performance is maximized. The temperature constraint of the chip is considered as an essential factor by the decision making process of *DsRM*. As the conducted evaluation shows, the decisions of *DsRM* do not lead to any thermal violations and it exploits available thermal headroom in order to maximize the performance.

Chapter 6

Power Density-Aware Resource Management

Dark silicon is the direct result of the continuous power density increments along with technology scaling and the corresponding thermal problems. Heterogeneous architectures that include different types of cores and accelerators have attracted a major focus in recent years, as they provide many possibilities for performance and power efficiency [123], especially in the dark silicon era. Given that the voltage and frequency requirements as well as capabilities vary among cores and accelerators, an efficient method to design such systems is by considering multiple tiles, in which elements inside a tile share the same voltage and frequency, but different tiles can execute at different voltage and frequencies at any given time point. An example of such an architecture is the Exynos 5 Octa (5422) chip [66] with ARM's "big.LITTLE" architecture composed of three tiles: a quad-core Cortex-A7 (ARM-A7), a quad-core Cortex-A15 (ARM-A15), and a Mali T-628 GPU with six shader cores.

Solving the problem of performance optimization under a temperature constraint for such heterogeneous architecture is much more complicated than solving it for homogenous multicores as presented in the previous chapter. The reason is that in heterogeneous multicores power and performance characteristics of applications significantly differ from a core type to another. Moreover, thermal characteristics of heterogeneous multicores are also different. Even if two heterogeneous cores have the same power value, the resulting temperatures, the two exhibit, might be different if they have different areas. Thus, these different characteristics need to be taken into account when optimizing the performance of heterogeneous multicore chips under a temperature constraint.

Therefore, in this chapter we introduce power density as a novel system-level constraint that serves as an abstraction from thermal issues. However, the case study presented in Section 1.2.1 shows how a power density constraint is not enough to avoid thermal violation, because the distribution of active and dark cores has a significant impact on the generated temperature. To address this challenge, we derive the power density constraint based on the thermal model of the chip so that it guarantees avoiding thermal violations. Also it considers core heterogeneity within a chip. Then, we present a power-density-aware resource management, referred to as *PdRM*, that manages the resources of heterogeneous tiled multicores, aiming at maximizing the overall system performance without violating the critical temperature. In particular, *PdRM* assigns applications to

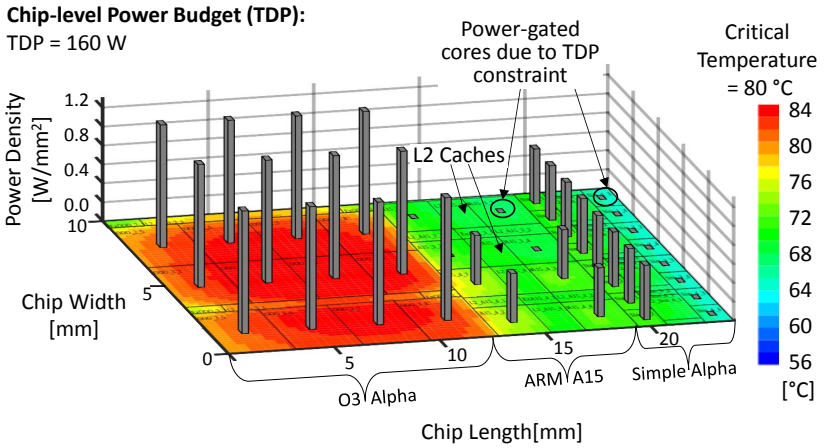


Figure 6.1: The power densities and temperatures of all cores when a chip-level power budget is used. The resulting thermal profile contains several thermal violations that lead to excessive DTM triggers, which in turn cause performance losses.

tiles, maps them to cores, and determines both the parallelism level of the applications and the V/f levels of the tiles, so that the performance is maximized under the derived power density constraint. After application mapping, the power density constraint can be adapted at run time to exploit any available thermal headroom on the cores and to react to workload changes.

In the following section, a motivational example is presented to show the potential benefits of such a power density constraint against a standard chip-level power constraint. Section 6.3 presents details about our presented technique. The evaluation results of our technique are presented in Section 6.5.

6.1 Motivational Example

In this example, we analyze the power densities and the thermal profile of a 36-core heterogeneous tiled multicore under three different constraints. The adopted chip consists of three tiles: a tile with 12 O3 Alpha cores, a tile with 16 simple Alpha cores, and a tile with 8 ARM-A15 cores. The nominal frequencies are 4 GHz for both Alpha tiles and 2 GHz for the ARM-A15 tile (further details of the setup are explained in Section 6.4). For applications, 18 instances of an H.264 video encoder from the PARSEC benchmark suite [41] are considered, where each application instance runs two parallel dependent threads. For simplicity of presentation, a simple mapping policy is considered for all three scenarios. Specifically, in order to distribute the applications throughout the tiles as much as possible, we assign them to tiles in a round robin manner and then map them to the cores in a tile consecutively.

In the first scenario shown in Figure 6.1, a chip-level power constraint (namely, TDP) is used. The applications are mapped to cores (running at nominal frequency) until the total power consumption reaches the predefined constraint, and the rest of the cores stay dark. For this constraint in this example (TDP = 160 W), only 12 applications of the 18 are mapped, resulting in 24 active cores and 12 dark ones. Figure 6.1 shows the resulting power densities and the temperature profile of the cores. It can be observed

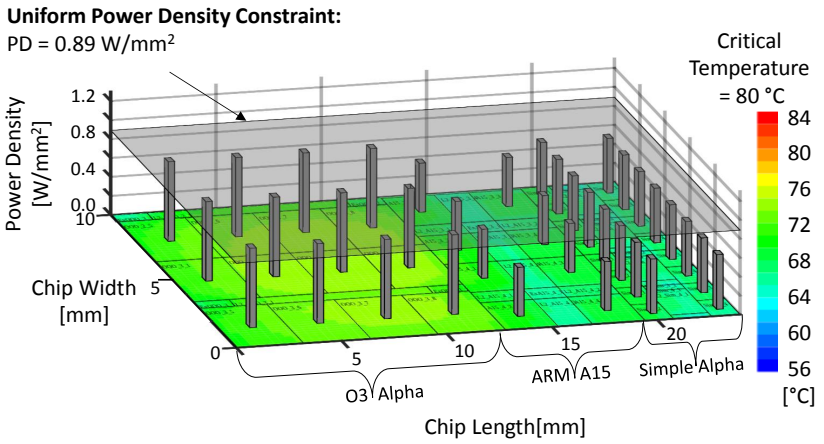


Figure 6.2: The power densities and temperatures of all cores when a uniform power density constraint is used. The resulting thermal profile does not have any thermal violation, but large thermal headroom exists.

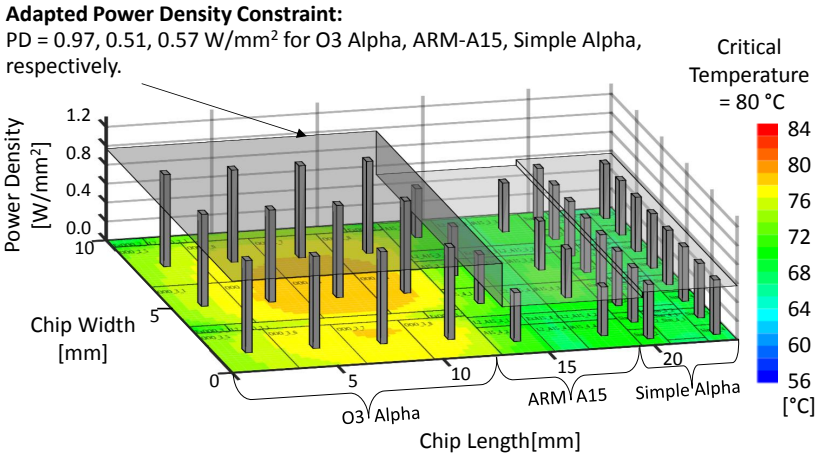


Figure 6.3: The power densities and temperatures of all cores when a selective or adapted power density constraint is used. The resulting thermal profile does not have any thermal violation, thermal headroom is exploited and thus improving the performance.

that adopting such a chip-level power constraint results in a big variance of the resulting power densities, which in turn leads to thermal violations on some cores and considerable thermal headroom on others. These thermal violations will trigger DTM leading to degrading the overall performance. The unbalanced thermal profile results due to the heterogeneity of the cores, namely the big variance of the power consumptions between the cores. Therefore, such a chip-level constraint might be efficient on homogenous cores in some scenarios, but certainly inefficient on heterogeneous ones.

In the scenario shown in Figure 6.2, all cores share a uniform power density constraint which cannot be exceeded. Using a power density constraint rather than a power constraint is more suitable for heterogeneous multicores, because the bigger cores will get more power budget than the smaller ones. That will help in improving the thermal profile. In particular, it reduces thermal violations and thermal headroom, and thereby the performance will improve. The power density constraint can be empirically computed

based on the geometric and thermal properties of the cores, so that no thermal violations incur when the power density on all cores is equal to or less than the power density constraint. After mapping the applications to all cores according to the considered mapping policy, DVFS is applied at the tile level in order to satisfy the power density constraint. As a result, the frequency of the O3 Alpha tile is scaled down to 3.0 GHz (gray cores), while the other tiles execute at nominal frequencies, and thus all 18 applications are mapped. Given that this power density constraint is empirically extracted to be thermally safe, we notice from Figure 6.2 that no thermal violation occurs and DTM is not triggered. However, it can be observed that the power density on simple Alpha cores and ARM-A15 cores is less than the uniform power density constraint. That means, these cores cannot fully utilize their allowed power budget. On the other hand, there are some cores that can consume more power if their power budgets are increased.

Therefore, in order to maximize the utilization of the cores, we re-adapt the power density constraint (Figure 6.3) by decreasing it for low power cores/applications and increasing it for high power cores/applications. Hence, this adaptation considers the power characteristics of the executed applications on the heterogeneous cores and thereby more thermal headroom is exploited, resulting in higher performance. Furthermore, this adaptation can also be applied when the workload changes at runtime, for example, when an application finishes its execution or a new application arrives.

From this motivational example it is clear that chip-level power constraint is not well suited for heterogeneous multicores, and adopting a power density constraint is a simple and effective approach for avoiding any thermal violation. Moreover, to be able to exploit the available thermal headroom, it is important to provide a mechanism that adapts this constraint by considering the power characteristics of the executed applications and the workload changes at runtime.

6.2 Problem Definition

Given a set of K weighted applications with weights $W = \{w_1, w_2, \ldots, w_K\}$, the objective is to distribute the chip resources, i.e., power and cores, among the applications, so that the overall weighted system performance[1] is maximized under the predefined critical temperature of the chip (T_{crit}). More specifically, we need to find the application-to-tile assignment, the V/f levels of each tile, the degree of parallelism of each mapped application, and the mapping of application threads to cores inside a tile.

The purpose of using the weights W is to balance the impact of the execution time of the applications on the overall system performance and to assign priorities to applications. For example, if the weights would not be used, an application with long execution time, but with high priority, would receive a small number of resources compared to some applications with short execution time but low priority. Besides their importance for balancing the impact of execution time on the performance, the weights can be additionally used to assign different priorities to the applications according to the desire of the user.

Mathematically, this problem can be expressed as finding the settings for each application $k = 1, 2, \ldots, K$ (i.e., its number of threads (h_k), the tile ℓ and its corresponding

[1] The performance of an application is defined as the inverse function of its execution time as shown in Equation (3.1).

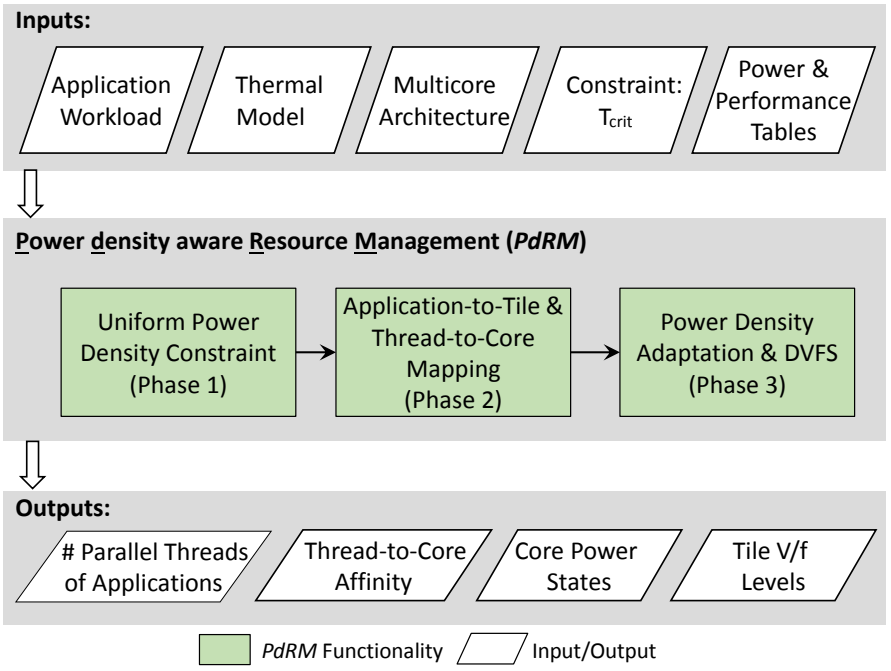


Figure 6.4: Overview of the proposed technique, *PdRM*.

core type τ, as well as its vf_k), the application assignment matrix $\mathbf{A}$ and the mapping matrix $\mathbf{M}$, in order to:

$$\text{Maximize} \sum_{\ell=1}^{L} \sum_{k=1}^{K} a_{\ell,k} \cdot w_k \cdot R_k\left(\tau, h_k, vf_k\right) \tag{6.1}$$

$$\begin{aligned} \text{subject to:} \quad & \textstyle\sum_{k=1}^{K} a_{\ell,k} \leq N_\ell && \text{for all } \ell = 1, 2, \ldots, L \\ & \textstyle\sum_{\ell=1}^{L} a_{\ell,k} \leq 1 && \text{for all } k = 1, 2, \ldots, K \\ & T_i \leq T_{crit} && \text{for all } i = 1, 2, \ldots, N \end{aligned}$$

There, τ indicates the corresponding core type of the tile ℓ, the application performance $R_k(\tau, h_k, vf_k)$ is estimated using Equation (3.1). T_i is the steady-state temperature of core i estimated using Equation (3.6). This problem is equivalent to *Generalized Assignment Problem (GAP)* as illustrated in [124]. GAP is known to be an $\mathcal{NP}$-hard problem in the strong sense [125]. Therefore, finding an optimal solution will result in an exponential time complexity.

Therefore and given that our technique is assumed to work at runtime, we need to find a simple, lightweight, yet efficient scheme to achieve the required goal in polynomial-time. For this reason, we decompose the problem into the three sub-problems: (1) Deriving a uniform power density constraint for all cores as an abstraction from thermal issues, while the heat transfer among cores is implicitly considered. (2) Under the derived power density constraint, we need to find the application-to-tile assignment and the corresponding core mapping so that the overall weighted performance is optimized. (3) In order to exploit any available headroom, the power density constraint needs to be re-adapted to consider the resulting assignment and power characteristics of the executed applications on the heterogeneous cores.

6.3 Power Density-Aware Resource Management

This section presents our power density-aware resource management, *PdRM*, which consists of three phases, as seen in Figure 6.4, detailed in the following subsections.

6.3.1 Uniform Power Density Constraint

A uniform power density constraint is derived so that it guarantees avoiding thermal violations and considers core heterogeneity within a chip and heat transfer among cores. It is considered that there might be some thermal nodes whose power consumption values are given, and thus there is no need to optimize the power density for such nodes. Hence, a binary vector $\mathbf{X} = [x_j]_{Z\times 1}$ is defined such that $x_j = 0$ implies that the power consumption on node j is given, while $x_j = 1$ implies that we want to compute our power density constraint on node j. In this way, Equation (8.4) is arranged as follows:

$$T_i = \sum_{j=1}^{Z} b_{i,j} \cdot \text{area}_j \cdot Pd_j \cdot x_j + \sum_{j=1}^{Z} b_{i,j} \cdot p_j \cdot (1 - x_j) + T_{\text{amb}} \sum_{j=1}^{Z} b_{i,j} \cdot g_j \qquad (6.2)$$

There, area_j is the area of node j, and $Pd_j = \frac{p_j}{\text{area}_j}$ represents the power density of node j.

From Equation (6.2), T_i can be set to T_{crit}. Then, a uniform power density value, defined as Pd^{uniform}, can be derived such that the temperature on node i is equal to T_{crit} when all cores of interest, i.e., all cores for which $x_j = 1$, have the uniform power density Pd^{uniform}. Specifically, Pd^{uniform} is computed as follows:

$$Pd^{\text{uniform}} = \frac{T_{\text{crit}} - \sum_{j=1}^{Z} b_{i,j} \cdot p_j \cdot (1 - x_j) - T_{\text{amb}} \sum_{j=1}^{Z} b_{i,j} \cdot g_j}{\sum_{j=1}^{Z} b_{i,j} \cdot \text{area}_j \cdot x_j} \qquad (6.3)$$

In order to compute a safe power density constraint and given that different nodes will have different temperatures even when all active cores have the same power density, it is necessary to compute Pd^{uniform} for all nodes $i = 1, 2, \ldots, Z$ and use the minimum computed power density as a thermally-safe power density constraint, referred to as Pd^{safe}. Specifically, by extending Equation (4 in [18]) to focus on power density rather than on power, Pd^{safe} can be computed according to Equation (6.4).

$$Pd^{\text{safe}} = \min_{1 \le i \le Z} \left\{ \frac{T_{\text{crit}} - \sum\limits_{j=1}^{Z} b_{i,j} \cdot p_j \cdot (1 - x_j) - T_{\text{amb}} \sum\limits_{j=1}^{Z} b_{i,j} \cdot g_j}{\sum\limits_{j=1}^{Z} b_{i,j} \cdot \text{area}_j \cdot x_j} \right\} \qquad (6.4)$$

Finally, although Equation (6.4) is general for any vector $\mathbf{X}$, Pd^{safe} is computed in Phase 1 of *PdRM* by assuming that $x_j = 1$ for all cores on the chip, only considering given power consumption for the thermal nodes that represent caches, memory controllers, etc. For example, the uniform power density constraint shown in Figure 6.2 can be computed through Equation (6.4) by assuming that $x_j = 1$ for all cores.

6.3.2 Application Mapping under Power Density Constraint

The derived power density constraint in the previous section enables making the mapping decision abstracted from thermal issues. Moreover, as the derived power density constraint Pd^{safe} is given for each core, the complexity of the mapping problem is reduced. The reason is that it is possible to evaluate if a mapping decision satisfies Pd^{safe} without the need of calculating the resulting temperature on other cores or dealing with heat transfer among cores, as these problems are already handled by executing below Pd^{safe}. In this section, the focus is on the second problem of assigning applications to tiles and threads to cores inside a tile. To solve this problem, we propose a heuristic, *MapAlgo*. Its main steps are explained in Figure 6.5, while its pseudo-code is listed in Algorithm 4.

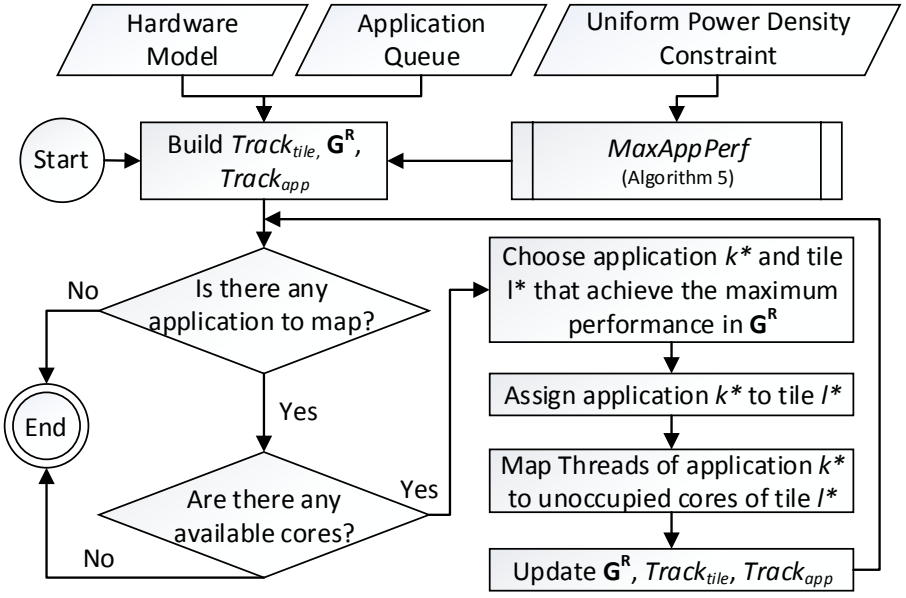


Figure 6.5: Application Mapping Heuristic, *MapAlgo*.

Algorithm *MapAlgo* relies on a table denoted as $\mathbf{G^R}$ whose entries correspond to all unique combinations (ℓ, k) for all $\ell = 1, 2, \ldots, L$ and for all $k = 1, 2, \ldots, K$. Each cell in row ℓ and column k of this table, i.e., $\mathbf{G^R}(\ell, k)$, will contain the maximum weighted performance of application k under the power density constraint Pd^{safe}, when the application is assigned to tile ℓ, by taking into account the current available number of cores in tile ℓ, denoted as A_ℓ, and the selected execution frequency of the tile vf_ℓ^{tile}. These values, i.e., A_ℓ and vf_ℓ^{tile}, are stored in an auxiliary table denoted as $Track_{tile}$, and will be updated every time an application is assigned to a tile.

The first step of *MapAlgo* is to build table $\mathbf{G^R}$. To achieve that, we first define an auxiliary function $MaxAppPerf\left(\ell, k, A_\ell, vf_\ell^{\text{tile}}, Pd^{\text{safe}}\right)$. For a given tile ℓ of a core type τ and application k, function $MaxAppPerf$ performs a search in table $R_k(\tau, vf, h)$, starting from $h = A_\ell$ and $vf = vf_\ell^{\text{tile}}$. At each step, the resulting power density is computed from the corresponding cell of the same entries in table $P_k(\tau, vf, h)$ and compared to the given power density constraint Pd^{safe}. If the power density constraint is not satisfied, the algorithm moves to the next cell in the table R_k, whose entry is $(\tau, vf-1, h)$ if $R_k(\tau, vf-1, h) > R_k(\tau, vf, h-1)$, or $(\tau, vf, h-1)$ otherwise.

Algorithm 4 Applications to Tiles and Threads to Cores Mapping

Input: Power density constraint Pd^{safe} and the set of applications with their power and execution time tables P_k and R_k;
Output: Matrices $\mathbf{A}$ and $\mathbf{M}$, and the V/f levels of the tiles;
{Initialization of tables}
1: **for all** ℓ in $1, 2, \ldots, L$ **do**
2: Initialize $Track_{tile}(\ell)$ with values $(N_\ell, VF_{\ell,y})$;
3: **for all** k in $1, 2, \ldots, K$ **do**
{Calculate $G^R(\ell, k)$ and $Track_{app}(\ell, k) \leftarrow$}
4: $MaxAppPerf\left(\ell, k, A_\ell, vf_\ell^{\text{tile}}, Pd^{\text{safe}}\right)$;
5: **end for**
6: **end for**
{Assigning applications to tiles}
7: $U \leftarrow \{1, 2, \ldots, K\}$; {Assign applications to empty set}
8: **while** $U \neq \emptyset$ **and** $\sum_{i=1}^{N} \sum_{j=1}^{K} m_{i,j} < N$ **do**
9: $\ell^\star, k^\star \leftarrow$ Row & column in $\mathbf{G^R}$ with highest performance;
10: $a_{\ell^\star,k^\star} \leftarrow 1$; {Application-to-tile assignment}
11: Assign the settings in $Track_{app}(\ell^\star, k^\star)$ to application $k^\star$;
12: Delete column $k^\star$ from table $\mathbf{G^R}$;
13: Update table $Track_{tile}(\ell^\star)$ by subtracting $h_{k^\star}$ from the set of available cores A_ℓ and by setting the V/f level of the tile vf_ℓ^{tile} according to $Track_{app}(\ell^\star, k^\star)$;
14: Update the values of row $\ell^\star$ in table $\mathbf{G^R}$ by considering the new A_ℓ and vf_ℓ^{tile} values;
15: $U \setminus \{k\}$; {Removing application k from set U}
16: Mapping application to the unoccupied cores of tile $\ell^\star$;
17: **end while**
18: **return** Matrices $\mathbf{A}$ and $\mathbf{M}$, and vf_ℓ^{tile} for all $\ell = 1, 2, \ldots L$;

In this way, function $MaxAppPerf$ greedily finds the maximum performance of application k when assigned to tile ℓ under the given inputs of the tile, i.e. A_ℓ and vf_ℓ^{tile}, and the given power density constraint Pd^{safe}. Once the maximum performance under Pd^{safe} is found, it is stored in $\mathbf{G^R}$, and the application settings that achieve it are stored in an auxiliary table called $Track_{app}$. This table has the same dimensions as $\mathbf{G^R}$, i.e., KxL, and it is simultaneously built with $\mathbf{G^R}$ to store the application settings that achieve the maximum performance stored in $\mathbf{G^R}$. Each cell in row ℓ and column k of this table, i.e., $Track_{app}(\ell, k)$, contains a tuple $\langle h, vf \rangle$, which represents the settings of application k (number of threads and V/f level) that achieve the maximum weighted performance of this application under Pd^{safe}, when it is assigned to tile ℓ. When first building table $\mathbf{G^R}$, function $MaxAppPerf$ is called by considering $A_\ell = N_\ell$ and $vf_\ell^{\text{tile}} = VF_{\ell,y}$.

Now the table $\mathbf{G^R}$ has been built, the next step of *MapAlgo* is to perform a search through $\mathbf{G^R}$ in order to find the maximum weighted performance in the table. Once found, we assign application $k^\star$ to tile $\ell^\star$, such that $a_{\ell^\star,k^\star} = 1$, where $k^\star$ and $\ell^\star$ are respectively the column and row index of the found maximum performance in table $\mathbf{G^R}$. The settings of application $k^\star$ are taken from the stored values in table $Track_{app}$ at cell $(\ell^\star, k^\star)$. Afterwards, column $k^\star$ is deleted from table $\mathbf{G^R}$, while the settings of tile $\ell^\star$ in table $Track_{tile}$ are updated so that the number of available cores A_ℓ is reduced by $h_k^\star$, and the V/f level of the tile vf_ℓ^{tile} is set according to $Track_{app}(\ell^\star, k^\star)$. Furthermore, it is necessary to update the values of row $\ell^\star$ in tables $Track_{app}(\ell, k)$ and $\mathbf{G^R}$ (using function $MaxAppPerf$ again) for all unmapped applications by considering the new A_ℓ and vf_ℓ^{tile} values. This is necessary because the tile now has less available cores and might be set to execute at lower V/f levels, and therefore the original values of row $\ell^\star$ in tables $Track_{app}(\ell, k)$ and $\mathbf{G^R}$ are no longer valid.

Algorithm 5 $MaxAppPerf$: Calculate the maximum performance for an application k within tile ℓ under Pd^{safe}

Input: Power density constraint Pd^{safe}, $\ell, k, A_\ell, vf_\ell^{\text{tile}}$;
Output: $G^R(\ell,k), Track_{app}(\ell,k)$;
1: $h = A_\ell$;
2: $vf = vf_\ell^{\text{tile}}$;
3: **while** $h > 0$ **and** $vf > 0$ **do**
4: $PowerDensity = P_k(\tau, vf, h)/h/coreArea_\tau$;
5: **if** $PowerDensity > Pd^{\text{safe}}$ **then**
6: **if** $R_k(\tau, h-1, vf) > R_k(\tau, h, vf-1)$ **then**
7: $h = h - 1$;{Decrease the number of threads}
8: **else**
9: $vf = vf - 1$;{Decrease the V/f level}
10: **end if**
11: **else**
12: $Track_{app}(\ell,k).h = h$;
13: $Track_{app}(\ell,k).vf = vf$;
14: $G^R(\ell,k) = w_k \dot{R}_k(\tau, h, vf)$;
15: Break;
16: **end if**
17: **end while**

After assigning an application to a tile, the application threads are greedily mapped to the available cores in that tile. The steps for assigning and mapping an application are repeated until all applications are mapped or all cores are assigned.

Algorithm 6 Runtime Power Density Adaptation

Input: Original power density constraint Pd^{safe} when $x_j = 1$ for all cores;
Output: Adapted V/f levels;
1: $x_j \leftarrow 1$ for all cores;
2: **repeat**
3: **for all** j in $1, 2, \ldots, Z$ **do**
4: $\ell \leftarrow$ tile number of core j;
5: **if** $x_j = 1$ **and** $vf_j^{\text{core}} = VF_{\ell,y}$ **then**
6: $p_j \leftarrow$ Power of the application mapped to core j when executing at $VF_{\ell,y}$;
7: $x_j \leftarrow 0$;
8: **end if**
9: **end for**
10: Compute new Pd^{safe} according to Equation (6.4) for the new x_j and p_j values;
11: Increase the V/f levels of the tiles holding cores with $x_j = 1$, while satisfying the new Pd^{safe}.
12: **until** $\sum_{j=1}^{Z} x_j = 0$ **or** no core underutilizes the power density budget
13: **return** New V/f levels;

6.3.3 Runtime Power Density Adaptation

The uniform power density constraint derived in Section 6.3.1 is a very efficient approach for heterogeneous multicores, since it is based on geometric and thermal properties of the different cores. However, it ignores the power characteristics of the executed applications on the different types of cores and accelerators, as seen in Figure 6.2. Therefore, this section presents a runtime adaptation scheme that exploits available thermal headroom

and also reacts to workload changes at runtime. A pseudo-code for this scheme is presented in Algorithm 6.

For example, when considering Figure 6.2, it can be seen that the ARM-A15 tile is producing much less power density than that assigned by the constraint from Equation (6.4). This could happen for two reasons: either the frequency steps for this core type are far from each other so that executing the tile at the next available frequency violates the power density constraint, or the tile is already executing at the maximum frequency and the power density still does not reach the constraint. The latter case means that the power density budget assigned to the ARM-A15 tile is underutilized and thus we can decrease it in order to increase the power density budget assigned to the other tiles. This can be done very simply through Equation (6.4) by setting x_j of these cores to zero and their power consumptions to the resulting power values for the given number of threads and highest V/f levels (Lines 6, 7 of Algorithm 6).

In this way, a new value of Pd^{safe} is derived and applied to the rest of the cores, whose x_j holds the value 1. Hence, the V/f levels are adapted by using DVFS according to this new power density constraint. The process is then repeated until $x_j = 0$ for all $j = 1, 2, \ldots, Z$, or there are no more cores that under-utilize their assigned power budget constraint (Line 12 of Algorithm. 6). For example, the adaptation shown in Figure 6.3 can be computed through Algorithm 6. Furthermore, this runtime adaptation is also triggered when there is a workload change at runtime, e.g., when an application finishes or a new application arrives.

Scenario	#PARSEC applications	#TCPA applications	Scenario	#PARSEC applications	#TCPA applications
Mix01	10	10	**Mix07**	12	35
Mix02	5	50	**Mix08**	3	20
Mix03	10	5	**Mix09**	6	6
Mix04	8	25	**Mix10**	2	40
Mix05	12	3	**Mix11**	11	7
Mix06	8	16	**Mix12**	10	30

Table 6.1: Mixed application scenarios.

TCPA applications	Weights	PARSEC applications	Weights
FIR	1.14E-04	x264	1.55E+00
Harris Corner	7.71E-02	bodytrack	4.68E-01
Edge Detection	5.69E-02	blackscholes	2.20E-01
Matrix Multiplication	6.65E-05	swaptions	1.01E-01
SAD	1.11E-02	ferret	1
Optical Flow	3.43E-02	-	-

Table 6.2: The weights of applications in *WS1*.

6.4 Experimental Setup

The employed heterogeneous multicore architecture for evaluating the presented technique *PdRM* is explained in Section 4.2.2 and shown in Figure 4.6. Several multi-threaded applications from the PARSEC benchmark suite [41] are considered, which are "x264", "bodytrack", "blackscholes", "swaptions" and "ferret". These applications can

TCPA applications	Weights	PARSEC applications	Weights
FIR	5.69E-05	x264	1.55E+00
Harris Corner	3.86E-02	bodytrack	4.68E-01
Edge Detection	2.85E-02	blackscholes	2.20E-01
Matrix Multiplication	3.33E-05	swaptions	1.01E-01
SAD	5.55E-03	ferret	1
Optical Flow	1.71E-02	-	-

Table 6.3: The weights of applications in *WS2*.

TCPA applications	Weights	PARSEC applications	Weights
FIR	5.1-05	x264	1
Harris Corner	1	bodytrack	1
Edge Detection	1	blackscholes	1
Matrix Multiplication	1	swaptions	1
SAD	1	ferret	1
Optical Flow	1	-	-

Table 6.4: The weights of applications in *WS3*.

TCPA applications	Weights	PARSEC applications	Weights
FIR	7.87E+04	x264	2.89E+00
Harris Corner	1.16E+02	bodytrack	9.56E+00
Edge Detection	1.57E+02	blackscholes	2.03E+01
Matrix Multiplication	1.35E+05	swaptions	4.45E+01
SAD	8.06E+02	ferret	4.48E+00
Optical Flow	2.61E+02	-	-

Table 6.5: The weights of applications in *WS4*.

be executed on Alpha and ARM cores. Additionally, computational-intensive applications are used, which are "SAD", "Edge Detection", "FIR", "Optical Flow", "Matrix Multiplication", and "Harris Corner" that can be executed either on Alpha and ARM cores as single-threaded applications, or on the adopted TCPA accelerator, which has the ability to parallelize these applications. The application scenarios adopted in our experiments are shown in Table 6.1.

For the weights of the targeted applications, four sets of weights are defined, namely, *WS1*, *WS2*, *WS3*, and *WS4*. The first set of weights, *WS1* is listed in Table 6.2. In *WS1*, the weights are set in two steps. First, we adjust the weights so that the weighted performance of all applications are normalized to the performance of one application, namely "ferret". Hence, to calculate the weight of any application, we divide its performance by the performance of "ferret". In this case, the weight of "ferret" is equal to 1, and the performances of all applications are normalized to the performance of "ferret". The performance of application k, that is used in calculating its weight, is the maximum one from its table $R_k(\tau, h, f)$, which is obtained at the best configuration; i.e., highest V/f level and highest number of threads. As mentioned above, the weights can be used to assign different priorities to the applications. Thus, it is arbitrarily chosen to give TCPA applications higher priorities than PARSEC applications by doubling their weights, as an example. These weights, referred to as *WS1*, are listed in the Table 6.2.

In the second set of weights, *WS2*, we choose the weights of the applications, so that their weighted performances are normalized to the performance of the application "ferret", similar to *WS1*, but without assigning different priorities to the applications. Thus, all the applications will have the same priority.

In *WS3*, the weights of all applications are set to 1. In the last set, *WS4*, we choose the weights to normalize the performance of all applications to the value 1. For that purpose, we obtain the maximum performance of each application k from its table $R_k(\tau, h, f)$, and assign this value to w_k. Hence, the maximum weighted performance of any application ($w_k.R_k$) will be equal to 1. The new set of weights are listed in Table 6.5.

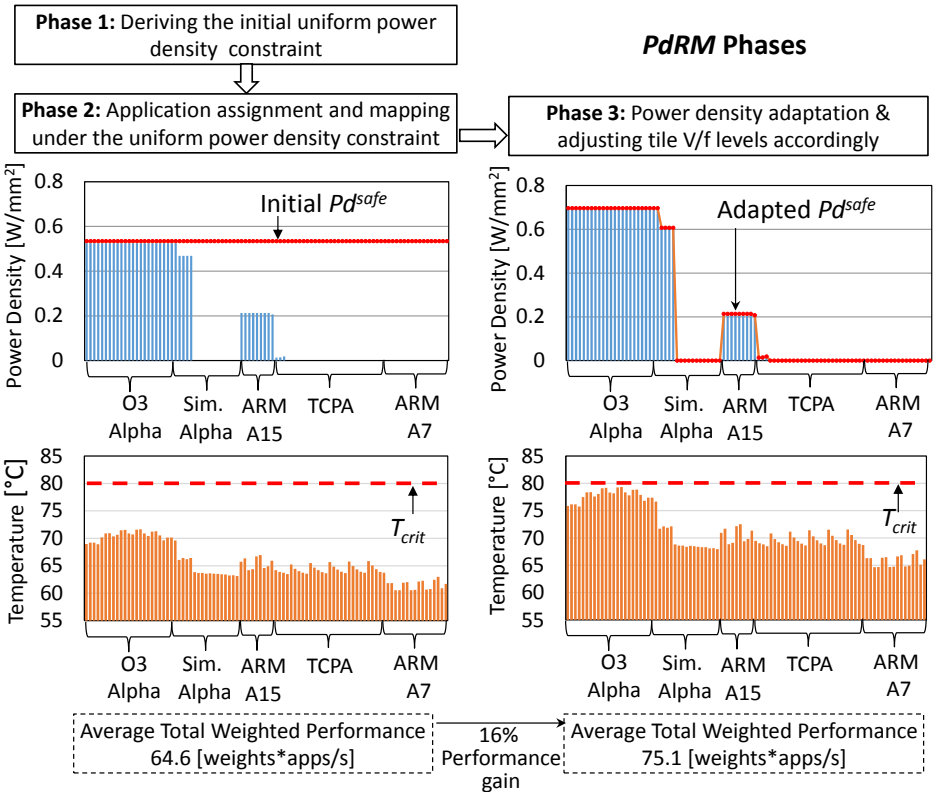


Figure 6.6: Demonstration of applying the proposed phases of *PdRM*. The cores shown in the X-axis of the charts are located according to the floorplan shown in Figure 4.6.

6.5 Evaluation results

In this section, the evaluation of our technique is presented. Firstly, the phases of *PdRM* and how it adapts to any workload changes are demonstrated. Secondly, several experimental evaluations are conducted to compare *PdRM* with several state-of-the-art techniques. Finally, the overhead of *PdRM* is discussed.

6.5.1 Demonstration of PdRM

In order to illustrate the behavior of our proposed technique *PdRM*, we conduct a demonstration experiment considering Floorplan A (Figure 4.6) and executing 6 applications

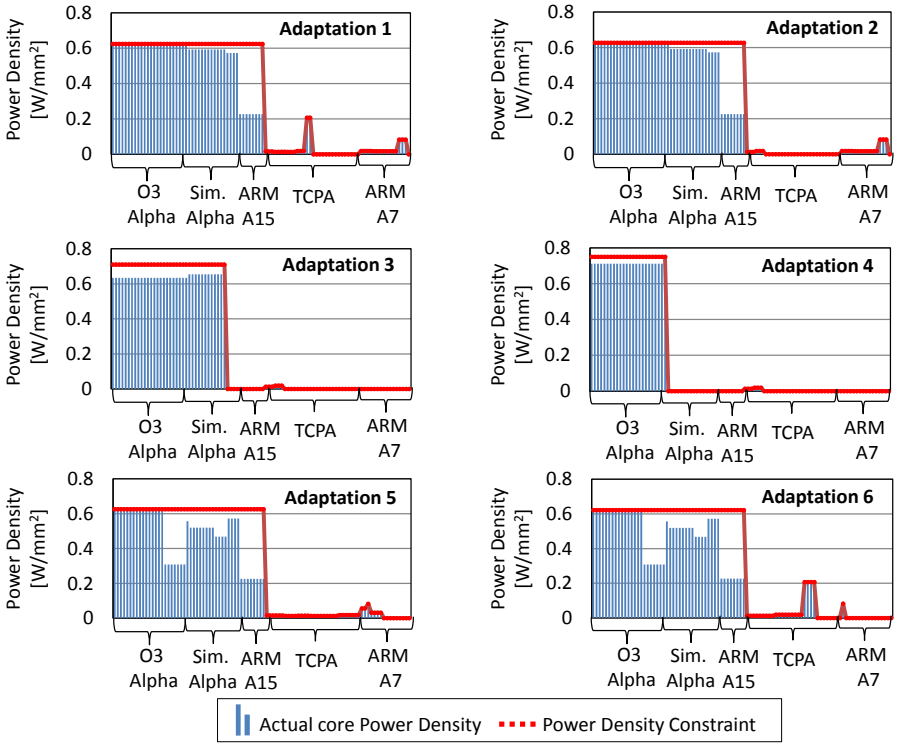


Figure 6.7: Demonstration of the runtime adaptation of *PdRM* under workload changes. The power density budget in active cores is increased when applications finish, and it is later decreased when new applications arrive. The cores shown in the X-axis of the charts are located according to the floorplan shown in Figure 4.6.

on it as detailed in Figure 6.6-Phase 2. In this experiment, the intermediate results and phases are detailed, as shown in Figure 6.6.

In the first phase, we derive the power density constraint for our adopted chip that allows running all the cores under the critical temperature, which is set to 80 °C. Given the floorplan of the chip and the critical temperature, according to Equation (6.4) from Section 6.3.1, we derive a safe power density equal to $Pd^{\text{safe}} = 0.534\,\text{W/mm}^2$, as shown in Figure 6.6-Phase 1. Afterwards, under the initial power density constraint, we apply our mapping scheme presented in Section 6.3.2. The resulting weighted performance, temperature profile and the actual power density values are shown in Figure 6.6-Phase 2. From the temperature profile, it can be observed that a large thermal headroom exists on all cores. Furthermore, by looking at the resulting power density on the cores, it is noticed that, while some cores utilize almost the whole budget, there are several cores that have a resulting power density which is much smaller than the given $0.534\,\text{W/mm}^2$ budget. This happens because different core types have different power consumption behaviors, even when they execute the same applications at the same V/f levels and occupy similar areas.

To exploit this issue, the runtime power density adaptation algorithm presented in Section 6.3.3 is applied. Thus, the budget assigned to low power density cores/applications is reduced, and the budget assigned to high power density cores/applications is increased, under the specific workload and mapping. As a result, the thermal headroom is decreased and hence the overall weighted performance is increased, as shown in Figure 6.6-Phase 3.

Figure 6.7 shows how *PdRM* adapts the power density budgets at runtime in order to react to workload changes and optimize the overall weighted system performance. From this figure, it is observed how the budget in active cores is increased when some applications finish, and how it is later decreased when new applications arrive. Moreover, *PdRM* prevents any thermal violations even when new applications arrive. This is achieved by the power density adaptation in Equation (6.4), that provides a thermally safe power density constraint.

6.5.2 Comparison with State-of-the-art Techniques

After evaluating the phases of our technique and the thermal profile for a specific scenario as an example, we conduct other experiments considering various mixed application scenarios in order to compare the overall system weighted performance achieved by our *PdRM* and other state-of-the-art approaches that share the same goal of maximizing the performance under a temperature constraint.

The first comparison candidate is the Intel's Turbo Boost [67, 126] and with the solution proposed in [57], in terms of the overall weighted performance, temperature, and power efficiency. Besides implementing these two techniques, a DTM technique similar to the one proposed in [118], which is a reactive control that reduces the V/f levels of the cores when the critical temperatures are exceeded. When the temperatures on the chip are reduced again, the V/f levels will be returned to their original nominal values. The control period of the adopted DTM is set to 1 ms. DTM will be activated on the chip along with all comparison candidates, except *TurboBoost*, which has its own thermal control methodology as explained in [67].

Intel's Turbo Boost [67, 126] allows cores to run at high V/f levels when there is available headroom. Particularly, the V/f levels of the cores are boosted in single steps (within a control period). If the temperature exceeds the critical threshold, the V/f levels are reduced in single steps until the temperature goes below the threshold.

The second comparison candidate is the solution presented in [57], which performs power management for heterogeneous tiled multicores under a unified chip-wide power budget (TDP). We referred to this technique as *TDPcontrol*. *TDPcontrol* reduces the power consumption of the tiles in proportion to the number of tasks the tile is executing, so that the overall power is brought below TDP. Since the results of this scheme depend on the value of TDP, these three cases are considered: 160 W, 200 W, and 240 W.

The last comparison candidate is a simple baseline technique, *MaxSpeed*, that sets the V/f level at the maximum possible values without considering any power constraint, but it rather depends on the DTM of the chip to throttle down the V/f levels when a thermal violation happens. Once the temperatures are decreased, the V/f levels are again set to the maximum values. The purpose of this technique is to evaluate the efficiency of only depending on the default DTM of the chip, while keeping setting the frequencies to the maximum values when possible.

To perform the comparison, experiments are conducted for each technique and for all the workloads of the application scenarios defined in Table 6.1. For fair comparison, the mapping scheme proposed in Section 6.3.2 are used with all comparison candidates, while each technique controls the V/f levels based on its methodology. Moreover, some cores might be power-gated in order to satisfy the constraints of these techniques. As above

mentioned, DTM will be activated with all techniques except *TurboBoost*. That holds when the technique could not prevent the thermal violations, DTM will be triggered. In these experiments, each workload is executed in a closed system manner. That means the applications of each workload will be re-executed until the end of the simulation, which is set to 10 s. Afterwards, the overall weighted performance is measured for all comparison candidates. Intuitively, the more efficient the technique, the higher the performance will be.

Figure 6.13 shows the comparison of the resulting overall weighted performance achieved by our *PdRM* and the evaluated state-of-the-art solutions and for all sets of weights, *WS1*, *WS2*, *WS3* and *WS4*, defined in Section 6.4 and shown in Tables 6.2, 6.3, 6.4, and 6.5. From this figure, it is notcied that *PdRM* outperforms all of the evaluated state-of-the-art approaches for all sets of weights. For example, for *WS1*, *PdRM* achieves average improvements in the total weighted performance of 1.16x, 1.91x, 12.8x compared to *TurboBoost*, *MaxSpeed*, *TDPcontrol*, respectively. In *WS3*, where the weights of all applications are set to 1, we notice that the performance of the different applications are not balanced. For example, we observe incomparable performance between *TDPcontrol* and other techniques. The reason is: *TDPcontrol*, due to the power budget limitation, could execute only PARSEC applications which have much longer execution time (less performance) than TCPA ones, and that results in very less final total performance in compared to the other techniques that could execute TCPA applications, as well. Indeed, these experiments stress the importance of using the weights. The results of applying *WS4* are similar to the ones of *WS2*, but with different absolute values of the total weighted performance.

Avoiding DTM triggers is an essential benefit of *PdRM* which contributes to the performance gain achieved by it, as illustrated in the detailed comparison in Figure 6.9. However, there is another factor that also contributes to the performance gain, which is exploiting thermal headroom on the chip. *PdRM* considers the power properties of the applications and cores, in order to decrease the budget assigned to low power density cores/applications, and increase the budget assigned to high power density cores/applications. That will exploit any potential thermal headroom and maximize the utilization of the chip cores. Figure 6.8 shows comparison of the average utilization of the chip between our *PdRM* and the other state-of-the-art techniques. The utilization of each core is calculated by dividing its assigned frequency by the maximum possible frequency of that core. When a core is power-gated, its utilization will be zero. By averaging the utilization of all cores through the whole simulation time, the average utilization of the chip is obtained. Figure 6.8 shows that *PdRM* maximizes the utilization of the chip by 1.24x and 2.63x compared to *TurboBoost* and *TDPcontrol*, respectively. To this end, we can observe the efficacy of *PdRM* in dark silicon era, because maximizing the utilization of the chip resources will alleviate dark silicon problem.

Among the state-of-the-art, it is obvious that *TurboBoost* is the most efficient technique. *PdRM* results in better performance than *TurboBoost*, thanks to the adopted power density constraint that enables the former from increasing the V/f levels of the tiles safely without exceeding the critical temperature. On the other hand, *TurboBoost* keeps increasing the V/f levels for all active cores, and when the critical temperature is exceeded, it throttles down again the V/f levels until the cores are cooled down. Thus, it keeps oscillating around the critical threshold through the execution time (see Figure 6.9). From Figure 6.9, we observe that *TurboBoost* can result in an instantaneous higher frequency, but on average, *PdRM* has 12% higher frequency. Additionally, to

provide detailed comparison between *PdRM* and the most efficient state-of-the-art technique, *TurboBoost*, the power efficiency of both of them is calculated. Figure 6.10 first shows how *PdRM* outperforms *TurboBoost* by 16% on average. While the power consumption of *PdRM* is 46% less. As a result, the power efficiency of *PdRM* is 105% more than *TurboBoost*. This adds more significant value to our *PdRM*.

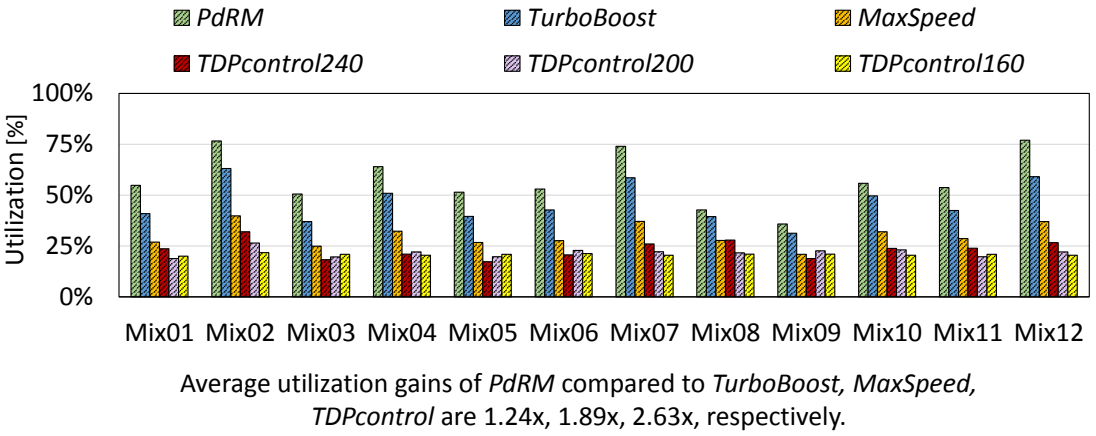


Figure 6.8: Comparison between the average chip utilization of our *PdRM* and several state-of-the-art techniques, while executing various scenarios of workloads.

The solution presented in [57], *TDPcontrol*, is the one that results in the lowest performance for all tested TDP values. In order to satisfy TDP constraint, *TDPcontrol* reduces the power consumption of each tile in proportion to the number of active cores in the tile, and without considering the heterogeneity in the power characteristics of the cores. As a result, the cores that have high power consumption (power hungry cores) can achieve the required power reduction of their tiles, with scaling down their V/f levels with few steps. On the other hand, the cores that have low power consumption will reach their minimum V/f levels before achieving the required power reduction of their tiles, and thus, these cores are power-gated. As a consequence, a considerable subset of applications could not be executed resulting in a very low total weighted performance. Additionally, *TDPcontrol* might lead to excessive triggers of DTM, like the case of 240 W and 200 W power budgets, because the power hungry cores will run at high V/f levels and consume the major part of the power budget, and that, in turn, leads to thermal violations. Thermal violations will trigger DTM, which significantly degrades the performance. Details about the work of DTM are illustrated in Figure 4 of the supplementary material. Additionally, we show the resulting thermal profile of the chip from applying *PdRM* and *TDPcontrol* at 240 W without activating DTM (see Figure 6.12). This figure shows how *PdRM* results in a safe thermal profile without the need of triggering DTM.

Additionally, in order to demonstrate the robustness of *PdRM*, additional experiments are conducted to evaluate *PdRM* with probabilistic power and execution time values. In particular, a normal distribution is adopted to generate for each application k random power variations over time, so that the mean of these values is equal to the average power value stored in P_k, while the deviation of these values is set to 20%. Similarly, variations in the execution time of each application k are generated. The results of these new experiments (Figure 6.11) do not significantly differ from the original ones. However, *PdRM* does not guarantee avoiding the thermal threshold in this case, and

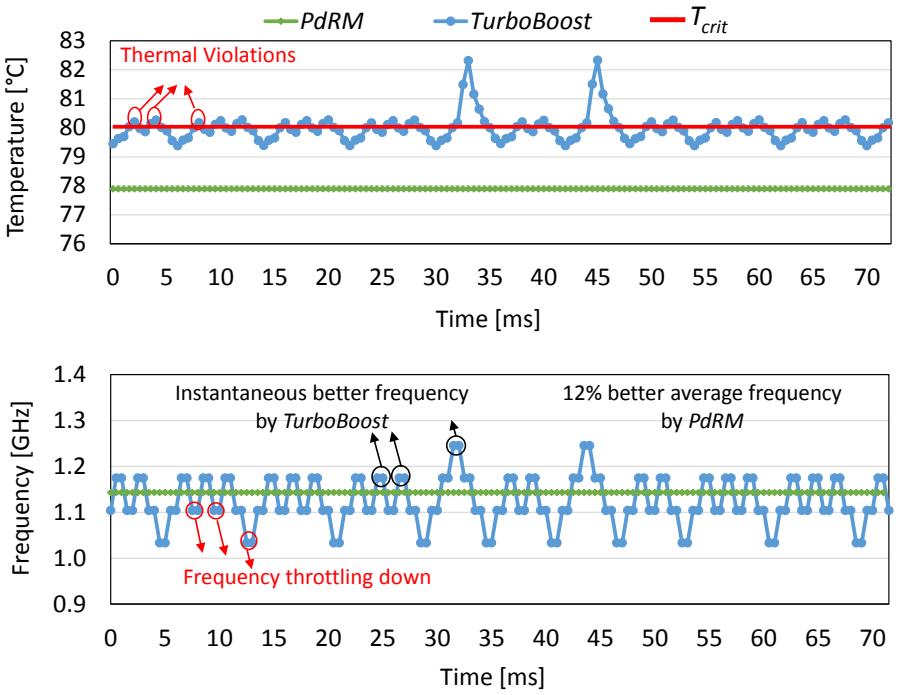


Figure 6.9: Comparison between the resulting temperatures and the frequencies of *PdRM* and *TurboBoost*.

two thermal violations are observed in the experiment of mix10. Nevertheless, *PdRM* still outperforms all other comparison candidates.

6.5.3 PdRM Overhead

The overhead of *PdRM* consists of the following parts: The first part is the overhead of calculating the uniform *PdRM* based on the RC thermal network. The second part is the application mapping overhead, which depends on the number of applications that are mapped. The last part is the overhead of the the power density adaptation. From the experiments that conducted on different workloads (Table 6.1), the average overhead of the aforementioned parts are as follows: 2.6 ms for the first phase of *PdRM*, which is executed only once at design time, 3 ms for application mapping, which is executed at the startup time of the applications, and 2.5 ms for power density adaptation, which is required after any change in the workload. Additionally, in order to test the overhead of the power density adaptation in the case of frequent workload changes, an additional scenario is considered, which has one workload change at each second. For that scenario, the total overhead of our *PdRM* is about 0.4% of the total execution time. According to these results, the overhead of *PdRM* is very small compared to the performance gain achieved by it.

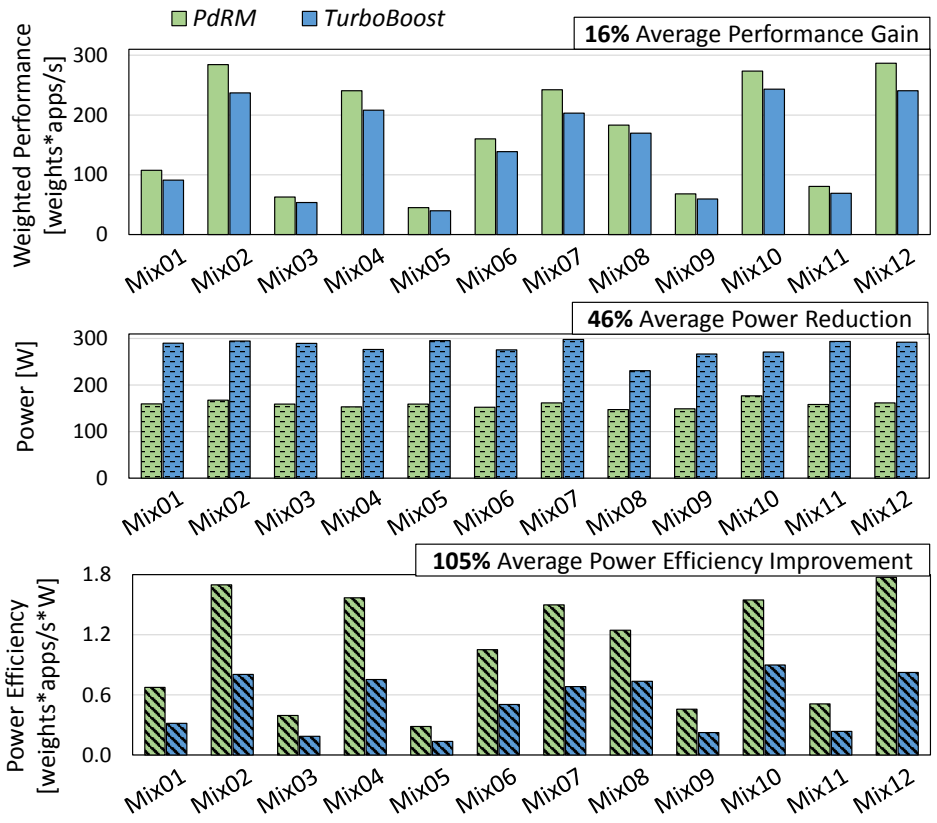


Figure 6.10: Comparison between the performance, power and power efficiency of *PdRM* and *TurboBoost*.

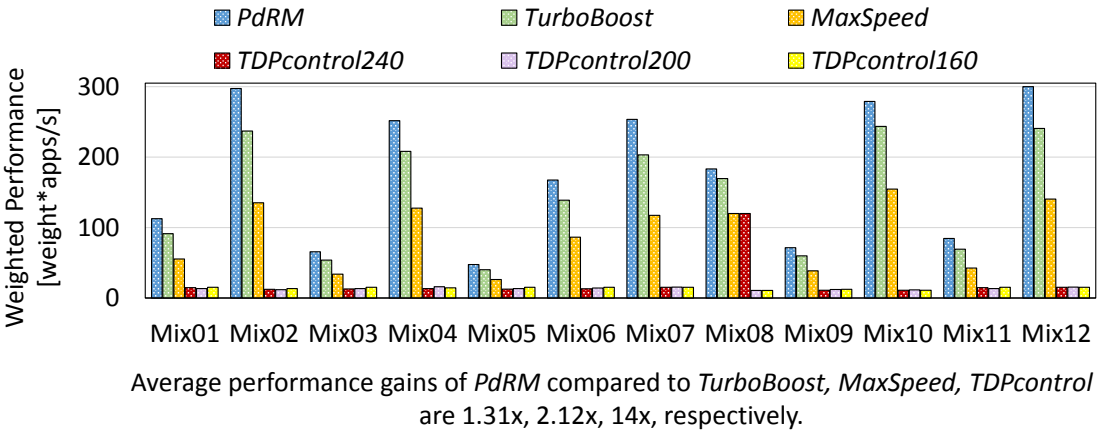


Figure 6.11: The resulting weighted performance under probabilistic power and execution time values.

6.6 Summary

This chapter has presented an efficient resource management technique based on power density constraint, which aims at maximizing the overall system performance on heterogeneous tiled multicores without violating a predefined critical temperature. The introduced novel power density constraint is a potent yet simple approach that guarantees

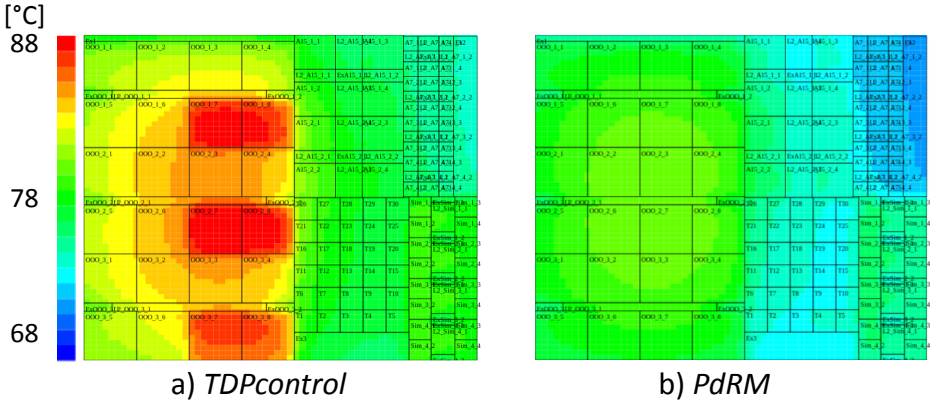


Figure 6.12: Comparison of the resulting steady-state temperatures of our *PdRM* and *TDPcontrol* [57] running Mix10 workload and assuming that DTM is deactivated.

avoiding thermal violations and considers the underlying core heterogeneity. Furthermore, it is shown that runtime adaptation of this power density constraint by considering the power characteristics of the executed applications on different types of cores and accelerators can result in further performance improvements. The evaluation has demonstrated that *PdRM* results in up to 14x higher average performance compared to state-of-the-art approaches. Therefore, this approach can motivate researchers to move from chip-level power budgets to dynamically adapted power density constraints, which are more efficient and heterogeneity-aware.

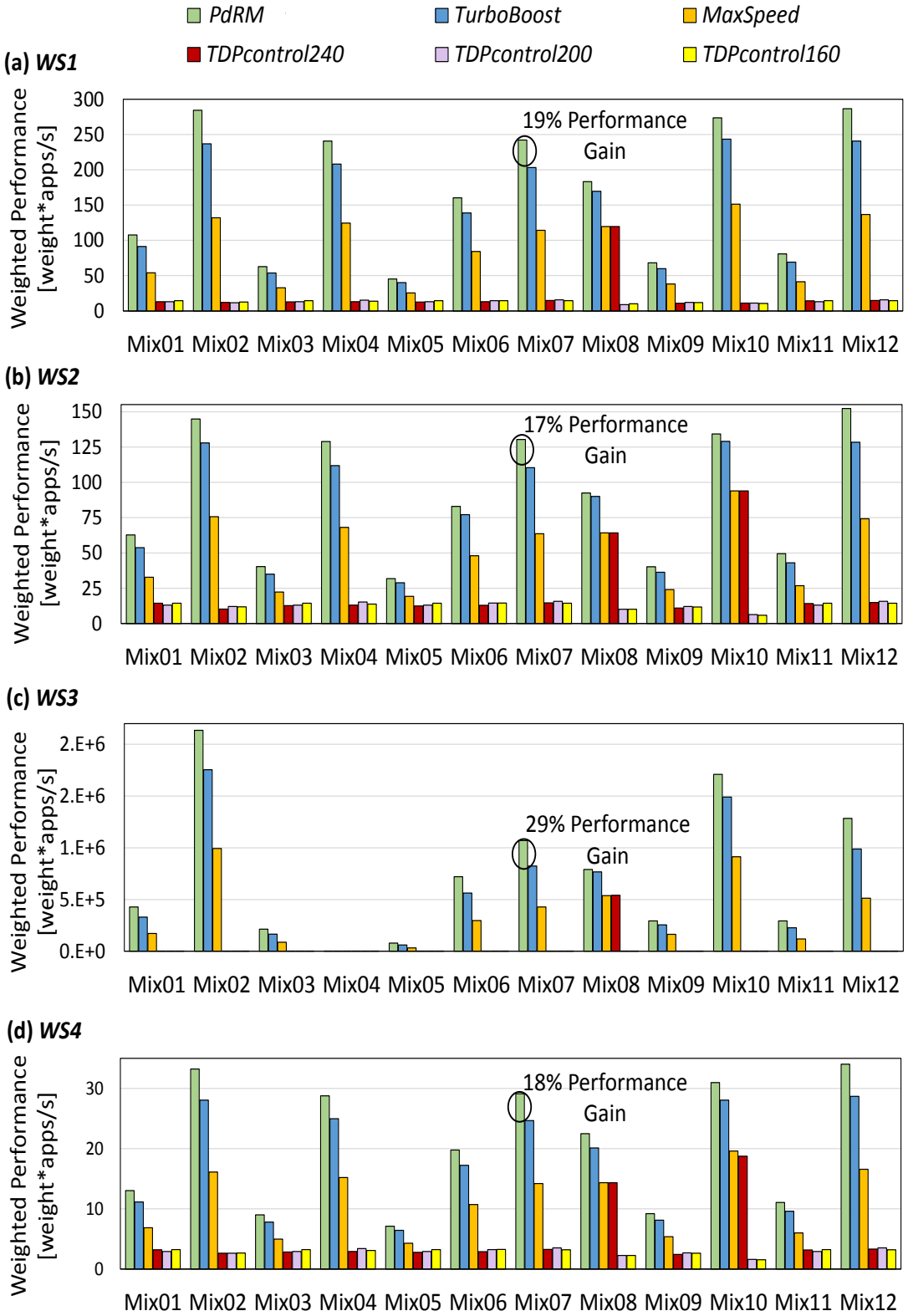


Table of the average performance gains of ***PdRM*** compared to comparison candidates for different weight sets:

	WS1	*WS2*	*WS3*	*WS4*
TurboBoost	1.16x	1.12x	1.23x	1.15x
MaxSpeed	1.91x	1.58x	1.79x	1.59x
TDPcontrol	12.8x	7.11x	1E+4x	7.04x

Figure 6.13: Comparison between the resulting overall system performance of our *PdRM* and several state-of-the-art techniques, while executing various scenarios of workloads. The experiments are conducted for four different sets of weights, *WS1*, *WS2*, *WS3*, and *WS4*, as defined in Section 6.4.

Chapter 7

Aging-Constrained Resource Management

As discussed earlier, system-level decisions may be the cause of circuit aging: Aiming at maximizing performance, resource management might increase the number of parallel threads of the applications and upscale the V/f levels of the cores, and that, in turn, raises the temperature. Increased V_{dd} and high temperature induce an accelerated aging manifesting as an increase in the threshold voltage (V_{th}). An increase in the threshold voltage (ΔV_{th}), in turn, may increase the critical path of the processor and eventually lead to timing violations and/or errors. To sustain reliability, chip designers add to the clock delay a timing guardband to compensate for ΔV_{th}. Determining the width of the guardband necessitates enforcing an upper bound (or constraint) for the aging (i.e., ΔV_{th}). This aging constraint is referred to as ΔV_{th}^{m}.

The state of the art imposes conservative V_{dd} and temperature bounds so that the *amount of aging* (i. e. ΔV_{th}) is safely kept below a predetermined aging constraint ΔV_{th}^{m}. The disadvantage of this practice is that the resource management's goal of maximizing performance is then limited. Hence, it is indispensable to explore the trade-offs between all involved relevant parameters and constraints, i.e. V/f levels, number of parallel threads, temperature, aging, and performance (see Figure 1.4) in order to achieve the optimization goal of maximizing performance under aging constraints.

Given this motivation, this chapter presents a new aging-aware design space that translates temperature and V_{dd} into an *amount of aging*, quantified by ΔV_{th}, based on an accurate physics-based aging model (Section 3.4). This new design space and its exploration enables a resource management to maximize the performance without the restrictiveness of the state of the art. Thus, a novel aging-constrained resource management technique referred to as *AgRM* is presented to exploit this aging-aware design space with the goal of maximizing the performance under an aging constraint. More specifically, the presented *AgRM* technique adjusts the system level parameters, i.e. the number of parallel threads of the applications and the V/f levels of the cores, so that the performance is maximized (optimization goal) under an aging constraint. The design flow for the presented aging-constraint resource management technique is shown in Figure 7.1.

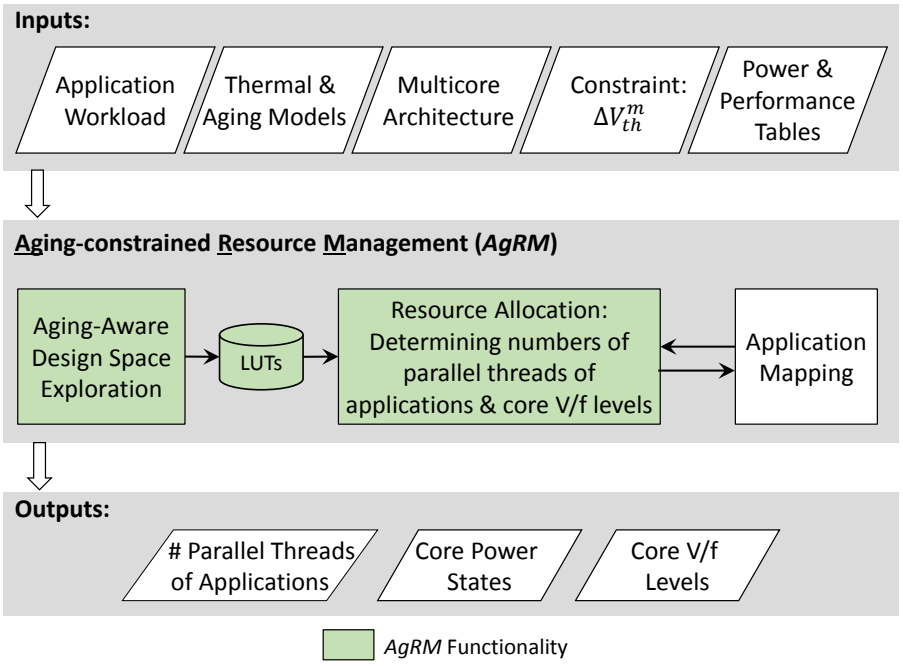


Figure 7.1: An overview of the proposed aging-constrained resource management.

7.1 Aging-Aware Design Space

7.1.1 Relevance of Accurate Aging Models

State-of-the-art aging models are empirical as they estimate ΔV_{th} [127] through heuristics i.e. without modeling the physical mechanisms. Thus, they only work for those temperature and V_{dd} values where measurements have been conducted. The purpose of these models is to provide a rapid estimation of aging for a projected lifetime under worst-case conditions. Therefore, it is infeasible to employ these empirical aging models for the purpose of a design space exploration where wide ranges of temperature and V_{dd} must be considered and where accuracy is key. In fact, temperature and V_{dd} are *not* independent with regard to aging. For instance, the reduction in ΔV_{th} due to lower V_{dd} can be diminished if temperature increases and vice versa. Hence, the *joint* impact of temperature and V_{dd} on V_{th} needs to be considered. In other words, system-level decisions like number of threads, V/f levels etc. indispensably necessitate an aging model that is able to correctly and accurately quantify the physical mechanisms that induce degradation (ΔV_{th}). The underlying differential equations in a physics-based aging model describe precisely the diverse defect generation mechanisms under the dependency of temperature and voltage. This is what gives the physics-based aging model the capability to consider *any arbitrary combination of temperature,* V_{dd}, contrary to empirical aging models. Therefore, the state-of-the-art physics-based aging model [106] is employed as seen in Figure 3.5. The details of the adopted aging modeling are discussed in Section 3.4.

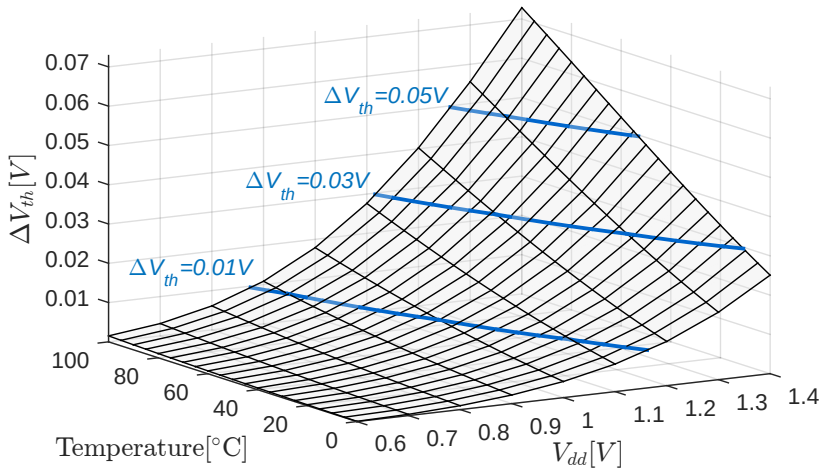


Figure 7.2: A new aging-aware design space: Accurate interdependencies between *amount of aging* (i.e. ΔV_{th}), temperature and V_{dd}, as demonstrated in Section 3.4

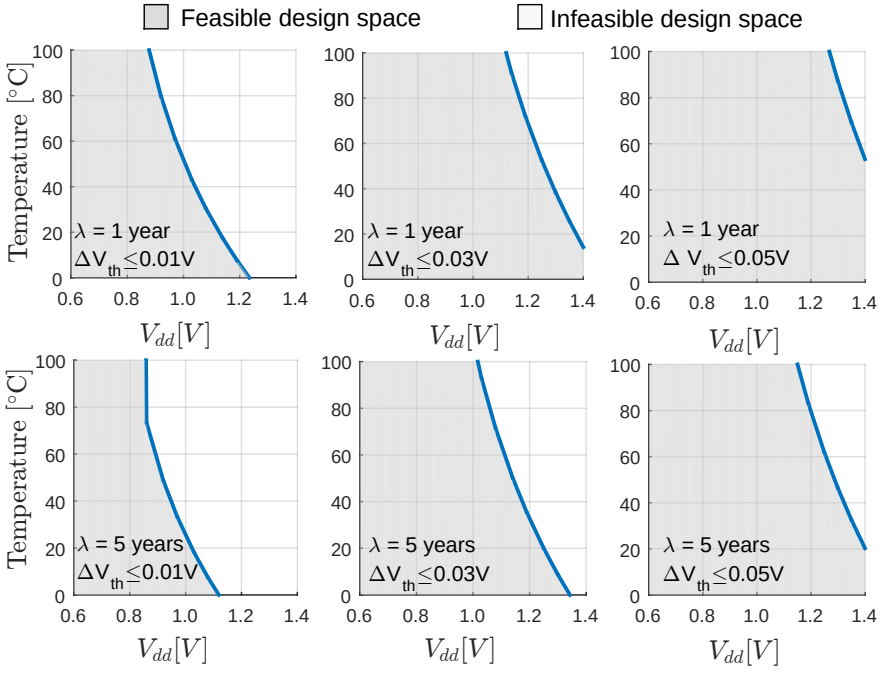


Figure 7.3: Feasible design spaces for varying aging constraints and lifetimes.

7.1.2 Design Space Exploration (DSE)

This section presents a new design-space and its exploration that links parameters and constraints as presented in Figure 1.4. Firstly, several design spaces are generated for various targeted lifetimes and aging constraints. Secondly, we explore one of these design spaces along with varying system-level parameters, i.e., V/f levels and number of parallel threads.

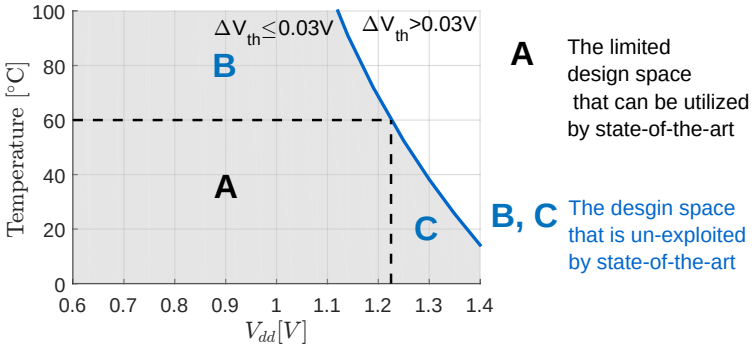


Figure 7.4: The *entire* design space that satisfies the aging constraint and the *limited* design space resulting from applying conservative temperature and V_{dd} bounds (dashed area) by the state of the art.

7.1.3 DSE for Various Lifetimes and Aging Constraints

Using the aging model presented in Section 3.4, we estimate the *amount of aging* for temperatures and V_{dd} within the ranges $[0\,°C \cdots 100\,°C]$ and $[0.6\,V \cdots 1.4\,V]$, respectively, and considering different targeted lifetimes (λ), i.e., 1 year, 5 years and 10 years. For an example, ΔV_{th} after one year is shown in Figure 7.2. The three cut-off lines represent specific aging constraints[1]. The design space of V_{dd} and temperature below each of these cut-off lines represents *feasible* options since they satisfy the aging constraint. Figure 7.3 shows different feasible design spaces for various aging constraints and lifetimes. Apparently, higher targeted lifetimes reduce the feasible design space.

Without loss of generality, we consider the scenario of the aging constraint $V_{th}^{m} = 0.03\,V$ and the lifetime $\lambda = 1$ year as an example and explore it further in Figure 7.4. The first observation is that the (blue) cut-off line can be translated into a *set of temperature constraints*, each corresponds to a specific V_{dd} so that the aging constraint is satisfied. For instance, if V_{dd} is set to 1.25 V, the temperature constraint will be equal to 52°C. However, if a lower V_{dd} is selected, e.g. 1.15 V, the temperature may reach 86°C, without violating the aging constraint. This is a key observation. However, the state-of-the-art technique [82] enforces one pair of temperature and V_{dd}. For instance, if the maximum allowed V_{dd} is 1.225 V, the temperature constraint is 60°C (two dashed lines in Figure 7.4). Therefore, a resource management cannot, by principle, select temperature and V_{dd} outside this *limited* space. In contrast, considering the *entire* aging-aware design space enables the selection of more temperature and V_{dd} values and hence opens new optimization potential.

7.1.4 DSE for Various System-Level Parameters

The new aging-aware design space is explored with (varying) system-level parameters: number of threads and V/f levels of the cores. The V/f level range is from (0.6 V, 0.8 GHz) to (1.4 V, 4.0 GHz). The step between these levels is 0.05 V and 0.2 GHz. The following application scenarios are discussed:

[1] All assumptions for parameter values are exemplary. The presented technique does not depend on these specific values nor these values are the limits or recommendations of any data sheets etc. They are solely chosen for illustrative purposes of the presented technique.

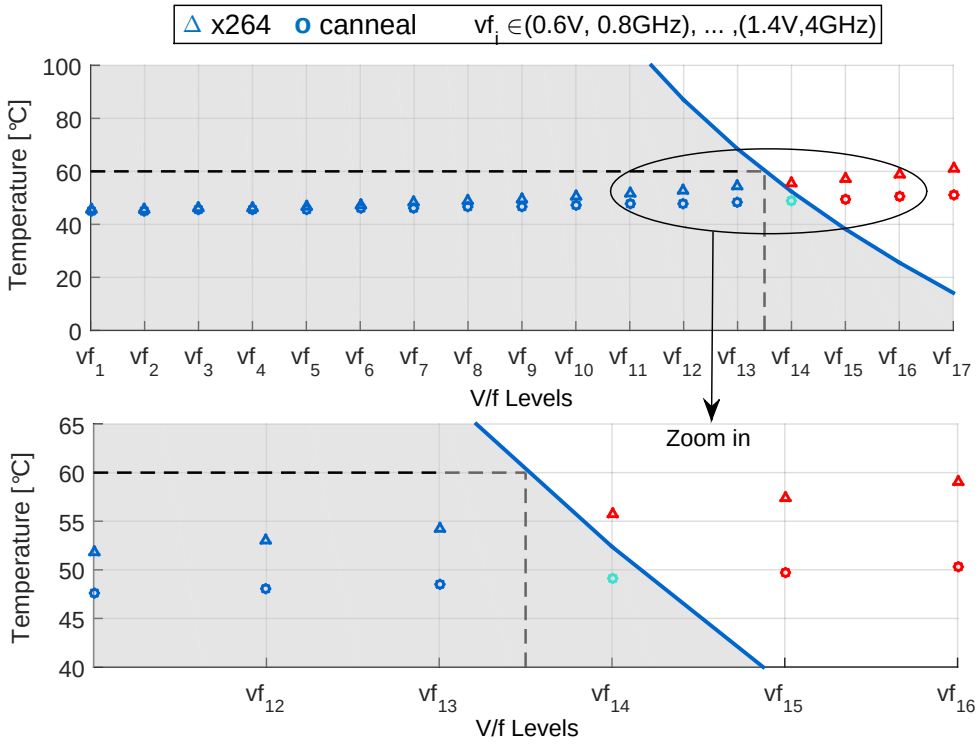


Figure 7.5: The resulting temperatures (Equation (8.4)) of running one single-threaded application on one core for different V/f levels whose V_{dd} values cover the adopted V_{dd} range, i.e., [0.6 V, 1.4 V]. The red design points represent the temperature points that are outside the feasible design space. The blue ones are within. Considering the entire design space allows choosing vf_{14}, which corresponds to (1.25 V, 1.34 GHz).

The first scenario considers two applications from PARSEC benchmark suite [41], "x264" and "canneal", each runs only a single thread individually on the chip. Figure 7.5 shows the resulting temperatures from running these applications for different V/f levels. As seen, the aging constraint is violated at vf_{14} (i.e., when the V_{dd} reaches 1.25 V), in the case of "x264". While in the case of "canneal", the V/f level can be reached (i.e., V_{dd} can be set to 1.25 V) without violating the aging constraint because "canneal" has lesser power consumption and therefore lower temperature. Again, it is of key importance that this opportunity of exceeding the V/f level is not possible by the the state of the art that must stick with a specific pair of temperature and V_{dd}.

Figure 7.6 illustrates the second scenario, in which 8 instances of "x264" are running simultaneously on the chip. Varying numbers of threads for these applications (i.e., 1, 2, 4, 8) are examined. It can be noticed that the maximum performance that can be obtained considering the *entire* design space is 54% more than the maximum obtained performance when the limited (region A) design space (as used by the state of the art) is employed.

In summary, there is a great potential advantage when employing the *entire* design space since it opens new optimization potential for system-level techniques that determine parameters like number of threads, V/f levels of the cores etc. To provide the resource management techniques with the aging-aware design space, we construct look-up tables

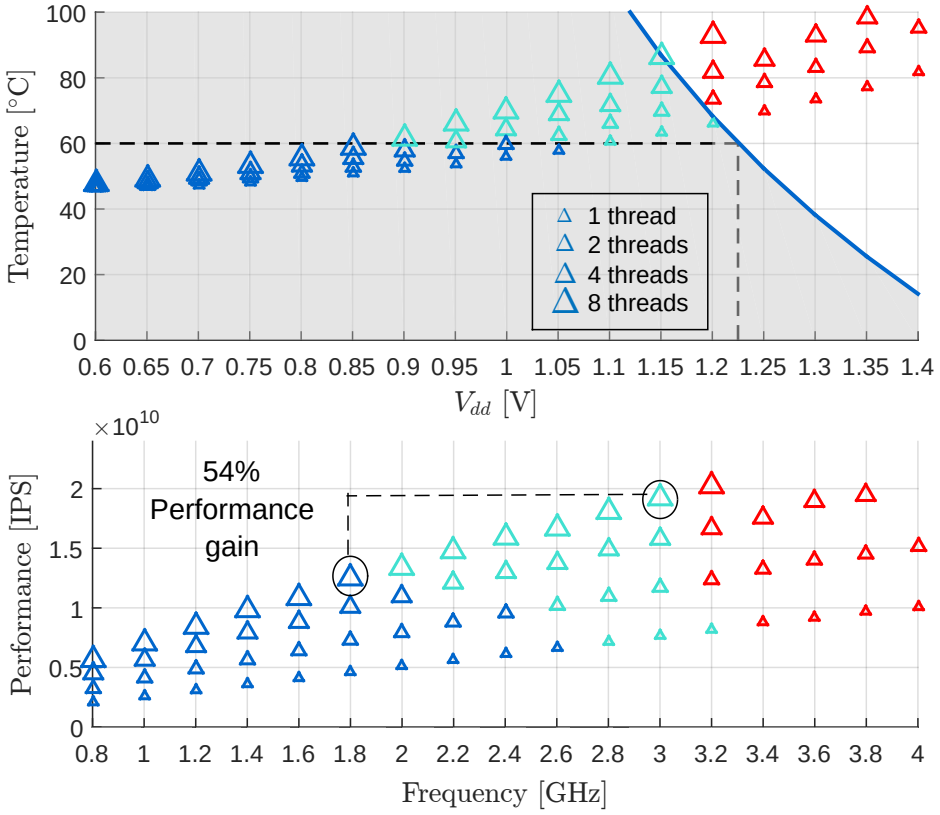


Figure 7.6: The maximum temperatures and the average performance, resulting from running eight instances of "x264", with multiple threads for each, and for different V/f levels. The points that are surrounded by circles represent the maximum obtained performance within the limited space and the complete one. Considering the whole design space allows increasing both the number of threads and the V/f levels, and hence increasing the performance.

(LUT) for different pairs of aging constraints and lifetimes. Each table stores the temperature constraints that satisfy the given aging constraint after the targeted lifetime for all possible V/f levels.

7.2 Problem Formulation

Based on the explored aging-aware design space, a new resource management technique is proposed to exploit the full potential of performance maximization under an aging constraint ΔV_{th}^{m} for a specific lifetime λ and a workload of K applications running on N cores. The objective is to select for each application k the number of threads h_k and the V/f level, vf_k, so that the system performance is maximized:

$$\text{Select } h_k, vf_k \text{ for all } k = 1, 2, \ldots, K, \text{ in order to:}$$
$$\text{Maximize } \sum_{k=1}^{K} R_k(h_k, vf_k) \text{ , s.t:}$$
$$T_k^{app} \leq LUT_{\Delta V_{th}^{m},\lambda}(vf_k) \text{ And } \sum_{k=1}^{K} h_k \leq N$$

There, the application performance $R_k(h_k, vf_k)$ is estimated using Equation (3.2). T_k^{app} denotes the maximum temperature of the cores that execute application k. $LUT_{\Delta V_{th}^m,\lambda}$ refers to a look-up-table (constructed based on the aging-aware design space) that stores the temperature constraints that satisfy the aging constraint V_{th}^m after a lifetime of λ for all possible V/f levels. This combinatorial optimization problem, unfortunately, cannot be reduced to any kind of *Knapsack* problem because, among others, the constraints are not global but local at the level of the cores of each application. Each combination C consists of a list of K pairs of a V/f level and a number of threads, i.e., $\{(vf_1, h_1), (vf_2, h_2), \cdots, (vf_K, h_K)\}$, such that each pair corresponds to one application. Enumerating all combinations is of exponential complexity. However, there are bounds of the parameter space: For example, as seen in Figure 7.5, each application has a maximum feasible (without violating the aging constraint) V/f level. This along with further existing bounds (explained later on), makes the *branch and bound* algorithm the best choice.

7.3 Aging-Constrained Performance Maximization

The aforementioned problem is solved using a branch and bound algorithm. The algorithm starts from an initial combination $C^\star$ as the root node of the search tree. this root node represents an upper bound of the performance but it might still violate the aging constraint. Thus, the algorithm repeatedly creates further nodes (combinations), that have lower V/f levels and/or lesser number of threads aiming to satisfy the aging constraint with a minimum performance reduction with regard to the performance of $C^\star$. It is to be noted that during this search, the resulting steady-state temperatures of each combination must be estimated. That requires determining the mapping of the applications to the cores. For this purpose, a simple[2] mapping scheme is applied, i.e. the applications are mapped to the cores in their default ordering within the workload set.

7.3.1 Finding the Root Node

Starting with a good root node is crucial for finding the best solution quickly.
Firstly, as seen in Figure 7.5, there is a maximum V/f level that the application can run at without violating the aging constraint. This maximum V/f level (denoted as $vf_k^\star$) depends on the running application (it can be higher for low-power applications), the number of parallel threads, and the number of simultaneously running applications. It is maximized when the application is executed on one core and the rest of the cores are inactive (i.e. no heat contributions from other cores). Thus, we find the maximum values of $vf_k^\star$ for all applications and assign them to the corresponding combination pairs of the root node $C^\star$ as the upper bounds for the V/f levels. Secondly, the trivial upper bound for the number of threads of each application is H_k. However, assigning H_k cores to each application k might lead to a total number of threads more than the number of the cores on the chip. Thus, it is necessary to find the number of threads of each application h_k so that the overall system performance is maximized while the total number of threads does not exceed the number of the cores. This can be obtained through dynamic programming inspired by what is proposed in the literature for similar problems [128].

[2]It is important to note that the presented resource management technique in this chapter works irrespective of the application mapping and a better mapping would improve the final results further.

Algorithm 7 Branch & Bound to maximize performance under aging constraint

Input: $C^\star, LUT$;
1: Create $RootNode$; $RootNode \leftarrow C^\star$;
2: $GLB \leftarrow 0$; $k = 0$; $Level[k].AddNode(RootNode)$;
3: **while** $k \leq K$ **do**
4: $k = k + 1$;
5: **if** $T_k^{app} > LUT(vf_k^{app})$ **then**
6: **for all** $Node$ in $Level[k-1].listNodes$ **do**
7: $Node.Branch(k, GLB, AC, Node^\star)$;
8: $Level[k].AddNodes(Node.ChildNodes)$;
9: **if** AC **and** $Node^\star.R > GLB$ **then**
10: $GLB \leftarrow Node^\star.R$; $BestNode \leftarrow Node^\star$;
11: **end if**
12: **end for**
13: **end if**
14: **end while**
15: **return** $BestNode$;

After finding the optimal number of threads of each application $h_k^\star$, the root node $C^\star$ is set as follows: $\{(h_1^\star, vf_1^\star), (h_2^\star, vf_2^\star), \cdots, (h_K^\star, vf_K^\star)\}$. It represents the upper bound of the performance with the aging constraint still to be examined. Therefore, the resulting temperatures of the cores are estimated. If for each application k, T_k^{app} does not exceed the corresponding temperature constraint to $vf_k^\star$, then that means the root node satisfies the aging constraint and the search is pruned. Otherwise, the algorithm needs to search further for the combination that does satisfy the aging constraint. Before starting to search, the list of pairs of the combination is sorted in a descending order according to the amount of the thermal violation of the cores of the corresponding application i.e. T_k^{app}-$LUT_{\Delta V_{th}^m,\lambda}$.

7.3.2 Branching and Bounding Rules

The algorithm starts with branching on the root node as reductions on the V/f level and the number of threads (to satisfy the aging constraint) of the first pair only (i.e. first application) resulting in child nodes representing the first level of the tree. Intuitively, the maximum number of the tree levels is K.

At each tree level k, the algorithm creates nodes from changing the pair k from the parent nodes at level $k-1$. It starts by reducing the V/f level and afterwards continues with reducing the number of threads. The reason is: When the V/f level is reduced first, the temperature constraint (upper bound) will be raised and therefore the possibility to satisfy the aging constraint is increased. In practice, this property also significantly reduces the number of examined steps.

Hence, the V/f levels are reduced step by step until a node is found that satisfies the aging constraint. Then, the resulting performance at this node is considered the *local lower bound* for the branching at the current parent node, and denoted as LLB. Afterwards, the algorithm continues with branching by reducing the number of threads without reducing the V/f level and then by reducing both simultaneously. In either way, the algorithm stops branching once the resulting performance is lower than LLB.

Algorithm 8 Branch on a Node

Input: k, GLB; {Branching on V/f level};
1: $vfSteps \leftarrow 1$; $LLB \leftarrow 0$;
2: **repeat**
3: create a new $ChildNode$; $ChildNode \leftarrow Node$;
4: Subtract $vfSteps$ from $ChildNode.C[k].vf$;
5: **if** $(ChildNode.R < GLB)$ **or** $(ChildNode.R < LLB$ **and** $T_k^{app}(ChildNode.C[k]) > LUT(ChildNode.C[k].vf))$ **then**
6: Remove $ChildNode$ and Break; {Stop Branching};
7: **else**
8: $Node.AddNode(ChildNode)$; $vfSteps \leftarrow vfSteps + 1$;
9: **end if**
10: **if** $T_k^{app}(ChildNode.C[k]) \leq LUT(ChildNode.C[k].vf)$ **then**
11: **if** $ChildNode.R > LLB$ **then**
12: $LLB \leftarrow ChildNode.R$;
13: **end if**
14: **if** $AC = true$ **then**
15: $Node^{\star} \leftarrow ChildNode$;
16: **end if**
17: **end if**
18: **until** $T_k^{app}(ChildNode.C[k]) \leq LUT(ChildNode.C[k].vf)$ **or** $vfSteps > vfMaxSteps$
{Branching on number of threads}
{Branching on both V/f levels and number of threads}
19: **return** $AC, Node^{\star}$;

During branching, if any node leads to satisfying the aging constraint on all cores, it is considered as a promising node for the final solution and its performance is considered a *global lower bound* for the search tree, and denoted as GLB. If the aging constraint is satisfied by another node with higher performance than GLB, then the performance of the new node replaces GLB. The last level to branch is the one that corresponds to the last application, whose T_k^{app} exceeds the aging constraint. Once the aging constraint is satisfied at this level, the promising node that has the highest performance represents the final solution. This algorithm is presented in Algorithm 7. The function that explains the branching process on one node is shown in Algorithm 8. Note that a flag called AC is used in the algorithms to indicate a satisfied aging constraint on all cores. The data structure of each node consists of the following fields: the combination C, the resulting performance R, and the list of child nodes.

7.4 Experimental Evaluation

The main metric of the evaluating *AgRM* is the overall system performance. The targeted multicore system is a 64-core chip shown in Figure 4.4. The complete setup is explained in details in Chapter 4, in which the tool flow is illustrated in Figure 4.1. Varying representative application scenarios are employed (Figure 8.9), so that single and multiple application mapping are considered. The scenarios S10, S11, and S12 consider only a single application. Each application runs a varied number of threads $1, .., 8$.

Scenarios	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12
blackscholes	2	3	1	4	-	-	5	-	4	12	-	-
bodytrack	2	3	2	-	4	4	-	-	1	-	-	-
swaptions	2	3	3	4	-	4	5	6	4	-	12	-
ferret	2	3	1	-	4	-	-	-	1	-	-	-
canneal	2	3	2	4	-	-	5	-	1	-	-	-
x264	2	3	3	-	4	4	-	6	4	-	-	12

Aging Constraints	ΔV_{th}^{m} [V]	Lifetime [year]
(i)	0.03	1
(ii)	0.05	5
(iii)	0.05	10

Figure 7.7: The first table contains the application scenarios. Each cell contains a number of application instances in each scenario. The second table shows the considered aging constraints.

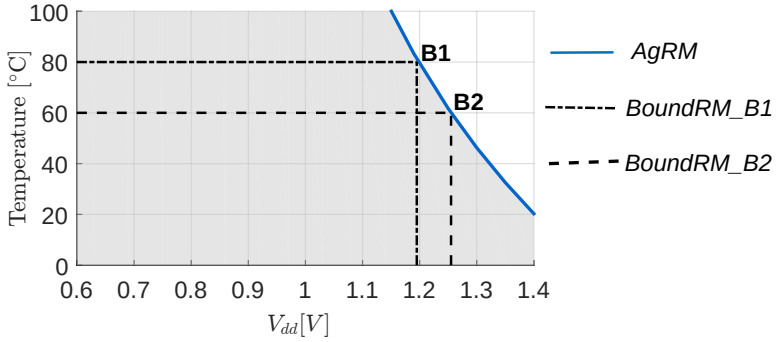


Figure 7.8: The adopted constraints by the comparison candidates.

The range of the voltage and frequency employed V/f level is from $(0.6\,\text{V}, 0.8\,\text{GHz})$ to $(1.4\,\text{V}, 4\,\text{GHz})$.

7.4.1 Comparison Candidates

The technically closest (it shares with us the same goal of maximizing the performance under aging constraint) related resource management, that also has reported the best results so far, is [82], which is referred to as *BoundRM*. They first estimate the max. voltage and temperature bounds for a targeted lifetime. Then they assign applications to cores and determine V/f levels aiming at maximizing the performance without violating the predetermined voltage and temperature bounds. It is to be noted that their *BoundRM* considers only single-threaded applications. To enable a fair comparison with *AgRM* which considers multi-threaded applications and hence it determines the number of the cores to assign to each application, it is assumed that in the case of the *BoundRM* technique, the cores are evenly distributed among the applications. Additionally, our policy of determining the number of threads is combined with *BoundRM* resulting in a new combined version referred to as *CombRM*[3]. Both *BoundRM* and *CombRM* are tested for two pairs of temperature and voltage bounds; B1 and B2, estimated using the same physics-based aging model that *AgRM* employs (see Figure 7.8). It is to be noted that the temperature bound of B1 is higher than that of B2, while the voltage bound of B2 is higher than that of B1. That makes these pairs of bounds representative.

[3]The reason for extending the state-of-the-art technique is to allow it to benefit from our policy of determining the number of threads. This way we diminish the magnitude of *AgRM* benefit compared with *CombRM* but still *AgRM* outperforms *CombRM* because of the use of the entire design space.

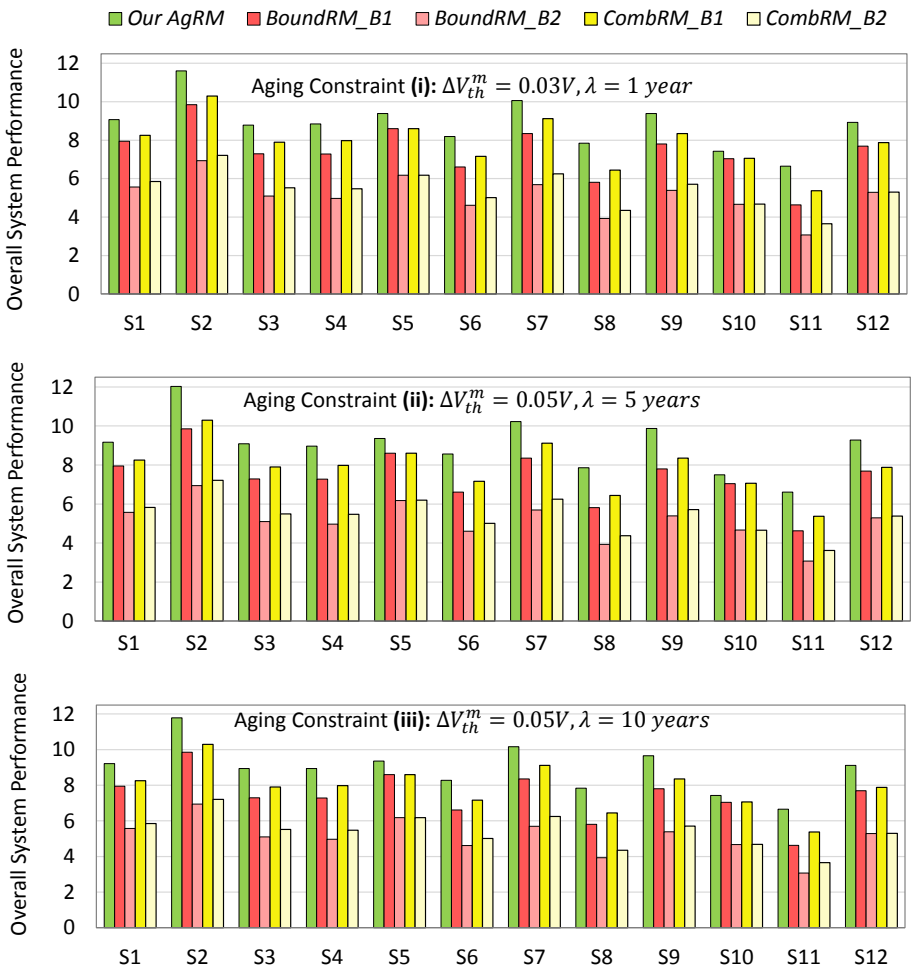
The work proposed in [82] enforces one pair of voltage and temperature bounds in the resource manager. However, comparing against [82] using only one pair of these bounds might not be fair and representative. Therefore, to provide a deeper evaluation, we evaluated [82] using two different pairs of voltage and temperature bounds, i.e., B1 and B2. In general, these bounds will be different from one multicore system to another based on its specifications, i.e., the possible temperature and voltage ranges. The important rule of choosing the bounds is that they must fulfill the given aging constraint, and that can be checked using the aging model, e.g., when a higher temperature bound is chosen, a lower voltage bound must be accordingly selected to fulfill the aging constraint. Thus, to evaluate [82],two pairs of bounds are selected from the allowed temperature and voltage ranges of the adopted system model. The first pair B1 considers a temperature bound of 80 ∘C, and therefore the maximum voltage bound that fulfills the given aging constraint is 1.2 V, as shown in Figure 7.8. The second pair B2 considers a lower temperature bound of 60 ∘C, and therefore the maximum voltage bound is 1.25 V.

7.4.2 Results

For the experiments, three representative aging constraints are evaluated with varying V_{th}^{m} and lifetimes (see Figure 8.9). For each scenario in Figure 8.9, we present: the resulting overall system performance for the aging constraint (i) and (ii) in the bar chart of Figure 7.9[4]. This chart shows that our *AgRM* outperforms the *BoundRM* for all application scenarios with an average of 23% and 79% for B1 and B2, respectively. The maximum performance gain reaches up to 43% and 115% at scenario S11. For elaboration, the *amount of aging* is estimated after applying the resulting decisions of *AgRM* and *BoundRM* at scenario S11 and considering aging constraint (ii)(see Figure 7.10). As seen, the *amount of aging* resulting from *AgRM* decisions is *closer* to the aging constraint, but still fulfilling it. That is why *AgRM* is able to gain more performance than *BoundRM*, which is, by principle, too conservative. Additionally, *AgRM* adjusts *both* the number of threads and the V/f levels of the cores to exploit the entire design space. *BoundRM*, in contrast, adjusts *only* the V/f levels of the cores. That also elaborates why *CombRM* achieves higher performance compared to *BoundRM*; it is because our policy of determining the number of threads is employed within *CombRM*.

Most importantly, these application scenarios fully utilize the system (each consists of multiple applications, and each application can run up to 8 parallel threads). Aging matters for such scenarios most and is of lesser interest in underutilized system scenarios. But for completeness and discussion, we run 6 scenarios with only two application instances leading to utilizing only 25% of the system. In these scenarios, the average performance gain of *AgRM* compared to *BoundRM* is 7%. Explanation: In Figure 7.4 a smaller number of active cores results in lower temperature and therefore the design points are in the regions A and C. Whereas Region A can be exploited by both our *AgRM* and the state of the art, region C only by our *AgRM*, but that provides less benefit (but still some) compared to design points in region B. In summary it means *AgRM* are having higher benefits where it counts, i.e. highly utilized systems (that age faster) and lesser benefits where it hardly counts, i.e. underutilized systems (that age slower).

[4]The results for aging constraint (iii) are not shown due to brevity, but a summary of these results is shown in the table of Figure 7.9.



The performance gain of our *AgRM* compared to the average performance of *BoundRM* under various aging constraints:

(i) $\Delta V_{th}^m = 0.03V, \lambda = 1\ year$			(ii) $\Delta V_{th}^m = 0.05V, \lambda = 5\ years$			(iii) $\Delta V_{th}^m = 0.05V, \lambda = 10\ years$		
Avg	**Min**	**Max**	**Avg**	**Min**	**Max**	**Avg**	**Min**	**Max**
43%	27%	73%	46%	27%	72%	44%	27%	73%

Figure 7.9: Comparison between the resulting overall system performance by our *AgRM*, *BoundRM* [82], and *CombRM*, where the last two techniques are examined with under two different pairs of voltage and temperature B1 and B2. The two bar charts illustrate the results under two aging constraint (i) and (ii).

The last observation is the impact of adopting the two pairs of bounds (B1 and B2) by the state-of-the-art technique. In general, adopting B1 by both *BoundRM* and *CombRM* results in higher performance than adopting B2 because in B1 a higher temperature value can be reached. But when the workload is small, B2 might result in higher performance compared to B1, since a higher V_{dd} can be applied and that can be exploited when the workload results in a lower temperature than the temperature bound of B2. State-of-the-art technique *BoundRM* is, by principle, not able to apply more than one pair of bounds (either B1 or B2). *This demonstrates again that enforcing a predetermined pair of temperature and voltage bounds (to satisfy aging constraint) is a too conservative approach.*

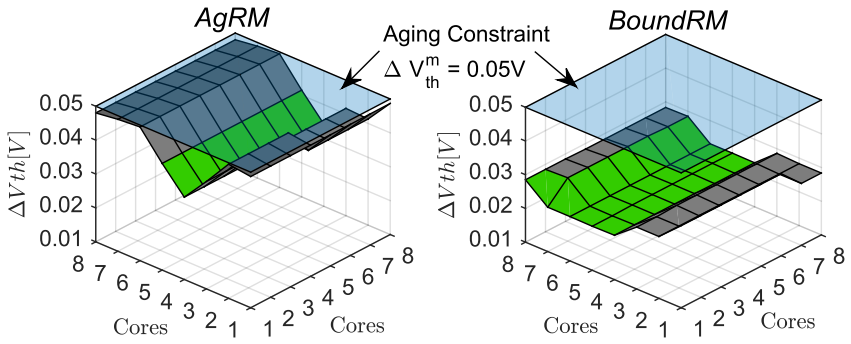


Figure 7.10: *Amount of aging*: Closer to the blue plane (constraint) means better exploitation leading to higher performance. An exemplary scenario showing why our *AgRM* is advantageous compared to the state-of-the-art technique.

Finally, the table shown in Figure 7.9 summarizes the average and the maximum performance gains of the presented *AgRM* for the relevant scenarios. In summary, the *AgRM* technique achieves an average of 43% compared to the average performance of *BoundRM*, with the identical aging constraint applied.

7.5 Summary

This chapter showed that the new aging-aware design space enables an aging-constrained resource management that is superior to the best known approach reported in the state of the art. The presented aging-aware design space depended on a precise physical models that have been validated against various batches of silicon dies. The experimental evaluation reported performance increase in comparison with the state-of-the-art technique from average of 43% to a maximum of 73% while fulfilling a set of aging constraints (The aging-constraints are varied to show the validity of the approach over various aging design goals). These results were possible since *AgRM* uses points in the design space that, by principle, cannot be exploited by the state of the art.

Chapter 8

Aging-Aware Boosting

The state-of-the-art technique that maximizes the performance at runtime without depending on any prior knowledge of the running applications is the DVFS-based boosting technique, which is widely adopted by several processor manufacturers like Intel Turbo Boost[1] [67, 68] and AMD Turbo Core [69]. In order to maximize the performance, boosting techniques allow processor cores to run faster than the base operating frequency. To keep the processor operating within safe margins, a simple closed-loop control system is employed to upscale the V/f level of the cores when the power and temperature are below specific limits, and downscale it otherwise. Boosting techniques have shown superiority in improving the performance [70], and therefore, processor manufacturers keep improving and issuing these techniques with the new generations of the processors [68]. However, *boosting techniques have negative impact on the processor aging*, since they stress the circuits and stimulate aging mechanisms. Particularly, boosting techniques increase long-term aging effects because they increase the V/f level and hence the supply voltage V_{dd}, which, in turn, increases the power density and thereby the temperature of the processor. As mentioned above, high voltage and temperature accelerate aging mechanisms. Furthermore, boosting techniques raise the so-called short-term aging effects (presented in Section 1.3) whenever they downscale the voltage level of the processor with the goal of satisfying the power and temperature constraints.

Therefore, to sustain reliability under boosting (i.e., to prevent any potential timing errors that would be incurred due to aging), the induced delays stemming from both *long-term* and *short-term* aging effects should be compensated for. That would necessitate employing a wider guardband than what would be dimensioned to solely compensate for long-term aging effects. However, a wider guardband leads to higher performance losses and thus hinders boosting techniques. The interdependencies between these partly contradictory constraints and goals are illustrated in Figure 8.1.

As a result, there is a pressing need to reduce the aging-induced delays and thereby the corresponding guardbands and their associated performance losses. The challenge here is that these aging effects originated due to the mechanism of the boosting technique that leads to frequent cycles of upscaling V/f levels to increase the performance and downscaling V/f levels to decrease the temperature. As discussed above, upscaling V_{dd} increases long-term aging, while downscaling V_{dd} causes short-term aging (see Figure 8.1).

[1] Intel Turbo Boost technique has been first implemented in Nehalem architecture in 2008 [67], but it is still adopted in recent Intel processors, like Intel Core i7 [68].

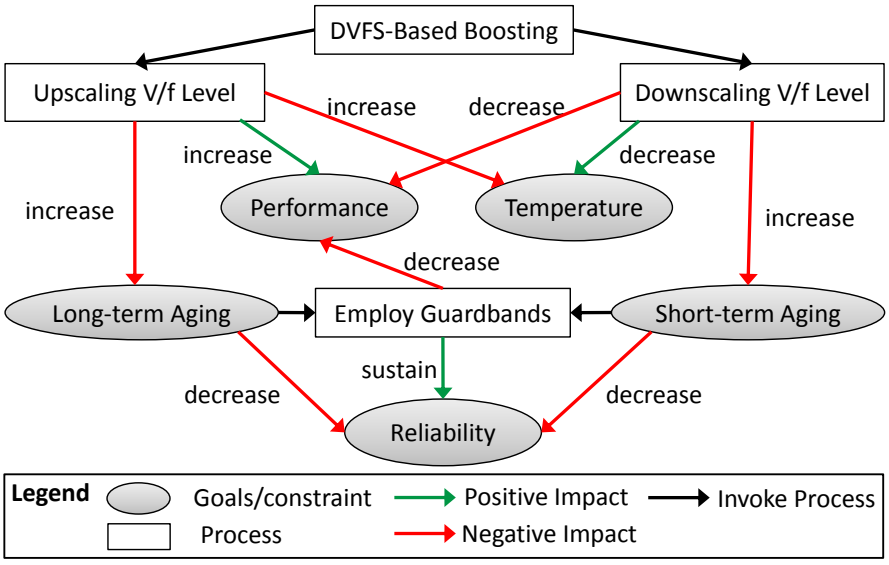


Figure 8.1: Illustrating the interdependencies between the partly contradictory constraints and goals of boosting.

Given this challenge, we present in this chapter a boosting technique that limits the stimuli of long and short-term aging effects, leading to reducing the delays induced by long and short-term aging effects, and thereby reducing the required guardbands to compensate for these delays. This way our proposed boosting technique is able to maximize the performance while at the same time reducing both long and short-term aging effects.

In the following section, a deeper insight into the well-known long-term aging effects and the recently-discovered short-term aging effects is provided. Section 8.2 presents an overview of the impact of employing guardbands on the *efficiency* of boosting, i.e., *its ability of maximizing the performance*. An analysis of the impact of the boosting technique on the aging effects is conducted in Section 8.3. Afterwards, Section 8.4 presents the proposed boosting technique. Finally, the evaluation of our proposed technique is demonstrated in Section 8.5.

8.1 Background of Long and Short-Term Aging Effects

As discussed in Chapter 1, long-term aging effects refer to the gradual increase in the threshold voltage (V_{th}) of transistors [43] over time. This increase in the threshold voltage, denoted as ΔV_{th}, leads to prolonging the critical path delay of the processor ($t_{critpath}$). However, the processor's clock delay (t_{clk}) is specified at design time according to the nominal specifications of the processor (i.e., V_{th} without any degradation). Hence, $t_{critpath}$ might become longer than t_{clk} resulting in timing violations and errors, as the following equations[2] show:

[2] As noted previously, only for the sake of simplicity, the equations in (8.1) use the simplified relation between the transistor delay and the drain current of transistor which does not consider the MOSFET short channel effects. However, the aging modeling in this dissertation employs a detailed modeling of I_d using the state-of-the-art industrial standard compact modeling for MOSFET (Berkeley Short-channel IGFET Model BSIM) [44, 45].

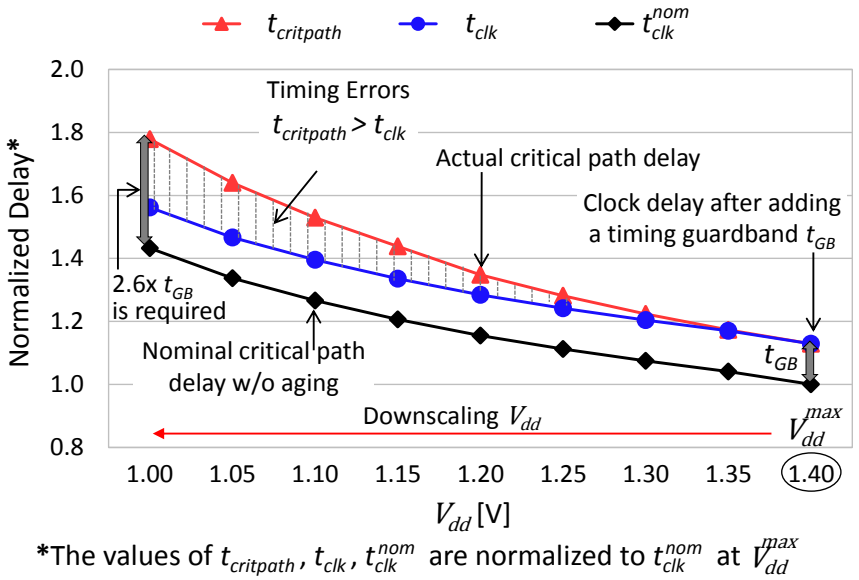


Figure 8.2: The BOOM processor has been analyzed in order to obtain its critical path delay. During V_{dd} downscaling caused by DVFS, the increase in $t_{critpath}$ is higher than the increase in t_{clk}, because the impact of ΔV_{th} on $t_{critpath}$ is magnified at lower V_{dd} values. The value of ΔV_{th} that caused by V_{dd}^{max} has been considered for a lifetime of 10 years.

Long-term aging effect:

$$
\begin{aligned}
t_{clk} = t_{clk}^{nom} &\propto \frac{1}{(V_{dd} - V_{th})} \\
t_{critpath} &\propto \frac{1}{(V_{dd} - V_{th} - \Delta V_{th})} \\
&\Rightarrow t_{critpath} > t_{clk} \Rightarrow \text{ Timing Errors} \\
\Delta t_L = t_{critpath} &- t_{clk} \text{ (at } V_{dd}^{max}) \qquad (8.1)
\end{aligned}
$$

Δt_L is the maximum increase in $t_{critpath}$ that results from the maximum value of ΔV_{th} that occurs at the maximum temperature (T_{crit}) and the maximum voltage (V_{dd}^{max}).

To avoid timing errors, it is necessary to add a timing guardband, referred to as t_{GB}, to the nominal clock delay t_{clk}^{nom}. t_{GB} must be equal to Δt_L to consider the maximum delay increase, as seen in Equation (8.2), in order to guarantee avoiding any timing errors that would be induced by long-term aging throughout the processor's lifetime.

$$t_{clk} = t_{clk}^{nom} + t_{GB}; t_{GB} = \Delta t_L \qquad (8.2)$$

This timing guardband results in a reduction in the operating frequency by a certain ratio, referred to as f_{GB}. Nevertheless, this guardband might not be sufficient to prevent timing errors when DVFS is applied, because recent measurements [46] and a research study [28] have reported a sudden increase in the critical path delay of circuit that temporarily arises when V_{dd} is downscaled.

To understand this phenomenon, the values of t_{clk}^{nom}, t_{clk}, and $t_{critpath}$ of the BOOM processor are analyzed using the employed aging modeling (details are in Section 3.4). Intuitively, all of these values (t_{clk}^{nom}, t_{clk}, $t_{critpath}$) will increase when V_{dd} is downscaled, as

seen in Figure 8.2. Before downscaling, i.e., $V_{dd} = 1.4\,\mathrm{V}$, the critical path delay $t_{critpath}$ is equal to t_{clk}, because of the added guardband t_{GB}. However, as can be noticed in Figure 8.2, when V_{dd} is downscaled, $t_{critpath}$ diverges from the corresponding t_{clk}, leading to timing errors.

The reason for this phenomenon is the combination between the lower V_{dd} values and the ΔV_{th} that is incurred at the previous higher V_{dd}. In practice, ΔV_{th} increases with increasing V_{dd}. However, when V_{dd} is downscaled, ΔV_{th} does not directly decrease, because the recovery process is much slower than V_{dd} switching process [28] which occurs in less than one microsecond in the presence of ultra-fast voltage regulators [129]. Consequently, the impact of the ΔV_{th} on $t_{critpath}$ (see Equation (8.1)) is magnified at smaller V_{dd} values, as seen in Figure 8.2. Thus, $t'_{critpath}$ at V'_{dd} is larger than t'_{clk} at V'_{dd}, as shown in Equation (8.3). When V_{dd} remains at lower levels for sufficient time (few milliseconds according to our model (Section 3.4)), ΔV_{th} decreases due to the partial recovery. Consequently, the temporal sudden increase in $t_{critpath}$ will diminish. This phenomenon of the sudden increase in $t_{critpath}$ that temporally arises due to V_{dd} downscaling is the so-called short-term aging effect.

Short-term aging effect:

$$
\begin{aligned}
&V_{dd} \rightarrow V'_{dd} \text{ such that } V'_{dd} < V_{dd} \\
&\text{Due to the impact of } \Delta V_{th} \text{ in Equation (8.1)} \\
&\text{and the fixed value of } t_{GB} : \\
&\Rightarrow t'_{critpath} > t'_{clk} \Rightarrow \text{ Timing Errors!} \\
&\Delta t_S = t'_{critpath} - t'_{clk} \text{ (at } V'_{dd}) \qquad (8.3)
\end{aligned}
$$

Δt_S is the delay increase in $t_{critpath}$ induced by short-term aging, when V_{dd} is downscaled to V'_{dd}. Thus, a guardband wider than t_{GB} is indispensably required.

To estimate the required guardband to compensate for short-term aging effects as well, it is necessary to extend the guardband estimation process explained in Section 3.5 to additionally consider the transitions between V_{dd} levels. As previously mentioned, Y aging-aware cell libraries need to be created, where each corresponds to a specific V_{dd} and the ΔV_{th} that results from the given V_{dd}. However, when V_{dd} is downscaled to V'_{dd}, V_{th} does not directly recover to the corresponding value of the new voltage, i.e., V'_{dd}. Therefore, for each voltage transition from V_{dd} to V'_{dd} we need to create an aging-aware library that corresponds to V'_{dd} and the V_{th} that results from the previous voltage value, i.e., V_{dd}. As a result, the total number of the aging-aware libraries will be YxY in order to consider all transitions between available V_{dd} values. The obtained guardbands are stored in a look-up table **GB** that will be used later by our technique.

It is noteworthy that while the research study of short-term aging at the system level is still preliminary, the research community at the physical and transistor levels has already focused on this new phenomenon in the last few years. The physics-based aging models [89], which describe in detail the underlying mechanisms of the defect generation under voltage dynamics, had predicted the magnified impact of aging-induced degradation when voltage is downscaled. Moreover, an evidence from silicon measurements has been presented in [46] demonstrating how the delay of circuits becomes higher than it should be when switching from a high to a low voltage due to the combined effect of aging-induced degradation at the high voltage together with higher susceptibility of circuits to aging at the lower voltage. The reason of such a high susceptibility to aging

at lower voltage can be seen in Equation (8.1), i.e., the smaller the V_{dd}, the larger the impact of ΔV_{th} becomes, and hence the higher susceptibility to aging.

8.2 Motivation

This section presents an experimental example that illustrates the impact of considering the guardbands on the efficiency of boosting techniques. In this example, we consider a 64-core chip (shown in Figure 4.4 simulated with gem5 [39] and McPAT [40] as illustrated in detail in Chapter 4. The application "x264" from PARSEC benchmark suite [41] is considered. We consider nominal voltage and frequency of 1.1 V and 2.8 GHz, respectively. The maximum voltage and frequency values are 1.4 V and 4.0 GHz, respectively. The critical temperature of the chip T_{crit} is set to 100 °C. A boosting technique is implemented similar to the state-of-the-art Intel boosting technique [67], that upscales the V/f levels of the cores when the peak temperature is below T_{crit}, and downscales it, otherwise. This process is repeated at each control period (i.e., 1 ms) during boosting.

The goal of this experiment is to evaluate the efficiency of boosting under three scenarios that vary in the way aging is considered. The first scenario does not consider any aging effects (i.e. neither long nor short-term). The second scenario considers *only* long-term aging effects. The last scenario considers *both* long and short-term aging effects. When aging is not considered (scenario 1), no timing guardbands with regard to aging are considered, and hence reliability is not insured, because timing violations might occur due to aging-induced delays. When aging is considered the frequency will be reduced with a certain ratio f_{GB}. Based on our aging estimation presented in Section 3.4, two values of guardband width are estimated; one compensates for the induced delay by long-term aging effects ($f_{GB} = 13\%$), while the second compensates for the induced delays by both long and short-term effects ($f_{GB} = 21\%$). That implies the reliability in the second scenario is also not ensured, because the delays induced by short-term aging effects are not compensated for. Only the last scenario sustains reliability. The performance is evaluated using the IPS of the running application. Figure 8.3 shows the resulting performance when boosting is applied normalized to the nominal performance under the nominal voltage and frequency where no boosting is applied. Intuitively, the wider guardband f_{GB} means lower operating frequency, and that leads to lower IPS, hence less performance. The results show that applying the necessary wider guardband (that considers both long and short-term aging effects) almost erases the performance gain that is expected to be achieved by boosting.

Thus, considering the required guardband to compensate the delays induced by long and short-term aging effects hinders the boosting technique from improving the performance.

8.3 Analyzing the Impact of Boosting on Aging

In this section, the impact of boosting on long and short-term aging effects is analyzed. Similar to the motivational example, we consider a 64-core chip simulated with gem5 [39] and McPAT [40] and executing an H.264 video encoder application. Further details of our experimental setup are discussed in Section 8.5. The maximum voltage and frequency values are 1.4 V and 4 GHz, respectively. The critical temperature of the

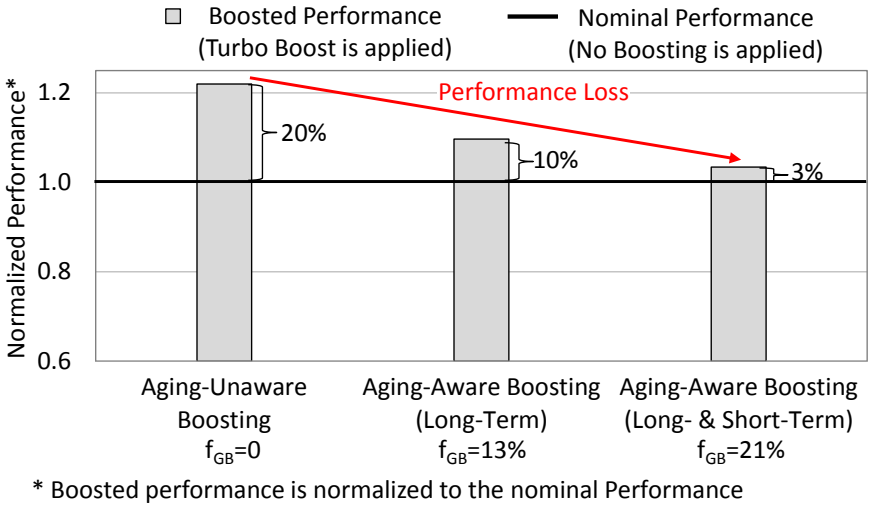


Figure 8.3: The resulting boosted performance under varied scenarios of considering aging effects.

chip is: T_{crit} = 100°C. The boosting technique upscales the V/f level when the peak temperature of the cores is below T_{crit}, and downscales it otherwise. This process is repeated at each control period (i.e., 1 ms) during the boosting time.

We examine in this experiment two scenarios "a" and "b" for the *maximum V_{dd} boosting level*, i.e., 1.4 V and 1.3 V, respectively, that lead to two different V_{dd}^{max}. Figure 8.4 first illustrates how the boosting technique employs DVFS over execution time. Secondly, we show the resulting aging-induced delays, which are derived by calculating the delays induced by long-term aging effects, Δt_L, and short-term aging effects, Δt_S. Δt_L is calculated using Equation (8.1), where V_{dd}^{max} is equal to the maximum boosting level of each scenario. Δt_S is calculated using Equation (8.3) for the V'_{dd} levels, to which the boosting technique downscales the voltage. As shown in Figure 8.4, when the boosting technique starts executing, it upscales the V/f level until reaching the maximum boosting level. It remains at this V/f level as long as T_{crit} is not exceeded. Once exceeded, for instance, at time t = 0.38 s in scenario "a" (see the first graph in Figure 8.4-a), the boosting technique starts downscaling the V/f level step by step until the maximum temperature becomes below T_{crit}. Again, it upscales the V/f level until reaching T_{crit}. Hence, it keeps scaling the V/f level up and down and oscillating around T_{crit}.

The aging-induced delays are shown in the third row of Figure 8.4. These curves demonstrate how long-term aging effects sustain throughout the execution time, while short-term aging effects only arise during V_{dd} downscaling. Additionally, short-term aging effects are influenced by the range between the maximum and the minimum levels that V_{dd} are scaling to. Particularly, the higher the range of V_{dd} downscaling, the higher the short-term aging effects are. By comparing the short-term aging effects at the time points t_1 = 0.45 s and t_2 = 0.55 s of scenario "a", it is noticed that 20% higher short-term effects are incurred at t_2, because the V_{dd} is downscaled to 1.15 V, while at t_1 the V_{dd} is downscaled only to 1.2 V. When scenario "a" is compared to scenario "b", it can be observed that reducing the maximum boosting V_{dd} level leads to reducing both long and short-term aging effects, but the performance gain of the boosting will be reduced as well. Namely, the resulting performance (without adopting guardbands) of scenario "a" and "b" is 22% and 16%, respectively. However, less aging effects require thinner

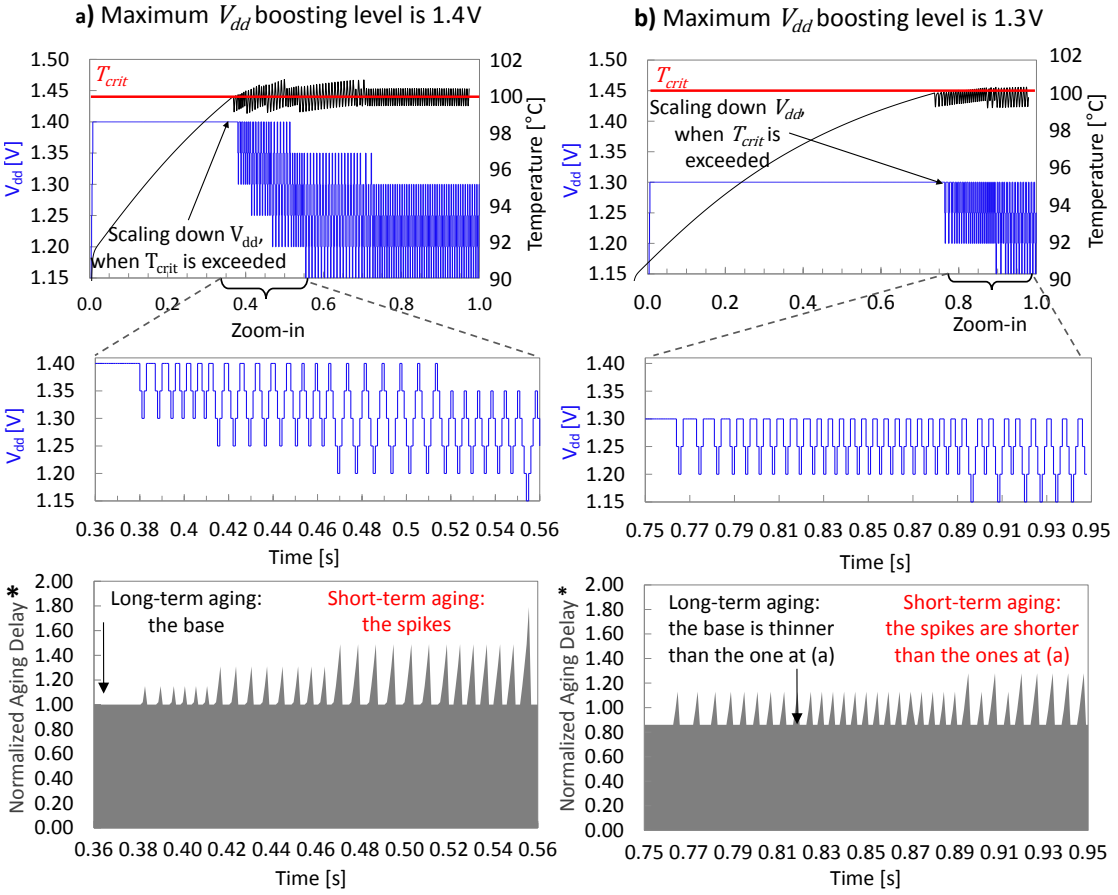


Figure 8.4: Analysis of the resulting long and short-term aging effects after applying a boosting technique under two maximum V_{dd} boosting levels. It demonstrates the factors that influence the aging effects, i.e., the occurrences of V_{dd} downscaling, the range between the maximum and the minimum levels that V_{dd} is scaled to, and the maximum V_{dd} boosting level.

guardbands and hence, the performance losses associated with the guardbands will be reduced.

Summary of the presented analysis:

- Boosting technique increases the delay induced by long-term aging, since it up-scales the V/f levels of the cores, thereby increasing their temperatures.
- Short-term aging effects frequently arise during boosting, due to the frequent cycles of V_{dd} downscaling.
- Short-term aging effects can be reduced by reducing the range between the maximum and the minimum boosting levels that V_{dd} is scaled to during boosting.
- The fewer occurrences of V_{dd} downscaling, the less often short-term aging effects arise over time.

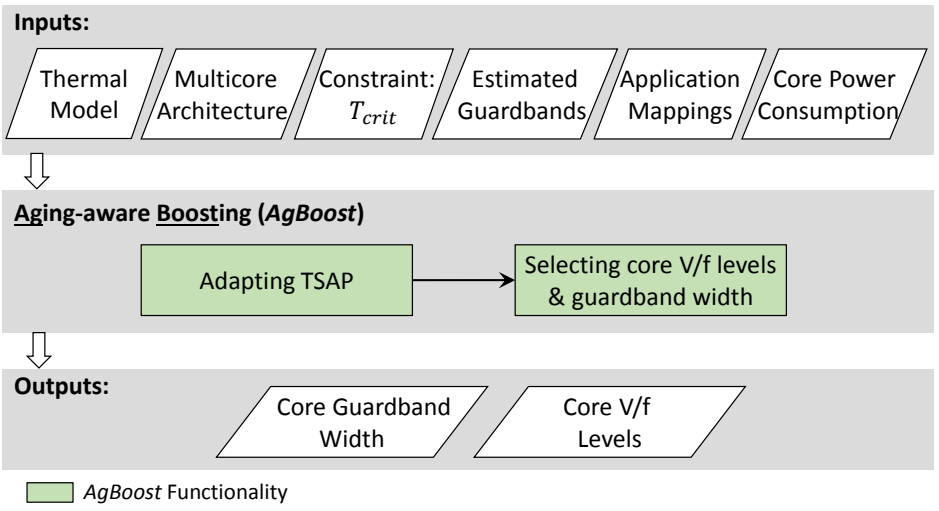


Figure 8.5: Overview of our Aging-Aware Boosting, *AgBoost*, that adjusts at runtime the V/f levels of the cores and the corresponding guardbands considering both long and short-term aging effects.

- Both long and short-term aging effects can be reduced by reducing the maximum V_{dd} boosting level.

8.4 Aging-Aware Boosting

According to the previous analysis, we propose an aging-aware boosting, *AgBoost*, that maximizes the performance while at the same time reducing both long and short-term aging effects. To achieve this goal, our technique reduces first the aging effects by employing a thermally-safe adaptive power constraint, that helps in decreasing the maximum V_{dd} boosting level (the main stimulus of long-term aging effects) and the occurrences of V_{dd} downscaling (the main stimulus of short-term aging effects). By reducing the aging effects, the required guardband to compensate for these effects will be reduced, as well. Secondly, our technique dynamically selects the required guardbands to compensate for the resulting aging-induced delays using a look-up table **GB** (built offline), in order to avoid unrequired performance losses incurred by static (pessimistic) guardbands. An overview of our technique is presented in Figure 8.5. The details about the design time and runtime processes that are required by *AgBoost* are shown in Figure 8.6.

8.4.1 Reducing Long and Short-Term Aging Effects

In order to reduce long and short-term aging effects, it is necessary to reduce three factors; the occurrences of V_{dd} downscaling, the range between the maximum and the minimum levels that V_{dd} is scaled to, and the maximum V_{dd} boosting level, as observed in the previous analysis. Additionally, we observe that the hidden player behind the first two factors is the temperature. That is, when the temperature exceeds T_{crit}, thermal violations occur, which, in turn, cause V_{dd} downscaling, and thereby, short-term aging effects arise. To return the temperature below T_{crit}, downscaling the V_{dd} for one level is not sufficient, rather V_{dd} needs to be downscaled multiple levels, inducing higher short-term aging effects. As a result, the first two factors can be reduced by mitigating thermal

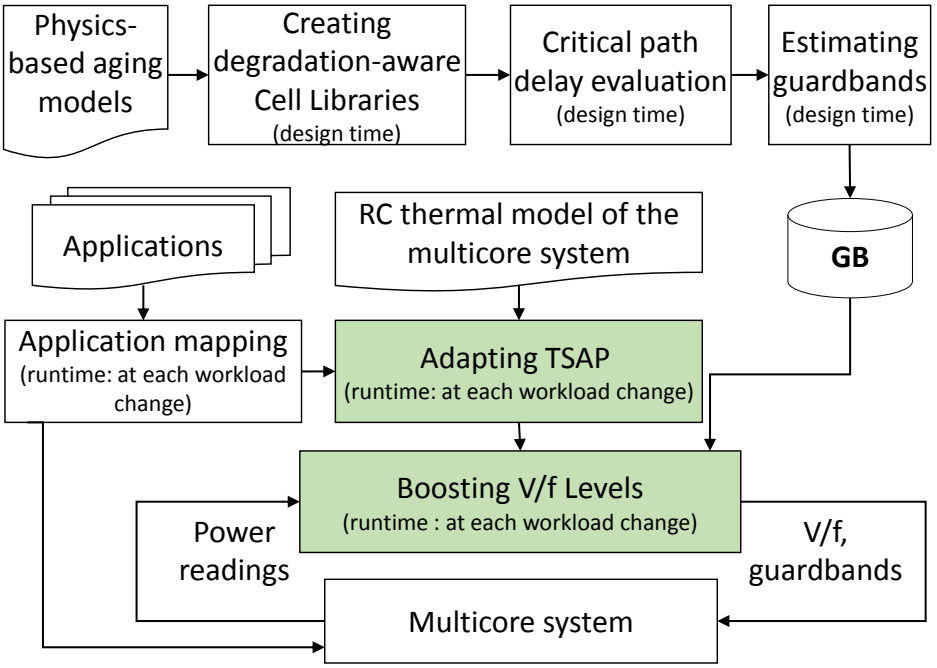


Figure 8.6: The design time and runtime processes that are required by *AgBoost*.

violations. Reducing the third factor, i.e., maximum V_{dd} boosting level, contradicts the goal of boosting, which upscales the V/f levels in order to maximize the performance. However, the obtained performance by upscaling the V/f levels might be erased if the higher V_{dd} leads to thermal violations and thereby V_{dd} downscaling, which in turn causes short-term aging effects that lead to high performance losses.

Solving this trade-off requires *finding the maximum V/f level that keeps the temperature below* T_{crit}. By revisiting state-of-the-art techniques we find that this requirement can be fulfilled by employing the thermally-safe power constraint (TSP) derived in [18], since it guarantees avoiding thermal violations before they occur. In particular, the boosting technique can upscale the V/f levels of the cores until reaching this constraint. However, this constraint is static and uniform for all cores, and does not consider the diversity of power consumptions between the cores that might be resulting due to running different applications on the chip at the same time. To elaborate on this point, we show in Figure 8.7 an example of applying a static thermally-safe power constraint (TSP [18]) on a 16-core chip, where three different applications are running on the cores. These are: "blackscholes" runs on four cores (Core_1, .., Core_4); "x264" running on the next four cores (Core_5, .., Core_8); "canneal" running on the rest of the cores. For $T_{crit} = 70°C$, TSP is equal to 7.7 W. The V/f level of each core is selected so that its power consumption remains below TSP. For the cores that execute "blackscholes" and "x264", the upscaling of the V/f levels is stopped before reaching the maximum V/f level available on the chip, in order to satisfy TSP constraint. Contrarily, the cores that run "canneal" reach the maximum V/f level ($f = 4\,\text{GHz}$). Nevertheless, the power consumptions of these cores are still far from TSP. As a result, there is still available power and thermal margins on the cores. To exploit them, the TSP constraint can be increased on the cores, where upscaling the V/f levels is limited due to the initial TSP constraint, as shown in Figure 8.7. This adaptation of TSP constraint allows higher V/f levels and thereby more

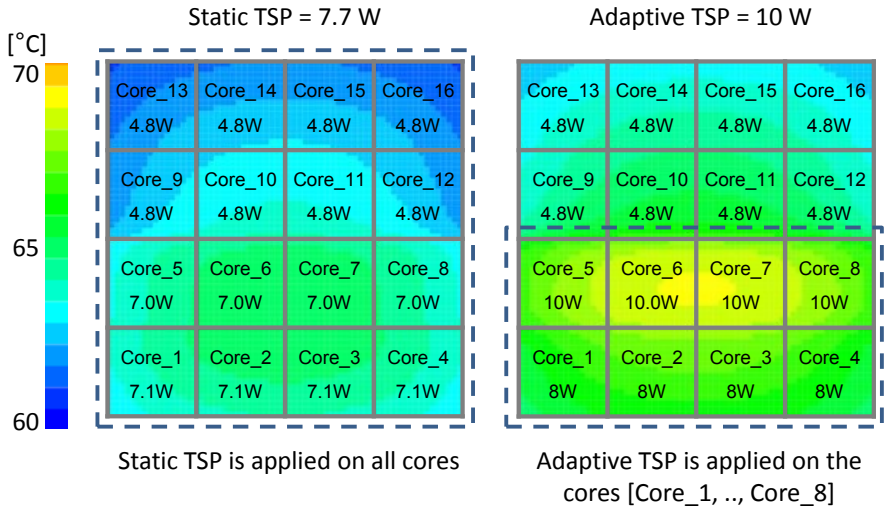


Figure 8.7: Illustrating how applying static TSP leads to unexploited thermal margin on some cores. Exploiting it can be achieved by adapting TSP constraint.

performance can be obtained. More importantly, this adaptation of TSP must still be thermally-safe.

Thus, Thermally-Safe Adaptive Power (TSAP) is derived by adapting the equations used in [9]. This requires employing an RC thermal network that represents the targeted chip. For this purpose, an RC thermal network is used as explained in Section 3.3. Then, the equation, that calculates the steady-state temperature of core i, i.e, Equation (3.6), is modified to differentiate between the cores that have available power and thermal margins and the ones that are able to exploit the available margins by calculating TSAP. Particularly, a column vector, $\mathbf{X} = [x_i]_{N\times 1}$ is defined, where $x_i = 1$, when the core i requires to calculate TSAP, and $x_i = 0$ otherwise. $\mathbf{X}$ can be involved into Equation (3.6) as follows:

$$T_i = \sum_{j=1}^{N} b_{i,j} \cdot x_j \cdot p_j + \sum_{j=1}^{N} b_{i,j} \cdot (1 - x_j) \cdot p_j + T_{amb} \cdot c_i \tag{8.4}$$

Then, we need to assume that all cores that require calculating TSAP ($x_i = 1$), consume the same power consumption P_{equal} and rewrite Equation (8.4) as follows:

$$T_i = P_{equal} \cdot \sum_{j=1}^{N} b_{i,j} \cdot x_j + \sum_{j=1}^{N} b_{i,j} \cdot (1 - x_j) \cdot p_j + T_{amb} \cdot c_i \tag{8.5}$$

To derive the power value of the cores that make the temperature of the core i equal to T_{crit}, T_i in Equation (8.5) is set to T_{crit}, as shown in the following equation:

$$P_{equal} = \frac{T_{crit} - \sum_{j=1}^{N} b_{i,j} \cdot (1 - x_j) \cdot p_j - T_{amb} \cdot c_i}{\sum_{j=1}^{N} b_{i,j} \cdot x_j} \tag{8.6}$$

This equation means, if the power consumptions of the cores with $x_i = 1$ are set to P_{equal}, the temperature of core i will be equal to T_{crit}. However, the temperature of the rest of the cores might be higher or lower than T_{crit}, even if these cores consume the same power P_{equal}. Therefore, to obtain a safe power constraint (TSAP), P_{equal} needs to be computed for all cores $i = 1, 2, \ldots, N$ and adopt the minimum computed power

value as our constraint TSAP.

$$\text{TSAP} = \min_{1 \leq i \leq N} \left\{ \frac{T_{crit} - \sum_{j=1}^{N} b_{i,j} \cdot p_j \cdot (1 - x_j) - T_{amb} \cdot c_i}{\sum_{j=1}^{N} b_{i,j} \cdot x_j} \right\} \tag{8.7}$$

Thus, this equation is used to calculate TSAP constraint for any core i with $x_i = 1$. The power consumptions of the other cores ($x_i = 0$) will be used in this equation, in order to exploit any available power and thermal margins on these cores, while calculating TSAP.

8.4.2 Minimizing Guardbands

Besides determining the V/f levels of the cores such that aging effects are reduced, our technique determines the minimum necessary guardbands to compensate for long and short-term aging effects incurred due to the determined V/f levels. For this purpose, we first conduct an offline analysis to estimate the necessary guardbands to overcome long and short-term aging effects for different V_{dd} scaling levels (as explained in Section 3.4). Based on the conducted analysis, we build a look-up table **GB** whose entries are represented by all possible transitions between available V_{dd} levels. For instance, the table cell (i, j) holds that V_{dd} is scaled from V_{dd}^i to V_{dd}^j. For upscaling, i.e., $V_{dd}^j > V_{dd}^i$, the cell will contain the guardband for long-term aging effect at V_{dd}^j. If $V_{dd}^j < V_{dd}^i$, the cell will contain the guardband to compensate for long and short-term aging effect, when V_{dd} is downscaled to V_{dd}^j. Our technique selects from this table the necessary guardband to compensate for the current delay induced by aging effects.

8.4.3 AgBoost Flow

Figure 8.8 illustrates the flow of our proposed boosting technique that employs TSAP to reduce the aging effects and selects the minimum necessary guardband to compensate for the aging-induced delays. *AgBoost* is implemented using a control loop, that runs at each specific control period and depends on reading current power values in making its decisions, similar to Intel Turbo Boost, which is implemented in modern processor chips [68]. Additionally, our *AgBoost* requires the matrices of the RC thermal network that represents the chip (see Equation (8.4), in order to calculate TSAP. The latter is not calculated at each control period, but calculated at each workload change. In particular, when the workload (set of running applications) changes, there is no guarantee that the obtained TSAP constraint for the previous workload will still be efficient (exploit available power and thermal margins) and thermally-safe. Therefore, TSAP needs to be adapted for all cores at each workload change. The initial values of vector $\mathbf{X}$ for the new workload are set to 1 for all active cores and to 0 for inactive ones. Afterwards, when the new workload starts running on the cores, the boosting technique can modify the vector $\mathbf{X}$ considering the diverse power consumptions on the cores resulting from running different applications. After modifying vector $\mathbf{X}$, TSAP is calculated again.

After calculating TSAP, the V/f levels of the cores need to be selected. Since the power consumptions of the cores are not known in advance, it is not possible to directly select

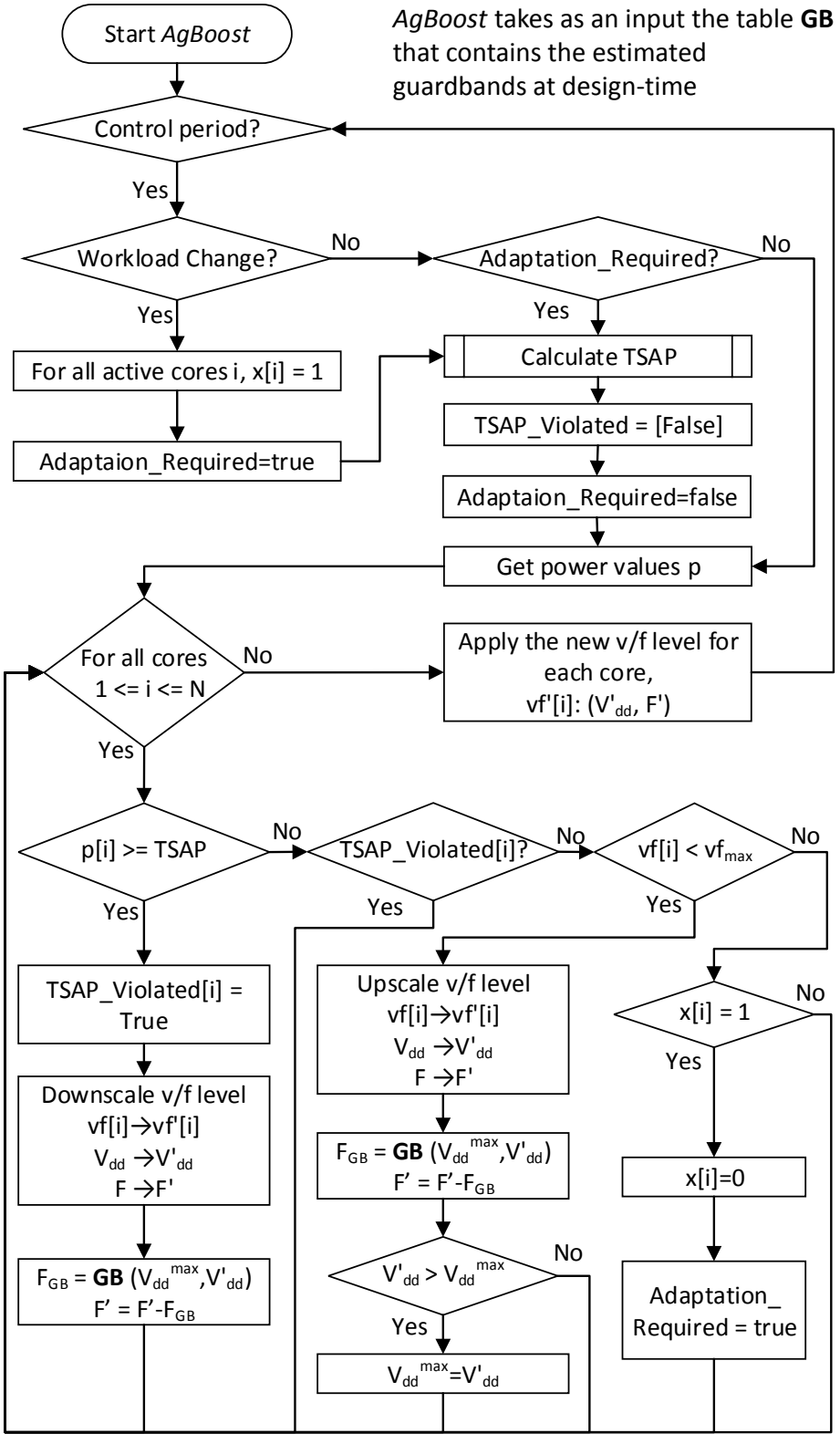


Figure 8.8: The flow chart of our Aging-Aware Boosting, *AgBoost*. The required guardbands to compensate for the delays induced by long and short-term aging effects are estimated at design time and stored in a table **GB** to be used by *AgBoost*.

the V/f levels of the cores that satisfy TSAP; rather, the V/f level of each core is upscaled step by step until reaching TSAP. Therefore, at each control period, *AgBoost* checks the current power consumptions of the cores (sampled from power sensors for example). If the power of the core is below TSAP, the V/f level is upscaled by one step. If TSAP is exceeded, that means the previous V/f level satisfies TSAP, and therefore, the V/f level is downscaled only by one step, and the core will remain at this level until a new TSAP is calculated. Hence, the maximum V_{dd} level of each core, i.e., V_{dd}^{max}, that it is possible to sustain on, is determined. Thus, the obtained performance by this V_{dd} level is sustained. After upscaling or downscaling the V_{dd} of the core to V'_{dd}, the corresponding required guardband is chosen from the look-up table **GB** according to the applied transitions of the V_{dd} levels, i.e., GB(V_{dd}^{max},V'_{dd}). It is to be noted that the parameters V_{dd}, V_{dd} and V_{dd}^{max} are local parameters for each core, because our target system enables DVFS per core (see the system model in Chapter Chapter 3).

As can be observed, *AgBoost* leads to downscaling V_{dd} one time whenever a new TSAP is calculated. As mentioned above, TSAP is calculated for each workload change and when adaptation is required to consider the power properties of the new workload. It is important to note that typically the boosting is meant to be applied for a short period of time and for a specific workload. Therefore, the workload changes should be limited during boosting. As a result, *AgBoost* significantly reduces the occurrences of downscaling V_{dd}, and also the range between the maximum and the minimum levels that V_{dd} is scaled to, so that the aging effects are reduced, while the performance is maximized.

8.5 Evaluation

In this section, the required experimental setup to evaluate our technique is explained. Afterwards, the conducted experiments and the results are illustrated. At the end of this section, the overhead of our technique is discussed.

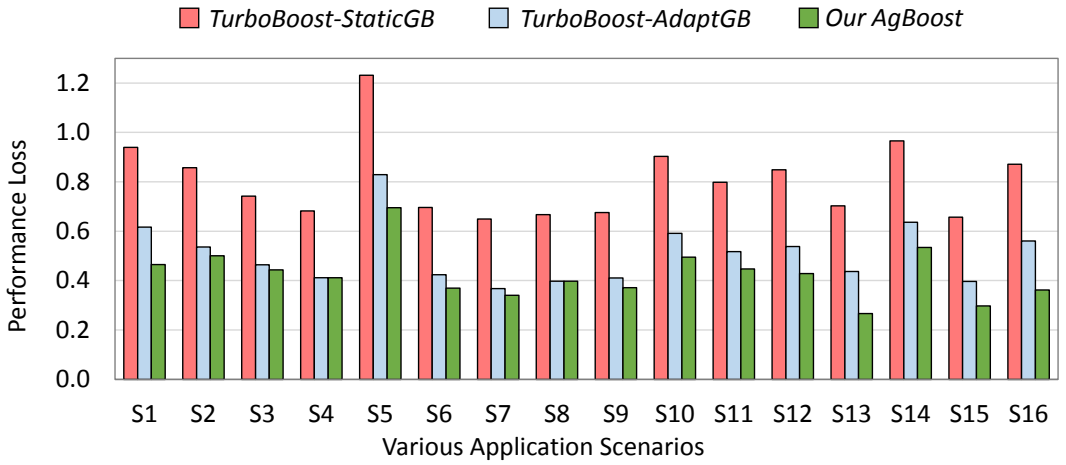
The targeted multicore system is a 64-core chip shown in Figure 4.4. The complete setup is explained in detail in Chapter 4, in which the tool flow is illustrated in Figure 4.1. Figure 8.9 lists the various application scenarios employed in our experiments. Each scenario of the first eight scenarios consists of multiple instances of the same application. The rest of scenarios represent workloads with multiple different applications.

8.5.1 Comparison Candidates

Since there is no other state-of-the-art boosting technique that considers aging, we compare our *AgBoost* with the state-of-the-art boosting, i.e., Turbo Boost [67], after combining it with two different guardbanding policies that consider both long and short-term aging effects. The first one employs a static guardband throughout the execution time. The size of this guardband must be equal to the maximum potential delay induced by short-term aging effect, i.e., 21%, according to our analysis, in order to avoid any timing errors. We refer to the combination of *TurboBoost* and the static guardband as, *TurboBoost-StaticGB*. The second guardbanding policy is the one proposed in [28] that

Single- Application Scenarios	S1	S2	S3	S4	S5	S6	S7	S8
x264	(8, 8)	-	-	-	(12, 4)	-	-	-
bodytrack	-	(8, 8)	-	-	-	(12, 4)	-	-
canneal	-	-	-	(8, 8)	-	-	-	(12, 4)
blackscholes	-	-	(8, 8)	-	-	-	(12, 4)	-
Multiple-Application Scenarios	**S9**	**S10**	**S11**	**S12**	**S13**	**S14**	**S15**	**S16**
x264	(4, 8)	(4, 8)	-	(4, 8)	(6, 4)	(6, 4)	-	(6, 4)
bodytrack	-	(4, 8)	(4, 8)	-	-	(6, 4)	(6, 4)	-
canneal	(4, 8)	-	-	-	(6, 4)	-	-	-
blackscholes	-	-	(4, 8)	(4, 8)	-	-	(6, 4)	(6, 4)

Figure 8.9: Table of application scenarios. Each cell contains the number of instances and the number of threads of each application in the scenario.



$$\text{Performance Gain (PG)} = \frac{\text{Performance (Boosting)} - \text{Performance (No Boosting)}}{\text{Performance (No Boosting)}}$$

$$\text{Performance Loss} = \frac{\text{PG (Aging-Unaware Boosting)} - \text{PG (Aging-Aware Boosting)}}{\text{PG (Aging-Unaware Boost)}}$$

Figure 8.10: Performance loss comparison between our *AgBoost*, *TurboBoost-StaticGB*, and *TurboBoost-AdaptGB*. It is to be noted that *TurboBoost-AdaptGB* is not an existing technique, but we merge two existing techniques, i.e.,[67], [28], for fair comparison. Performance loss reduction of *AgBoost* is 47%, 15%, on average, compared to *TurboBoost-StaticGB*, *TurboBoost-AdaptGB*, respectively.

adapts the guardband during execution time according to the current aging-induced delay. We refer to the combination of Turbo Boost and the adaptive guardbanding [28] as *TurboBoost-AdaptGB*.

8.5.2 Experimental Results

We conduct experiments for each technique under different applications with a different number of threads. Application mapping to the cores is considered as an input to all comparison candidates. Each technique determines at runtime the V/f levels of the cores and the required guardbands. The evaluation metric is the incurred performance loss

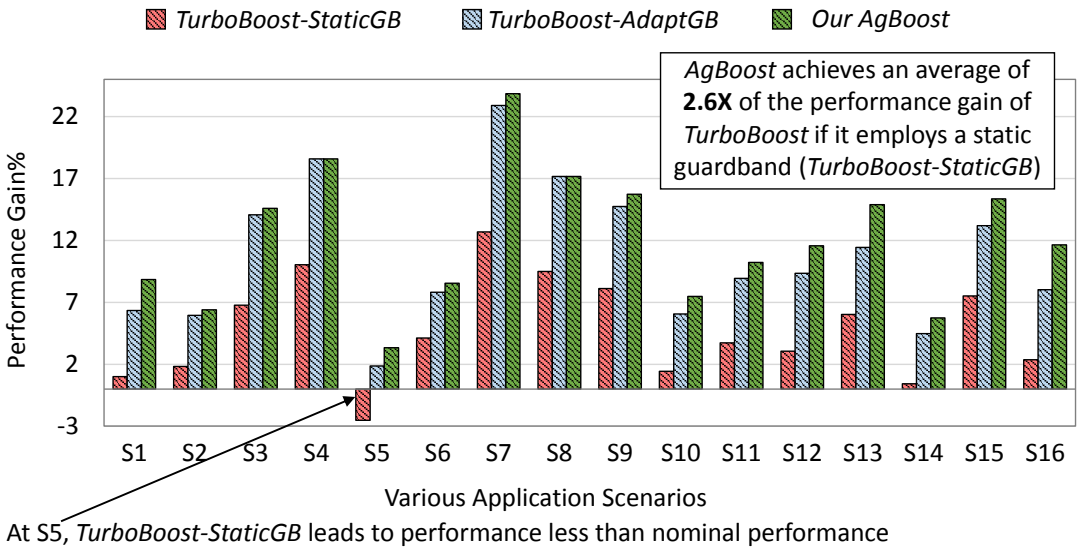


Figure 8.11: Performance gain of *AgBoost* is an average of 21% compared to *TurboBoost* if it employs an adaptive guardband.

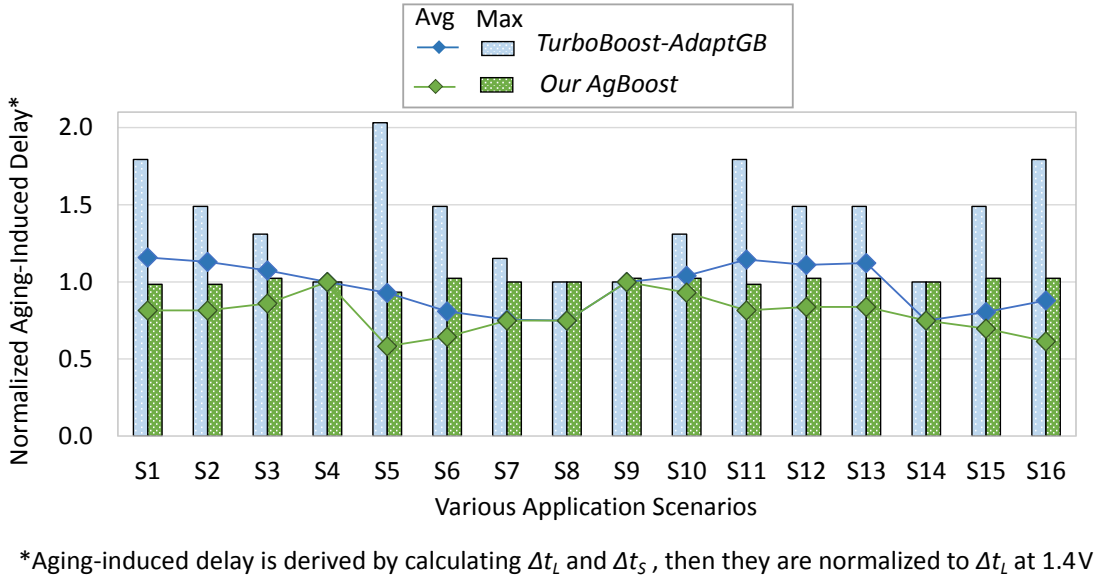


Figure 8.12: Comparing the resulting aging-induced delay by our *AgBoost* and *TurboBoost-AdaptGB*. *AgBoost* achieves up to 54% and an average of 27% reduction in the maximum aging-induced delay throughout the boosting execution compared to *TurboBoost-AdaptGB*.

from applying the determined guardband by each technique. The performance loss is calculated as shown in the equations in Figure 8.10.

The incurred performance losses of the evaluated boosting techniques are shown in Figure 8.10. As expected, applying a static guardband, *TurboBoost-StaticGB* almost erases the performance gain that the aging-unaware boosting can achieve. In particular, the performance loss in several scenarios of *TurboBoost-StaticGB* is near 1, which means it is almost equal to the performance gain. In contrast, applying adaptive guardbands by *TurboBoost-AdaptGB* reduces the performance losses, compared to applying a static

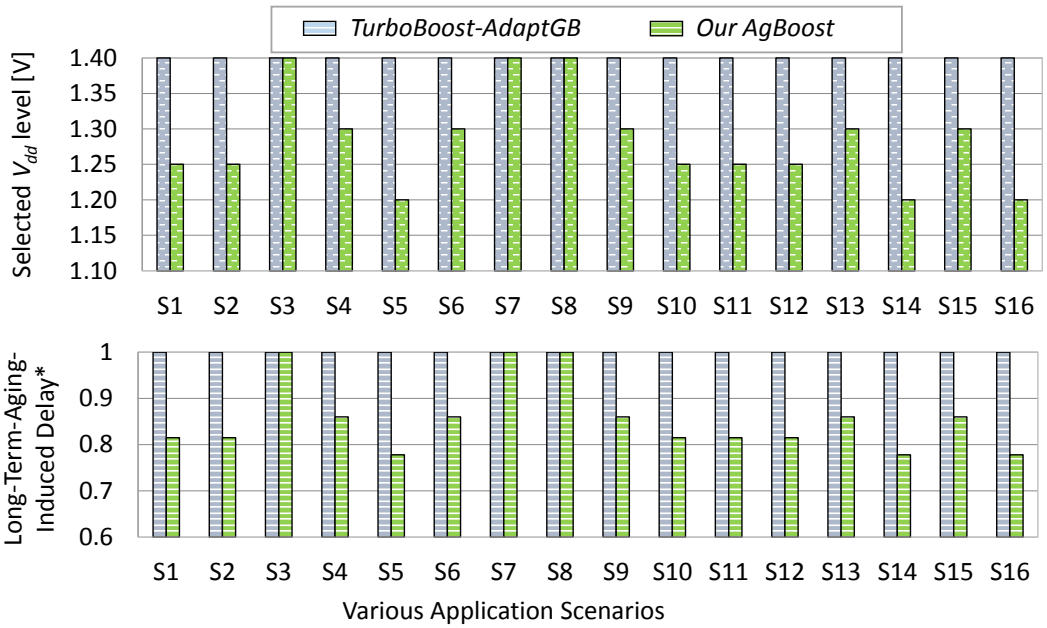


Figure 8.13: The resulting delay increase due to long-term aging effects when applying our *AgBoost* and the state-of-the-art *TurboBoost* [67].

guardband. While reducing the aging effects *and* applying adaptive guardbands achieve the most reduction in performance loss as the result of our *AgBoost* shows. In particular, *AgBoost* results in up to 62% and 39%, and an average of 47% and 15%, less performance loss compared to *TurboBoost-StaticGB*, *TurboBoost-AdaptGB*, respectively. The performance gain percentages compared to these techniques are shown in Figure 8.11.

Figure 8.12 shows the resulting delays induced by long and short-term aging effects when the evaluated boosting techniques are applied. It is to be noted that these resulting delays are normalized to the delay induced by long-term aging at the maximum available V_{dd}, i.e., 1.4 V. As it can be noticed from Figure 8.12, our *AgBoost* reduces aging-induced delay with an average of 27% compared to *TurboBoost-AdaptGB*. Moreover, it can be observed that the resulting normalized aging-induced delay by our technique is less than 1, which means it is less than the delay induced by only long-term aging effects that are induced at 1.4 V. This demonstrates how our *AgBoost* reduces both long and short-term aging effects. Additionally, Figure 8.13 illustrates the resulting delays induced by only long-term aging effects in order to show the ability of our technique to reduce long-term aging effects, since it selects lower V_{dd} compared to *TurboBoost-AdaptGB*.

Figure 8.14 demonstrates a more detailed comparison between *AgBoost* and *TurboBoost-AdaptGB*. When *AgBoost* starts execution, it calculates TSAP for the given application mapping, and finds the V_{dd} level that satisfies this TSAP value by upscaling V_{dd} one level per each control step, and at each step it checks if the resulting power consumption of the core reaches TSAP or not yet. Once the V_{dd} level that satisfies this TSAP is found, *AgBoost* remains at that level until the workload changes. Typically, boosting aims at maximizing the performance for a specific workload during a short period of time [67]. However, to evaluate our technique's ability to adapt to workload changes, if occurred, we change the workload of this scenario at two points of the execution time, namely $W1$ and $W2$, which represent the fifth and ninth seconds, respectively. At the first

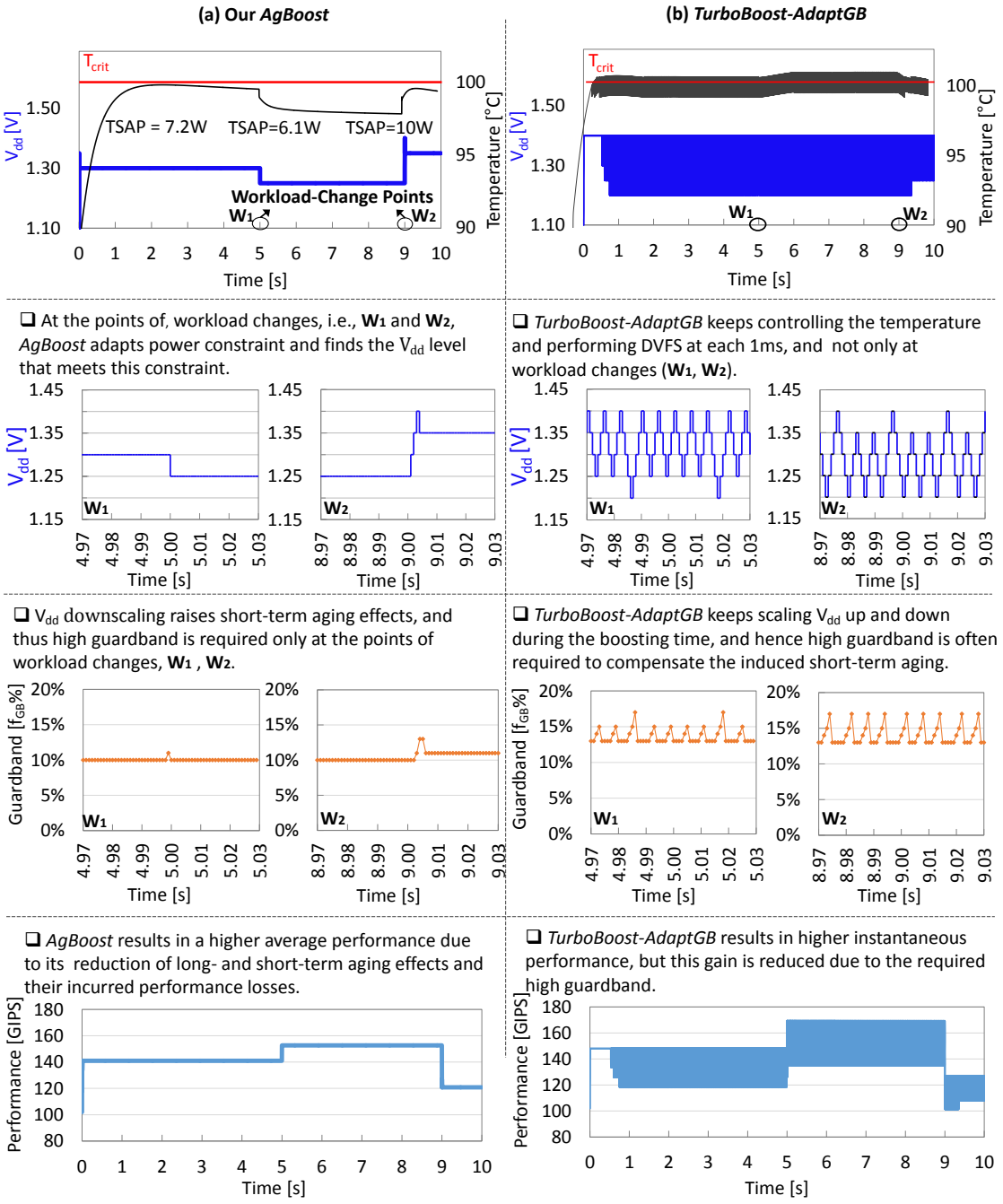


Figure 8.14: *AgBoost* minimizes long and short-term aging effects leading to 19% less performance loss compared to *TurboBoost-AdaptGB*.

point W_1 a new application arrives. Thus, our *AgBoost* calculates at this point a new TSAP value for the new application mapping. Then, it finds the V_{dd} level that satisfies this constraint. Evidently, when a new application arrives, the V_{dd} level is downscaled in order to avoid thermal violations. At the second point W_2 two applications finish execution, and hence *AgBoost* needs to find the V_{dd} level that satisfies the new TSAP value, which has a higher value than the old one, because less applications are running. As observed, *AgBoost* needs to downscale V_{dd} only when a new TSAP is calculated. That means short-term aging effects are rarely arisen, compared to the traditional boosting techniques that frequently downscale V_{dd} and thereby frequently cause short-term aging

effects. Additionally, the maximum V_{dd} level that satisfies TSAP is lower than the maximum available V_{dd} level and that, in turn, reduces the long-term aging effect as well. Consequently, the necessary guardband to compensate for the delays induced by aging effects is reduced as it can be noticed from Figure 8.14. Contrarily, *TurboBoost-AdaptGB* upscales V_{dd} to the maximum available V_{dd} level and downscales V_{dd} whenever T_{crit} is exceeded. That, in turn, leads to higher aging effects and thereby higher guardbands are required. Although *TurboBoost-AdaptGB* adapts the guardbands during execution time, *AgBoost* results in 19% less performance loss due to its ability of reducing the aging effects besides adapting the guardbands.

8.5.3 Overhead Discussion

The overhead of our proposed boosting technique *AgBoost* consists of two parts. The first part is the required time to calculate TSAP. The second part is the required time to execute the control loop over all cores, in order to determine the V/f level and the associated guardbands of the cores. The complexity of the second part is similar to our comparison candidate *TurboBoost-AdaptGB*, which represents the combination between the state-of-the-art Intel Turbo Boost [67] and the adaptive guardbanding [28]. Similar to our *AgBoost*, *TurboBoost-AdaptGB* determines the V/f levels and the associated guardbands for all cores. It can be noticed that the second part, which is similar to the state of the art, does not contain any internal search loop, it simply goes through all cores to assign their V/f levels and obtain the required guardbands from a look-up table **GB** as explained in 8.4.2. Therefore, we need to calculate only the overhead of the first part of *AgBoost*, i.e., calculating TSAP, which is the additional overhead that our technique has, compared to the state of the art, and the major contributor to the total overhead.

According to our experiments, the required time of calculating TSAP is 0.1 ms, where the experiments are conducted on a desktop PC (Intel core i5). As discussed above, TSAP will be calculated only at each workload change or when adaptation is required to consider diverse power properties of the applications that comprise the workload. Typically, boosting is meant to be for a specific workload that temporally demands a higher performance for a short period of time. That implies that it is not supposed to have workload changes during boosting period. With respect to the potential number of adaptation within a specific workload, the worst-case number of TSAP adaptation within a specific workload is equal to the number of applications that comprise the workload.

As a pessimistic example, we assume a workload consisting of 10 different applications, and this workload is changing every one second, because the approximate average execution time of parsec applications is about 1 s as reported in [130] for the smallest input set. It is to be noted that two parsec applications (i.e., “x264”, “canneal”) are executed on the same desktop PC, that executes the experiments, and similar average execution time is observed for the smallest input set. The overhead of calculating TSAP will be $10 * 0.1 = 1$ ms for each 1 s, and that means it represents only 0.1% of the boosting time. It is significant to note that TSAP in our experiments is calculated in software at application level. Implementing our technique in hardware similar to Intel Turbo Boost would even minimize its overhead further. More concisely, our *AgBoost* is a light-weight technique suitable to be applied at runtime.

8.6 Summary

This chapter presented *AgBoost* the first aging-aware boosting that reduces both long-term and short-term aging effects, while selecting the maximum allowed V/f level (governed by the thermally-safe adaptive power TSAP metric). We analyzed boosting in representative scenarios from where we designed our simple-to-apply, yet highly efficient boosting technique. Through several evaluations with varying parameters (like number of threads), we demonstrated a reduction of 47% (average) in performance loss that otherwise incurs if current state-of-the art boosting (like Intel Turbo Boost [67]) is deployed (which we enhanced with an aging-aware guardband for fair comparison). Moreover, *AgBoost* reduces the maximum aging-induced delay by 27% on average.

Chapter 9

Thermal-Aware Guardbanding

To sustain reliability, it needs to employ a guardband to compensate for the slowdown that arises as transistors age. Otherwise, errors due to timing violations occur during processor lifetime. There are two main *guardband types* that satisfy reliability, which are frequency guardband (F_GB) and voltage guardband (V_GB). State-of-the-art techniques select one of these guardband types at design time of the processor (circuit level) and adopt it throughout the processor's lifetime, based on the traditional view of the impact of the guardband type; that is F_GB causes a performance loss, whereas V_GB causes a power increase.

In this chapter we investigate the impact of the guardband type at the system level through the case study that will be presented in Section 9.1. This case study shows that V_GB is not necessarily able to sustain the performance even though it allows employing the maximum frequency without any reduction. This is because V_GB increases the power consumption which, in turn, may lead to triggering DTM to cool down the cores, leading to performance degradation. Hence, the performance would decrease contrary to what is expected. Therefore, traversing from the circuit level to the system level is necessary to be capable of considering the workload-induced temperatures and thereby accurately estimating the impact of the guardband type on performance. This implies that performance can be optimized through selecting the best fitting guardband type with regard to the workload.

To exploit this potential of performance maximization under reliability and temperature constraints, this chapter presents a thermal-aware guardbanding that determines at the system level the appropriate guardband type based on the estimated temperatures of the running workload. When the workload consists of multiple applications with diverse power properties, different guardband types might be selected for the cores that run different applications. Moreover, when the workload changes; a new application starts, or an application finishes its execution, the guardband types need to be selected again. In practice, these guardband types can be implemented similar to the implementation of different voltage and frequency levels. As a matter of fact, manufactures started implementing ultra-fast voltage regulator, in which switching between different voltage and frequency levels occur within less than 1 μs [129]. Hence, the overhead of switching between guardband types can be also considered as negligible.

In the following section, we present a case study showing that the ultimate impact of a guardband type can not be accurately estimated at the circuit level irrespective of the

workload at the system level. Section 9.2 presents the problem formulation. Afterwards, our presented technique is explained in Section 9.3. Finally, the evaluation results of our technique are demonstrated in Section 9.4.1.

9.1 Motivation

In this section, we conduct a case study that motivates *dynamic guardband selection at system level*. We consider a 64-core chip simulated by gem5 [39] and McPAT [40] (shown in Figure 4.4. Various applications from the PARSEC benchmark suite [41] are considered to cover different scenarios with respect to on-chip power and temperatures. We considered a nominal voltage V_{dd}^{nom} and a nominal frequency f^{nom} of 1.4 V and 4 GHz, respectively. The estimated V_{dd}^{gb} and f^{gb} according to our aging modeling (Section 3.4) are 1.491 V and 3.42 GHz, respectively. Our targeted multicore system features a thermal management unit (TMU), that is responsible for avoiding thermal violations by power-gating the cores that exceed the predefined critical temperature T_{crit}, which is set to 100°C.

Firstly, we show the impact of the guardband type on the power consumption and the performance for two different applications, i.e., "canneal" and "x264". For each application, we run two experiments to test V_GB and F_GB. In each experiment, 8 instances of the application are executed on the cores, with 8 parallel threads for each. The average power consumptions of the active cores and the resulting system performance are shown in Figure 9.1.

The power consumption will always increase when applying V_GB compared to F_GB, since V_GB leads to increase the value of the supply voltage as seen in Equation (1.3), thereby augmenting the power consumption of the core (see Equation (9.1).

$$P = \alpha \cdot C_{\text{eff}} \cdot V_{\text{dd}}^2 \cdot f + P_{\text{leak}}\left(V_{\text{dd}}, T\right) \tag{9.1}$$

There, α represents the activity factor or utilization of the core, C_{eff} represents the effective switching capacitance of the core. P_{leak} is the leakage power, which depends on the supply voltage and the core's temperature T. Considering the worst-case temperature on the chip (T_{crit}) and according to our simulation using McPAT, P_{leak} is equal to 3.5 W and 4.2 W for V_{dd}^{nom} and V_{dd}^{gb}, respectively. As an example, in the case of "x264", α and C_{eff} are equal to 0.3 and 2.2, respectively. To calculate the resulting power consumptions of applying F_GB and V_GB, we apply Equation (9.1) for (V_{dd}^{nom}, f^{gb}) and for (V_{dd}^{gb}, f^{nom}), respectively. That results in total power consumptions of 8 W and 10 W, as shown in Figure 9.1. As expected, V_GB results in higher power consumption compared to F_GB.

The resulting performance when applying V_GB is increased compared to F_GB in the case of "canneal". This observation is expected because F_GB leads to lower frequency (see Equation (1.2)) and thereby lower processor speed. However, in the case of "x264", applying V_GB reduces the performance by 24% compared to F_GB. This is because the higher increase in the power consumption in the case of "x264", when V_GB is employed, triggers the TMU which power-gates additional 14 cores whose estimated temperatures exceed T_{crit}. Consequently, the system performance is degraded in this case. This is unlike the case of "canneal" in which the increase in the power consumption due to employing V_GB does not lead to thermal violations.

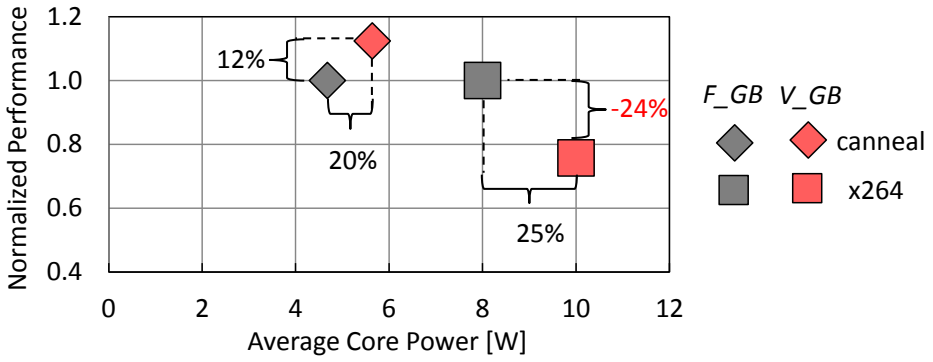


Figure 9.1: Even though V_GB allows operating at a higher frequency compared to F_GB, this does not always provide a better performance in a thermally-constrained system. For x264, employing V_GB leads to less performance, due to the incurred thermal violations.

Additionally, the impact of the guardband type on the performance is investigated under different *chip loads*. The *chip load* in the context of this work is the *percentage of cores demanded by the running workload to the total number of cores within the chip*. Therefore, we execute in parallel multiple instances of an application, in which every instance runs 8 threads, so that the total number of the executed application threads corresponds to the *chip load* that we aim to examine. For example, we simulate 8 instances of an application when a 100% *chip load* is required since the chip consists of 64 cores. Fig 8.3 summarizes our results for 5 different applications under two *chip load* scenarios. When the chip is fully loaded (Fig 8.3-a), only "canneal" gains a performance improvement (12%) from employing V_GB, while all other studied applications suffer from performance losses when V_GB is employed even though it allows a higher operating frequency than F_GB. This is because the increased power consumption due to V_GB by all applications (except "canneal") lead to triggering TMU which in turn power-gates the cores that exceed T_{crit}. When the *chip load* is reduced to 62% all applications except "x264" gain more performance when V_GB is employed, as shown in Fig 8.3-b. This is because when the *chip load* becomes less, the generated temperatures by the running workload become less and thus the possibility of having thermal violations when V_GB is employed becomes smaller. This, in turn, results in higher performance when V_GB is employed compared to F_GB.
It is to be noted that when the workload consists of different applications running simultaneously, exploring the impact of the guardband type becomes even more complicated. The reason is that the generated temperatures by one running application might affect the temperatures of the cores that run another application.

Summary of our motivational case study:

- V_GB is not always able to provide better performance than F_GB despite the higher operating frequency.
- The resulting system performance, when the guardband is applied, depends on the running application as well as the *chip load*.

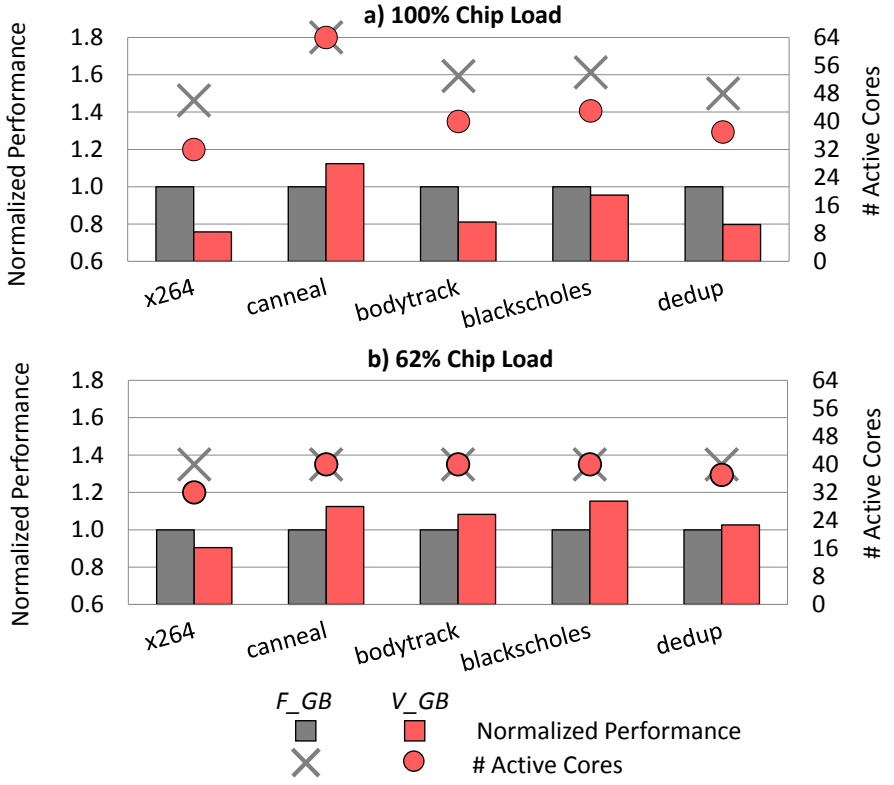


Figure 9.2: Motivational case study demonstrating the role of workload characteristics (*chip load* and application properties) on determining the impact of aging guardbands on performance. V_GB may not provide better performance than F_GB as assumed. The potential of gaining performance when applying V_GB diminishes while the *chip load* increases.

9.2 Problem Formulation

The purpose of this chapter is to explore the performance optimization potentials of selecting the guardband types at the system level, apart from any other optimization means like DVFS. Therefore, in this chapter we assume that all active cores run on the maximum V/f level, i.e., VF_Y. Each core i is associated with a specific guardband type, referred to asGt_i^{core}, which can be set to V_GB or F_GB at a specific point of time. When any guardband type is applied on the core, its voltage or frequency will be changed. We refer to the original voltage and frequency of the core before applying the guardband as V_{dd}^{nom} and f^{nom}, respectively. When Gt_i^{core} is set to V_GB, the applied voltage and frequency pair is (V_{dd}^{gb}, f^{nom}). Otherwise, i.e., if Gt_i^{core} is set to F_GB, (V_{dd}^{nom}, f^{gb}) is applied. This new applied voltage and frequency pair is denoted as VF'. It is to be noted that the same guardband type must be assigned to the cores of one application, because these cores must run at the same speed (frequency) to avoid any synchronization stall between the application threads. We refer to the guardband type of the cores that run application k, i.e., map_k, as Gt_k^{app}.

Since the selected guardband type affects the applied voltage and frequency values of the cores, it consequently affects the power and performance of the applications. As seen in Chapter 3, the power table P_k stores the resulting power consumptions when running application k at all possible number of threads and V/f levels. In this chapter, one V/f level is considered (VF_Y) and two guardband types. Therefore, we store in $P_k(h, Gt)$

the power consumptions for all possible number of threads ($h = 1 \dots H_k$) and for the two guardband types ($Gt \in V\_GB, F\_GB$). Similarly, $R_k(h, Gt)$ stores the application performance values for all possible number of threads and guardband types, which is estimated using IPS metric as follows:

$$R_k(h, Gt) = \frac{IPS_k(h, VF'_Y)}{IPS_k(H_k, VF_Y)} \in [0, 1] \tag{9.2}$$

As revealed by the motivation example, the resulting system performance of applying guardband types V_GB and F_GB depends on the temperatures generated by the workload. Therefore, we propose to dynamically selects the appropriate guardband type according to the running workload, with the **objective** of maximizing the overall system performance under reliability and temperature constraints. The overall system performance is the summation of the performance of all running applications at their current settings as Equation (9.3) shows:

$$System\ Performance = \sum_{k=1}^{K} R_k(h_k, Gt_k) \tag{9.3}$$

Selecting the appropriate guardband types according to the running workload implies that different guardband types might be selected over time if the workload changes. Moreover, when the workload consists of multiple applications that are mapped to the chip at the same time, different guardband types might be selected for the cores that execute different applications, because these applications might have diverse power properties, thereby leading to different temperatures on the cores. Thus, for each application k, we need to select the appropriate guardband type Gt_k^{app} for the cores that execute it, i.e., map_k. Besides selecting the guardband type, there is a need to adjust the power states of the cores in order to proactively avoid thermal violations. It is to be noted when some cores that run an application are power-gated, the number of threads of that application h_k will be reduced. Hence, the problem can be formulated as follows:

For a workload of K applications mapped to N cores, it is required to select for each application k: the guardband type Gt_k^{app} of the cores in map_k, the number of threads h_k, and the power-states S_i of the cores in map_k, in order to maximize the overall system performance (Equation (9.3)) while the steady-state temperatures (Equation (8.4)) of the cores are kept below the critical temperature T_{crit}. The reliability constraint is satisfied through applying the required guardbands to prevent timing violations, as Equation (1.2) and Equation (1.3) show.

This problem can be formulated as a Multiple-Choice Knapsack Problem (MCKP) [131], in which several items need to be packed to a knapsack of a certain capacity, where each item has multiple configurations. Each configuration has a profit and a weight. The problem is how to choose one configuration from each item so that the profit sum is maximized without having the weight sum to exceed the knapsack capacity. The knapsack in our problem is the chip, where its capacity is the available thermal budget (TB) on the chip, which is estimated as: $T_{crit} - T_{max}$, where T_{max} is the maximum steady-state temperature among the cores. The items are the K applications, where each application has multiple configurations of the number of threads and the guardband type, as shown in the application model. Each configuration (h, Gt) has a profit, i.e., application performance (R_k), and a weight ΔT_k^{app}, which is the temperature increase

on the cores caused by executing the application. For a given number of threads h and a given guardband type Gt, ΔT_k^{app} is calculated by employing Equation (3.7) and considering that the core power before mapping application k is zero as the following equation shows:

$$\Delta T_k^{app}(h, Gt) = [\Delta T_i]_{N\times 1} \text{ where } \Delta T_i = \sum_{\ell=1}^{H_k} (b_{i,map_k(\ell)} \cdot (P_k(h, Gt) \cdot S_{map_k}) \tag{9.4}$$

The summation of the weights means in this problem the total temperature increase on the chip cores resulting from executing K applications. Thus, the knapsack capacity will be satisfied if the maximum temperature increase of the resulting weight sum is within the available thermal budget. Mathematically, this problem can be expressed as follows:

$$\begin{aligned}
&\text{Select } Gt_k, h_k \text{ for all } k = 1, 2, \ldots, K \text{ and} \\
&S_i \text{ for each } i \in map_k, \text{ in order to:} \\
&\text{Maximize } \sum_{k=1}^{K} R_k(h_k, Gt_k) \text{ , subject to:} \\
&Max(\sum_{k=1}^{K} \Delta T_k^{app}(h_k, Gt_k)) < TB
\end{aligned}$$

This problem is NP-hard problem [131], and solving it through an exhaustive search results in an exponential time complexity.

9.3 Thermal-Aware Guardbanding

Knapsack problems can be solved in polynomial time using dynamic programming, because dynamic programming reduces the number of combinations that need to be examined. In particular, dynamic programming breaks down the problem into smaller sub-problems. Then, it solves the sub-problems optimally and builds up the final solution gradually based on the found sub-solutions. Storing the obtained sub-solutions at every step in tracking tables enables dynamic programming algorithms to reuse these sub-solutions in further steps. Hence, the combinations required to test are much less compared to exhaustive search. Nevertheless, dynamic programming algorithms cannot be considered as light-weight solutions, and their time complexity needs to be taken into account. We propose a dynamic programming algorithm referred to as *sGuard-DP* in order to solve the defined problem for static workloads.

Additionally, we propose a heuristic referred to as *sGuard-H*, which has a less time complexity than *sGuard-DP* (as it will be evaluated in Section 9.4), to be used for dynamic workloads. The proposed heuristic *sGuard-H* might not reach the maximum performance that *sGuard-DP* reaches, because it greedily tests some combinations of the feasible solutions. An overview of the proposed algorithms for our thermal-aware guardbanding is shown in Figure 9.3.

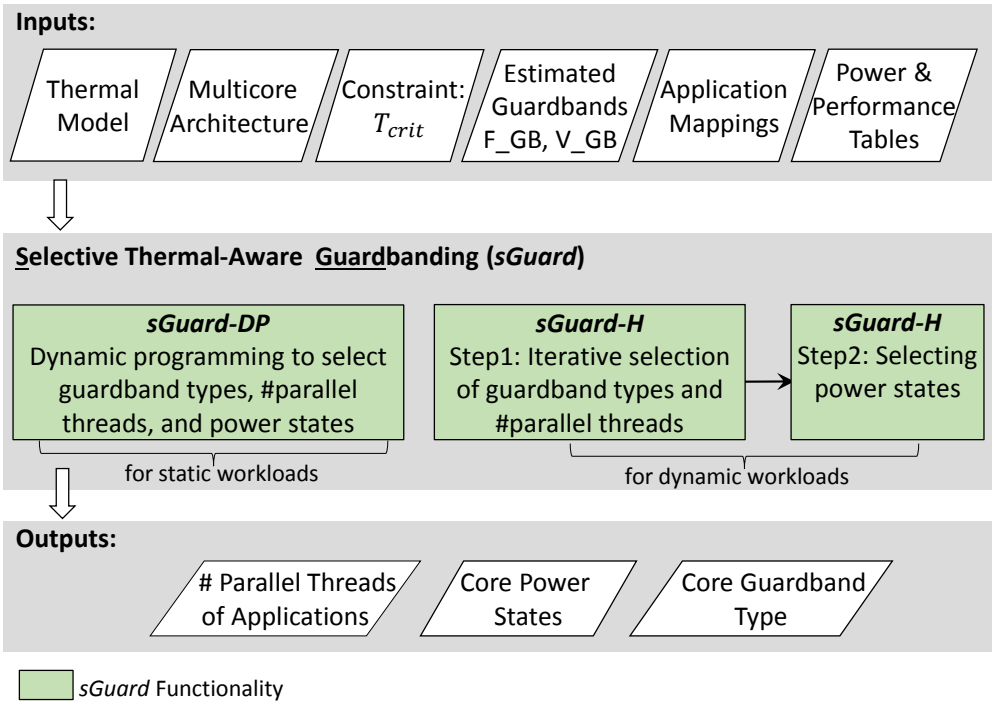


Figure 9.3: Overview of the proposed thermal-aware guardbanding *sGuard*, where two approaches are proposed, i.e.,*sGuard-DP* and *sGuard-H*, to be applied for static workloads and dynamic workloads, respectively.

9.3.1 Dynamic Programming-Based Thermal-Aware Guardbanding

The inputs of the algorithm are: the available thermal budget TB (i.e., $T_{crit} - T_{max}$), the set of K applications with their power and performance models, (P_k, R_k), as well as their mapping sets (map_k). The outputs of the algorithm are: the guardband type Gt_k^{app} and the number of threads h_k of each application k as well as the power states of all cores, so that the overall performance is maximized under the predefined critical temperature. To break down the problem, the proposed algorithm divides the available thermal budget on the chip to smaller thermal budgets (portions), $tb_1, tb_2, \cdots, tb_m$, where m is the number of the portions and $tb_m = TB$.

To obtain the sub-solutions, our algorithm uses an auxiliary function, $opt(k, tb)$, defined in Algorithm 9, that finds the optimal settings of application k under a given thermal budget tb. More precisely, it finds the guardband type and the number of threads of application k that maximizes the application performance without leading to an increase in temperature more than the thermal budget tb. To achieve that, $opt(k, tb)$ employs a linear search on the P_k table to find the entry (Gt, h) that results in a change in temperature less than tb. It is to be noted that the application performance stored in R_k increases with increasing the number of threads, and also it will be higher when V_GB is selected rather than F_GB. Thus, it can be considered that the table P_k is sorted in ascending order according to the application performance. Therefore, the index of the search in $opt(k, tb)$ starts from the entry that holds the maximum number of threads H_k rather than starting from the first row. Moreover, V_GB is checked before F_GB, as the former leads to a higher performance.

Algorithm 9 Find the optimal settings for application k that maximize its performance under thermal budget tb (the auxiliary function $opt(k, tb)$)

Input: Power and performance tables P_k and R_k (Section 3.2);
Output: h_k, Gt_k^{app};

```
1: for h = H_k to 1 do
2:   if Max(T_k^app(P_k(h, V_GB)) < tb then
3:     return h, V_GB
4:   else if Max(T_k^app(P_k(h, F_GB)) < tb then
5:     if Max(T_k^app(P_k(h − 1, V_GB)) < tb and R_k(h − 1, V_GB) > R_k(h, F_GB) then
6:       return h − 1, V_GB
7:     else
8:       return h, F_GB
9:     end if
10:  end if
11: end for
12: return ∅;
```

Starting from the entry (H_k, V_GB), $opt(k, tb)$ checks the maximum increase in temperature, $Max(T_k^{app}(P_k(H_k, V\_GB))$, that would be incurred if application k runs the maximum number of threads H_k (default mapping) and the guardband type of its cores is set to V_GB (see Equation (9.4)). For this entry the power states of all cores in map_k will be equal to 1, because the application runs the maximum number of threads H_k. If $Max(T_k^{app}(H_k, V\_GB))$ is equal or less than tb, $opt(k, tb)$ returns the values of (H_k, V_GB). Otherwise, $opt(k, tb)$ checks both $Max(T_k^{app}(H_k, F\_GB)$ and $Max(T_k^{app}(H_k - 1, V\_GB))$ in order to select the one that achieves a higher performance within tb. If none of them was within tb, $opt(k, tb)$ checks the next pair, i.e., $Max(T_k^{app}(H_k - 1, F\_GB))$ and $Max(T_k^{app}(H_k - 2, V\_GB))$, and so on until reaching the last entry $(1, F\_GB)$. If the search is finished and no feasible settings within the given tb are found, the function returns $\varnothing$.

During the search, when $opt(k, tb)$ examines the maximum temperature increase $Max(T_k^{app})$ for number of threads h less than H_k (in case running H_k threads violated tb), it needs to power-gate some cores in map_k so that the number of active cores will be equal to h. Since map_k is sorted in ascending order according to their susceptibility to temperature increase, $opt(k, tb)$ will set the power state of the last $H_k - h$ cores in map_k to 0, because they will be the cores that would violate this budget.

Our dynamic programming algorithm builds two tables, DP, DP^T. The entries of these tables are represented by all possible unique entries (k, tb). Each cell of the first table, $DP(k, tb)$, contains the maximum performance for the first k applications (assuming an arbitrary order for the applications) so that the maximum temperature increase resulting from mapping these applications is equal or less than tb. The corresponding temperature increase on the cores resulting from mapping the first k applications is stored as a vector in the cell $DP^T(k, tb)$, of the second table. Our algorithm starts from the first application $k = 1$ and calculates its maximum performance at all available thermal budgets tb, as shown in Equation (9.5), which is the initial boundary condition of building DP table.

$$DP\,(1, tb) = \begin{cases} 0 & \text{if } opt(1, tb) = \varnothing \\ R_1(opt(1, tb)\_h, opt(1, tb)\_Gt) & \text{otherwise} \end{cases} \tag{9.5}$$

Simultaneously, $DP^T(1, tb)$ is initialized as follows:

$$DP^T(1, tb) = \begin{cases} [0] & \text{if } opt(1, tb) = \varnothing \\ T_1^{app}(opt(1, tb)\_h, opt(1, tb)\_Gt) & \text{otherwise} \end{cases} \tag{9.6}$$

To calculate $DP(k, tb)$ for $(k > 1)$ and $(k <= K)$, the following recursive function is used:

$$DP(k, tb) = \max \begin{cases} DP\left(k-1, tb'\right) & \text{if } opt(k, tb'') = \varnothing \\ 0 & \text{if } Max(DP^T(k-1, tb') \\ & +T_k^{app}(opt(k, tb'')\_h, opt(k, tb'')\_Gt)) > tb \\ DP\left(k-1, tb'\right) + & \\ R_k\left(opt(k, tb'')\_h, opt(k, tb'')\_Gt\right) & \text{otherwise} \end{cases} \tag{9.7}$$

$$\text{where } 0 \leq tb' \leq tb \text{ , } 0 \leq tb'' \leq tb \mid tb' + tb'' \geq tb$$

Using Equation (9.7), our algorithm finds the maximum performance out of all the possible combinations of assigning different portions tb'', tb' from the given thermal budget tb to application k and the previous $k-1$ applications. tb'' and tb' will be set to values from the range $0, tb_1, tb_2, \ldots, tb$. To avoid re-calculating combinations that have been examined for the previous thermal budgets that are less than tb, we added the condition $tb' + tb'' \geq tb$ that must be satisfied in order to examine the combination tb', tb''.

For each combination (tb', tb''), we assign to application k the thermal budget tb'' and call the auxiliary function $opt(k, tb'')$. If there are no feasible settings of application k under tb'', the performance of this combination is set to the performance of the previous $k-1$ applications at tb', i.e., $DP(k-1, tb')$. If there are feasible settings of application k under tb'', we check the sum of two vectors of temperature increase; the first one induced by application k at tb'' and calculated using Equation (9.4), while the second one is induced by previous $k-1$ applications, which is already stored in $DP^T(k-1, tb')$. If the resulting sum does not contain a temperature increase larger than tb, the combination is accepted and the resulting performance of this combination will be the sum of the performance of the previous $k-1$ applications and the resulting performance of application k at the thermal budget tb''.

Once the the maximum performance is found out of these combinations, it is stored in $DP(k, tb)$ which represents the maximum performance of the first k applications under the thermal budget tb. The corresponding sum of the temperature increase vectors is stored in $DP^T(k, tb)$, which represents the temperature increase induced by the first k applications where their performance is maximized under tb, as Equation (9.8) shows.

$$DP^T(k,tb) = \begin{cases} DP^T\left(k-1,tb'\right) & \text{if } opt(k,tb'') = \varnothing \\ [0] & \text{if } Max(DP^T(k-1,tb') \\ & +T_k^{app}(opt(k,tb'')\_h, opt(k,tb'')\_Gt)) > tb \\ DP^T\left(k-1,tb'\right) + & \\ T_k^{app}\left(opt(k,tb'')\_h, opt(k,tb'')\_Gt\right) & \text{otherwise} \end{cases} \tag{9.8}$$

There, tb' and tb'' are the values that result in the maximum performance that is stored in $DP(k,tb)$ as seen in Equation (9.7). Additionally, these values (tb'' and tb') are stored in a tracking table called $Track\_tb$. Thus, in the cell $Track\_tb(k,tb)$ both values tb'' and tb' are stored to track the thermal budgets assigned to the current application k, and to the previous $k-1$ applications, respectively. Furthermore, the values $opt(k,tb'')\_h, opt(k,tb'')\_Gt$ that result in the maximum performance of application k according to Equation (9.7) will be stored in tracking tables $Track\_h$, $Track\_Gt$, respectively.

The final solution is found in the last cell of the table, i.e., $DP(K,tb_m)$, which holds the overall maximum performance of K applications under the maximum available thermal budget. However, that does not necessarily mean that all K applications are mapped, rather the configurations of all K applications are examined. Extracting the settings of the applications starts from the last application $k=K$ until reaching the first one $k=1$. We begin from table $Track\_tb$ at the same entry (K,tb_m) of the maximum performance. $Track\_tb(K,tb_m)$ contains two values tb'', tb', as previously mentioned. If $tb''=0$, that means application K is not mapped, and hence all of its cores (map_k) are power-gated. If it is greater than 0, that implies that application K is mapped where the number of its threads is stored in $Track\_h(K,tb_m)$ and the selected guardband type is stored in $Track\_Gt(K,tb_m)$. For application $K-1$, its assigned thermal budget will be stored at the cell $Track\_tb(K-1,tb')$. Similarly, we extract the settings of application $K-1$ from the rest of tracking tables. We continue with this process until extracting the settings of all applications, which will be the output of the algorithm.
It is noteworthy that the power-gated cores of each application k are the last $H_k - h_k$ cores of the set map_k, because these cores will be the cores that exceeded the available thermal budget in the sub-solutions, as mentioned before in the explanation of the auxiliary function *opt*.
Algorithm 10 presents building table DP, while Algorithm 11 presents how the final solutions are extracted.

9.3.2 Iterative Thermal-Aware Guardbanding

As mentioned above, we propose a heuristic *sGuard-H* to select the guardband type of the cores considering the running workload but with less time complexity than the proposed dynamic programming algorithm, i.e., *sGuard-DP*. *sGuard-H* decomposes the problem into two steps. Firstly, it selects the guardband type of the cores. Secondly, it adjusts the power states of the cores in order to avoid any thermal violation. Iterative search is employed in this heuristic. The input of *sGuard-H* are the application set K with their power and performance models, as well as the application mapping sets map_k.

Algorithm 10 Building dynamic programming tables

Input: Application set with their power and performance tables P_k and R_k and the mapping decision map_k;

1: **for** $tb = tb_1$ **to** tb_m **do**
2: Set $DP(1, tb)$ according to Equation (9.5);
3: Set $DP^T(t, tb)$ according to Equation (9.6);
4: **end for**
5: **for** $k = 2$ **to** K **do**
6: **for** $tb = tb_1$ **to** tb_m **do**
7: $MaxDP \leftarrow -\infty$
8: **for** $tb' = 0$ **to** tb **do**
9: **for** $tb'' = 0$ **to** tb **do**
10: **if** $tb' + tb'' \geq tb$ **then**
11: Set $DP(k, tb)$ according to Equation (9.7);
12: Set $DP^T(k, tb)$ according to Equation (9.8);
13: **if** $DP(k, tb) > MaxDP$ **then**
14: $MaxDP = DP(k, tb)$;
15: $MaxDP^T = DP^T(k, tb)$;
16: $tb'^\star = tb'$;
17: **if** $DP(k, tb) = 0$ **or** $DP(k, tb) = DP(k-1, tb')$ **then**
18: $tb''^\star = 0$;
19: **else**
20: $tb''^\star = tb''$;
21: **end if**
22: **end if**
23: **end if**
24: **end for**
25: **end for**
26: $DP(k, tb) = MaxDP$;
27: $DP^T(k, tb) = MaxDP^T$;
28: $Track\_tb(k, tb) = (tb''^\star, tb'^\star)$;
29: $Track\_h(k, tb) = opt(k, tb''^\star)\_h$;
30: $Track\_Gt(k, tb) = opt(k, tb''^\star)\_Gt$;
31: **end for**
32: **end for**
33: **return** Tables $DP, Track\_h, Track\_Gt, Track\_tb$;

For selecting the guardband types, *sGuard-H* first initiates Gt_i with V_GB for all cores, since V_GB allows a higher frequency and hence a better performance can be achieved if no thermal violation occurs. After that, *sGuard-H* estimates the resulting power consumption and the corresponding steady-state temperatures of the cores (see Equation (8.4)). If the estimated temperatures of all cores do not exceed T_{crit}, V_GB will be the selected guardband type for all cores. Otherwise (i.e. if a thermal violation is foreseen), *sGuard-H* needs to change the guardband type of the application cores *iteratively*. To achieve that, *sGuard-H*, during temperature estimation, obtains the hottest core $i^\star$ whose guardband type is still V_GB in order to change it to F_GB. At the same time, the guardband types of all cores in $map_{k^\star}$, where $k^\star$ is the application that runs on core $i^\star$, are changed to F_GB. Note, that *sGuard-H* changes the guardband type for the hottest core, rather than only for the core that violates T_{crit}, because the

Algorithm 11 Extract application settings h_k, Gt_k^{app} and the core power states S_i

Input: Tables $DP, DP^T, Track\_h, Track\_Gt, Track\_tb$;

```
1: tb = tb_m;
2: for k = K to 1 do
3:    if Track_tb(k, tb)_tb'' > 0 then
4:       h_k = Track_h(k, tb);
5:       Gt_k^app = Track_Gt(k, tb);
6:    else
7:       h_k = 0;
8:       Gt_k^app = ∅;
9:    end if
10:   for i = H_K to h_k do
11:      S_map_k(i) = 0;
12:   end for
13:   tb = Track_tb(k, tb)_tb';
14: end for
15: return h_k, Gt_k^app for all applications and S_i for all cores.
```

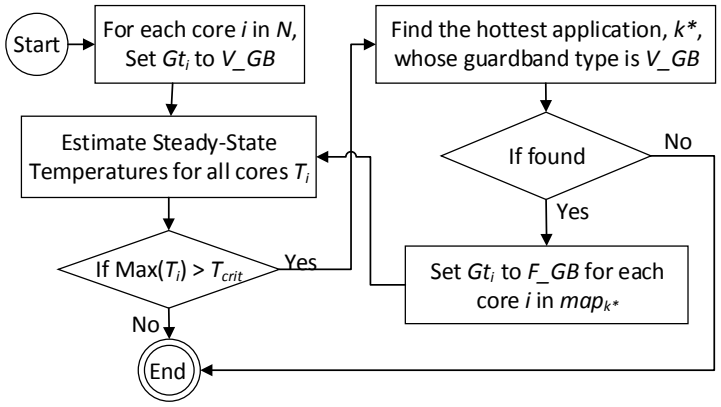


Figure 9.4: Flow diagram of the iterative guardband selection of *sGuard-H*

latter might have been set to F_GB in a previous iteration, nevertheless, it still violates T_{crit}. Thus, changing the guardband type of the hottest core will lead to decreasing the temperatures on the chip. Afterwards, the steady-state temperatures of the cores are estimated again. This process is repeated as long as thermal violations are foreseen and there is at least one core whose Gt_i is still V_GB. The flow diagram of this iterative selection of the guardband types is demonstrated in Fig.9.4.

The second step of our heuristic is to specify the power states of the cores, because thermal violations might still occur, even if *sGuard-H* set F_GB to all cores due to many reasons like high *chip load* and/or high-power applications. For this purpose, *sGuard-H* estimates the steady-state temperatures considering that the guardband types determined in the previous step are employed. If a thermal violation is foreseen, the hottest core is power-gated. When a core is power-gated, the number of threads of the application that runs on that core will be reduced. This process is repeated until none of the cores exceeds the critical temperature, T_{crit}.

9.4 Evaluation

The targeted multicore system is a 64-core chip shown in Figure 4.4. The complete setup is explained in details in Chapter 4, in which the tool flow is illustrated in Figure 4.1. We considered a nominal voltage and frequency of 1.4V and 4GHz, respectively. The estimated guardband width for F_GB and V_GB are 14% and 91 mv, respectively. We consider a critical temperature of 100 ^{c}ircC, similar to the critical temperature of Intel Xeon Phi [33].

Scenario	Applications
Mix1	x264, canneal
Mix2	blackscholes, ferret
Mix3	canneal, bodytrack
Mix4	x264, canneal, blackscholes
Mix5	canneal, bodytrack, dedup
Mix6	ferret, x264

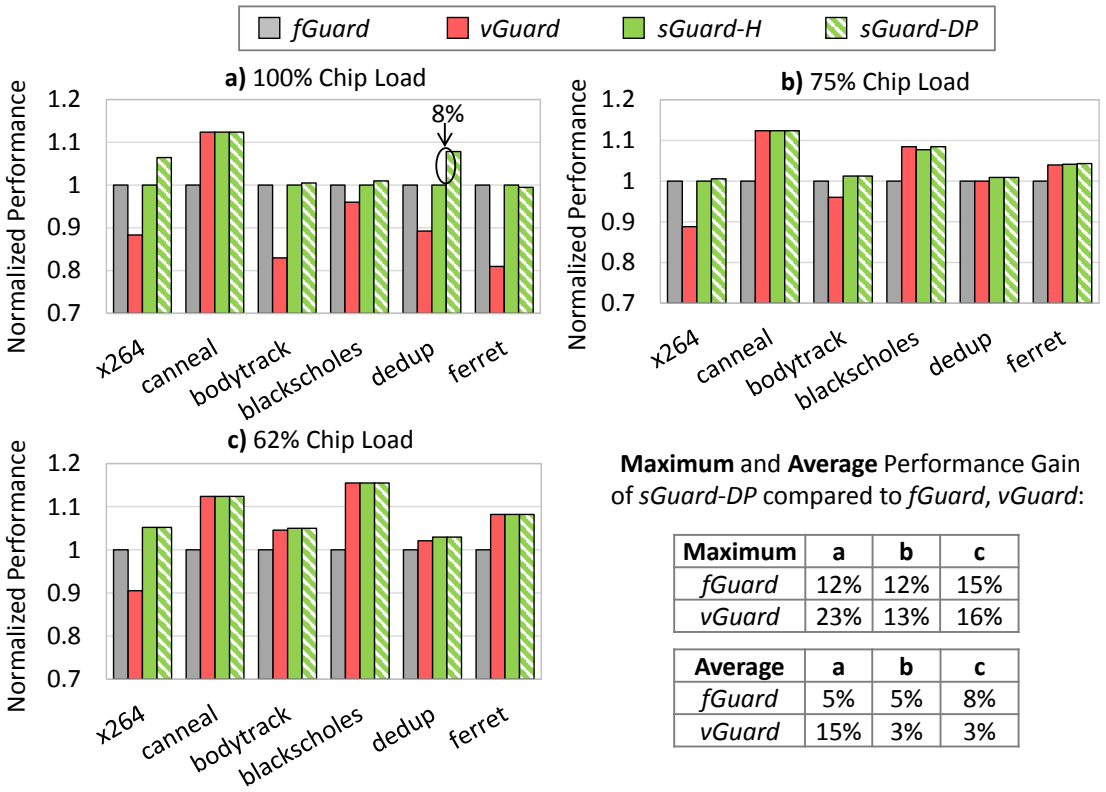
Table 9.1: Scenarios of workloads with multiple applications

9.4.1 Experimental Results

To evaluate the effectiveness of our thermal-aware guardbanding techniques *sGuard-DP* and *sGuard-H*, we additionally implemented two other techniques *vGuard* and *fGuard*, that constantly employ V_GB [48] and F_GB [47], respectively, on all cores. As discussed, our targeted system is thermally-constrained, and thus, thermal violations must be prevented. *sGuard-DP* jointly selects the guardband types and the power states of the cores, so that the performance is maximized without leading to thermal violations. The second step of *sGuard-H* power-gates cores so that no thermal violation occurs. For *vGuard* and *fGuard*, it is necessary also to prevent thermal violations. Therefore, the same process of avoiding thermal violation in *sGuard-H*, i.e., the second step of *sGuard-H*, is applied in *vGuard* and *fGuard*, to allow fair comparison.

The metric used to evaluate the efficiency of the four guardbanding techniques is the overall system performance (Equation (9.3)). It is to be noted that the performance values in all figures are always normalized to the performance of *fGuard*. We summarize our evaluation results for various scenarios of static and dynamic workloads under five categories (as seen below). As discussed in Section.9.3, *sGuard-DP* is more suitable for static workloads. Therefore, we evaluate it for the first and second categories, because the rest of categories are testing dynamic workloads.

1) Static Workload – Single Application: First we study static workloads (i.e. no changes in the application set over time) consisting of several instances (determining the *chip load*) from the same application. This is analogous to our motivational case study in Section 9.1. As shown in Figure 9.5, the resulting performance of *sGuard-DP* and *sGuard-H* in most cases is similar to the performance of *vGuard* or the performance of *fGuard*. The reason is that the workload is static and a single application is executed, and thus *sGuard-DP* and *sGuard-H* select most likely one guardband type of the cores that is appropriate to the given single application of the static workload.



Maximum	a	b	c
fGuard	12%	12%	15%
vGuard	23%	13%	16%

Average	a	b	c
fGuard	5%	5%	8%
vGuard	15%	3%	3%

Figure 9.5: Comparison results of the average system performance between *sGuard-DP*, *sGuard-H*, *vGuard* and *fGuard* techniques under *static* workloads.

In few cases, like the case of "dedup" application at scenario (a) of Figure 9.5, *sGuard-DP* outperforms both *vGuard* and *fGuard*. In this case, a thermal violation is foreseen at the cores that run one instance of the application; although the instances are identical and consume the same power, but due to the geometrical locations of the cores on the chip, some cores are more subject to temperature increase than others. Since *sGuard-DP* jointly determines the guardband types and the power states of the cores after examining a lot of feasible combinations, it finds that, in this scenario power-gating the cores that exceed the T_{crit} is more efficient (result in more performance) than converting the guardband type of the cores to F_GB. However, *sGuard-H* will not be able to test this feasible option, since it solves the problem through two steps; first determining the guardband types, and then determining the power states of the cores. As a result, *sGuard-DP* achieves 8% more performance compared to *sGuard-H* in this particular scenario. The benefit of *sGuard-DP* over *sGuard-H* becomes more evident when the workload consists of different applications as seen in category (2).

As a summary of this category, our *sGuard-DP* achieves a performance gain up to 23%, 15% compared to *vGuard* and *fGuard*, respectively, in which a fixed guardband type is constantly employed throughout without considering the workload.
This demonstrates again why a thermal-aware guardbanding is indispensable to optimize performance.

2) Static Workload – Multiple Applications: Here we study static workloads, where each of them consists of multiple different applications (see Table 9.1). *sGuard-DP* achieves a performance gain up to 16% and an average of 6% compared to *sGuard-H*. We notice that the difference in performance between *sGuard-DP* and *sGuard-H* is higher in

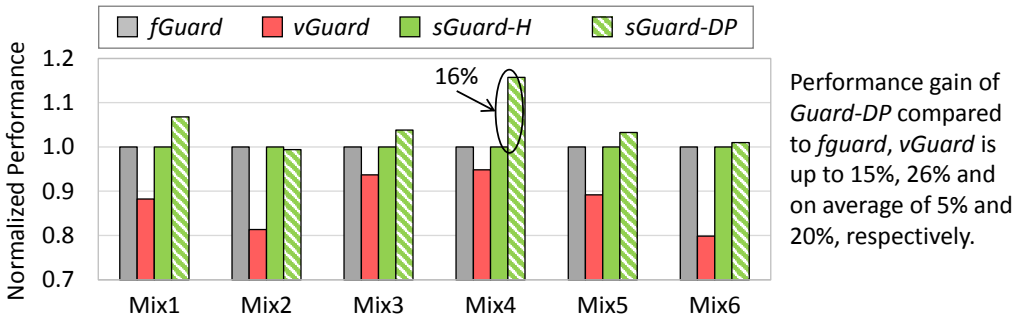


Figure 9.6: Comparison results of the average performance between *sGuard-DP*, *sGuard-H*, *vGuard* and *fGuard* techniques under *static* workloads that consist of different multiple applications.

this category compared to the first one where each workload consists of multiple instances of the same application. The reason is that when different applications exist in the workload, more combinations need to be studied in order to obtain higher performance. Our dynamic programming algorithm *sGuard-DP* studies more feasible combinations than the proposed heuristic *sGuard-H*, and thus better performance is obtained.

3) Dynamic Workload – Single Application: This category considers a dynamic workloads in which the *chip load* (i.e. number of demanded cores by the running workload) varies over time. An example of dynamic workloads is illustrated in Fig.9.7, in which the workload consists of multiple instances of "x264" application. The number of instances running on the cores are changing over the execution time of the experiment. Hence, the *chip load* percentage is changing as shown in the upper graph in Fig.9.7 resulting in an average of 66%. Our *sGuard-H* reacts to these changes and accordingly selects the appropriate guardband type at any workload change resulting in different frequencies over time. For instance, when the *chip load* increases above 50% (at t=8s) *sGuard-H* selects F_GB to avoid potential thermal violations that might occur if V_GB remained in use, and thereby the frequency is reduced to 3.4 GHz at t=8 s. Thanks to this thermal-aware guardbanding, the performance gain of our *sGuard-H* reaches up to 32% more than the resulting one at V_GB and up to 13% more than the resulting one at F_GB.

A similar experiments are run for the rest of applications and the resulting average performance is plotted in Fig.9.8(a). Similar experiments are conducted for 54% average *chip load*, and the results are shown in Fig.9.8(b). As noticed, in this category the achieved performance by *sGuard-H* in not similar to either *vGuard* or *fGuard*, unlike static workloads. Rather, *sGuard-H* outperforms both of *vGuard* and *fGuard* in most applications because in this case the decision of *sGuard-H* with regard to the selected guardband will vary over time according to the workload changes.

4) Dynamic Workload – Multiple Applications: For a more general evaluation, this category considers a dynamic workload in which the *chip load* varies over time, and the workload consists of different applications. Thus, we use the scenarios defined in Table 9.1 and additionally let the chip load change over time, i.e., the number of instances of each application in the scenarios will vary over time. Fig.9.9 shows the minimum, the maximum and the average performance gain of *sGuard-H* compared to *vGuard* and *fGuard* over the execution time of each scenario. While Figure 9.11 shows comparison of the average performance resulting from these techniques.

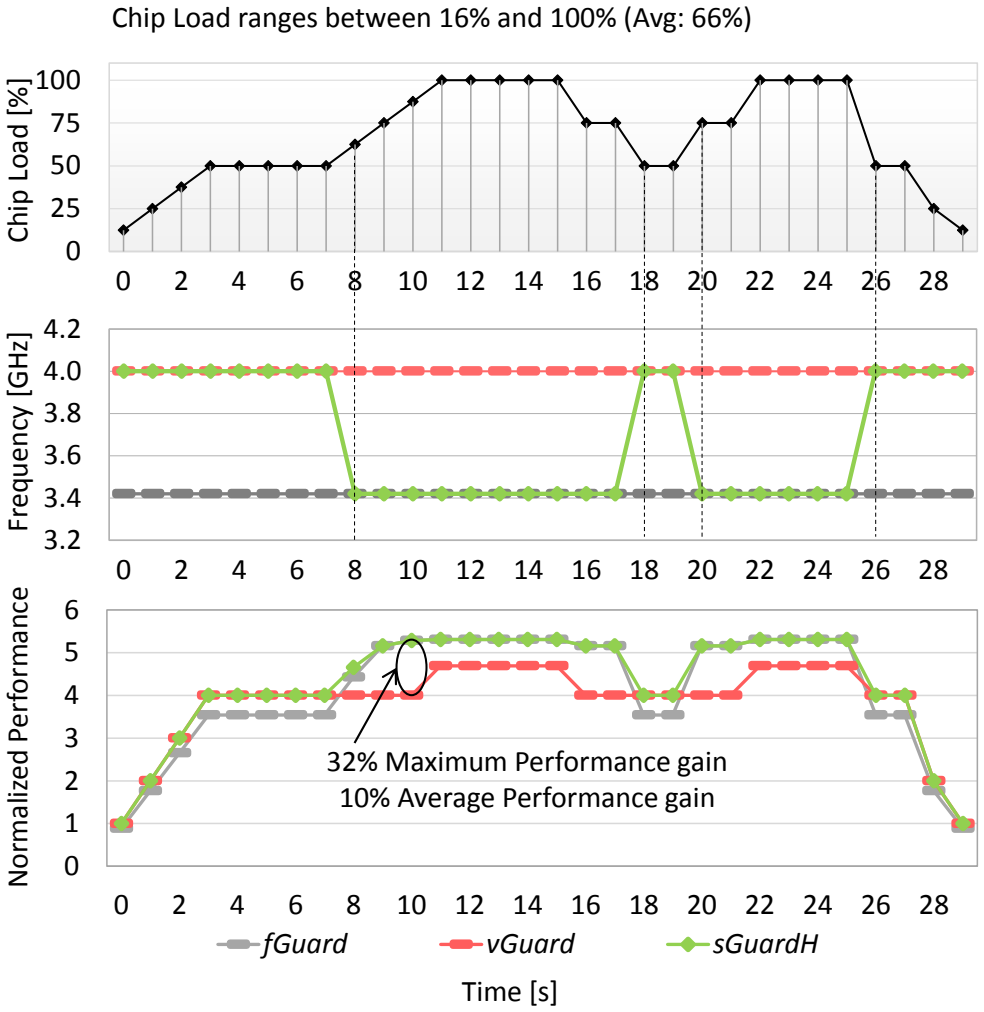
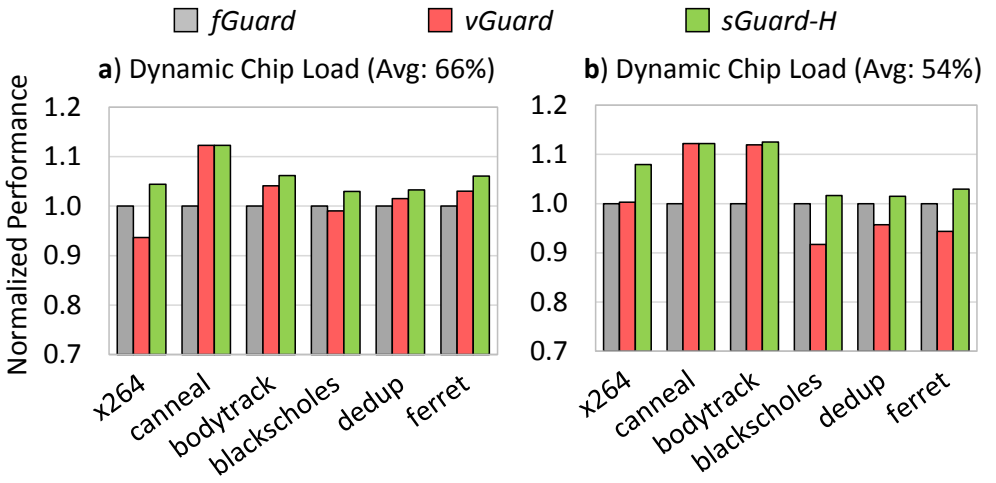


Figure 9.7: Demonstrating how the selected guardband type by *sGuard-H* changes between V_GB and F_GB over time following the changes in the *chip load*. Accordingly, the frequency changes between 4GHz and 3.42GHz. The resulting performance gain of *sGuard-H* reaches up to 32% and 13% compared to V_GB and F_GB, respectively.

As mentioned above, when the workload consists of multiple instances of the same application as in category (1 and 3), *sGuard-H* most likely assigns the same guardband type for all cores at one point of time, because every application instance consumes the same power. However, when the workload consists of multiple instances from different applications, *sGuard-H* may select different guardband types at one point of time due to variances in the power consumptions resulting from running different applications. To demonstrate that, Figure 9.10 shows how different guardband types are selected at one time for Mix1 that composed of "x264" and "canneal" applications.

5) Evaluating our *sGuard-H* at a chip, instead of core, granularity: In our system model, we assume that the V and f can be changed for every core individually, similar to some commercial processors, like [37]. However, such a feature may not be available in all processors like in Intel Xeon Phi [33].Therefore, we implemented a version of our technique, referred to as *sGuard-H (chip)*, which considers that all cores share the same V and f and hence the same guardbands. Our experiment in Figure 9.11 shows that the performance gain of *sGuard-H (chip)* is less than the one of our original technique *sGuard-H*. The reason is that at the chip granularity the selected guardband



Maximum and **Average** Performance Gain of *sGuard-H* compared to *fGuard*, *vGuard*:

Maximum	a	b
fGuard	12%	12%
vGuard	11%	10%

Average	a	b
fGuard	6%	7%
vGuard	4%	6%

Figure 9.8: Comparison results of the average system performance under *dynamic* workloads.

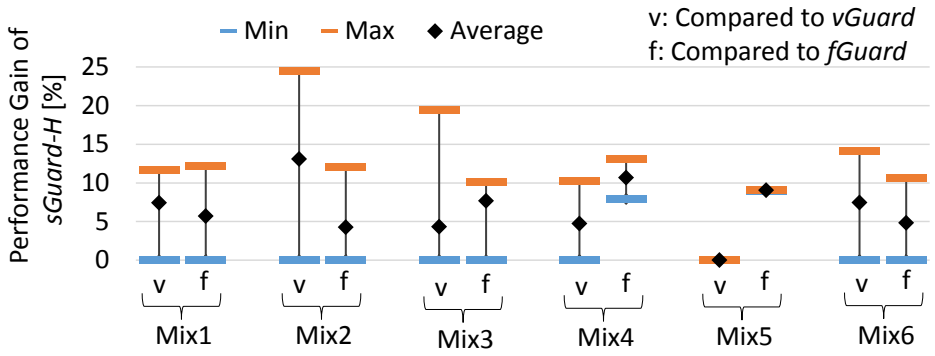


Figure 9.9: Comparison of the minimum, maximum and average performance gain of *sGuard-H* compared to *vGuard* and *fGuard* over the execution time of each workload scenario. The maximum performance gain of *sGuard-H* reaches up to 25% and 15% compared to *vGuard* and *fGuard*, respectively.

type will be appropriate for the *dominant* application (i.e. the one that leads to generate the highest temperature on the cores). Particularly, if the estimated temperatures of the cores that execute the *dominant* application exceed the T_{crit}, F_GB must be selected for these cores. Since only per-chip VF scaling is possible, the guardbands of all cores will be set also to F_GB, although their temperatures might still be far from T_{crit}, and V_GB is more appropriate for them. Such an unexploited thermal margin on these cores leads to the less performance gain. As an example from our experiments shown in Fig. 12, *sGuard-H* achieves 4% more performance than *sGuard-H (chip)* in the case of Mix4. However, *sGuard-H (chip)* still achieves better performance than fixed guardbanding, i.e., the performance gains of *sGuard-H (chip)* are 11% and 9% compared to *vGuard* and *fGuard*, respectively. The performance gains of *sGuard-H* in this experiment are 14% and 10% compared to *vGuard* and *fGuard*, respectively.

Overhead Discussion: As mentioned before, *sGuard-DP* is utilized for static workload,

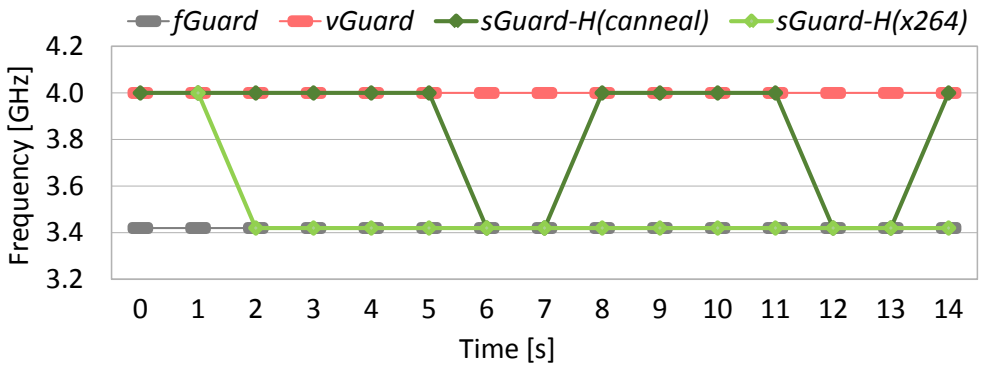


Figure 9.10: Mapping multiple applications leads to selecting different guardband types at one time at the cores that run different applications.

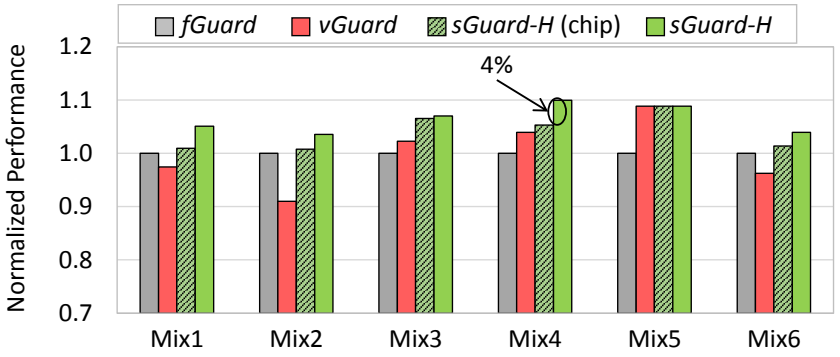


Figure 9.11: Average Performance Comparison for different scenarios. The Performance gain of *sGuard-H* is up to 14% and 10% compared to *vGuard* and *fGuard*, respectively.

because the required time to take its decision might not be affordable if the workload frequently changes (the workload changes when an application finishes or a new application arrives). To provide a deeper insight into the overhead of *sGuard-DP*, we measure the required time to take its decisions for the adopted scenarios shown in Table 9.1. The maximum required time is 184 ms. Although we assume that *sGuard-DP* is utilized for static workloads, it can be utilized in dynamic workloads as well, if the period of time between workload changes is much longer than the required time of the decision making of *sGuard-DP*, so that its overhead becomes affordable. Otherwise, *sGuard-H* is utilized which has much lower overhead. Specifically, we measure the overhead of *sGuard-H* for the same adopted scenarios, and it is 2 ms, which means it has 85X less overhead than *sGuard-DP*.

The overhead of the comparison candidates, i.e., *vGuard* and *fGuard*, is similar to the overhead of *sGuard-H* (around 2 ms), due to the same time complexity of these heuristics. In particular, *sGuard-H*, *vGuard* and *fGuard* have a cubic time complexity, due to their iterating through all cores to check thermal violations, and at each iteration the steady-state temperatures are estimated, where estimating the steady-state temperatures has a quadratic time complexity (Equation (8.4)).

9.5 Summary

This chapter introduced thermal-aware guardbanding that selects the appropriate guardband type for every core while taking the workload-induced temperatures into account and thereby optimizing performance. The diverse evaluations demonstrated that the presented guardband selection can significantly improve system performance (up to 32%) compared to employing a single fixed guardband type throughout the processor lifetime, while the same reliability and temperature constraints are considered. Concisely, this investigation can motivate researchers to move from circuit-level fixed guardbanding to system-level thermal-aware guardbanding, in order to be able to consider workload-induced temperatures.

Chapter 10

Conclusions

This dissertation focused on the problem of performance optimization under temperature and aging constraints. The first part of the dissertation (Chapters 5, 6) focused on performance optimization under temperature constraints, while the second part (Chapters 7, 8, 9) considered both temperature and aging.

Chapter 5 showed how dark silicon brings new challenges as well as opportunities for the targeted problem. Thus, a dark silicon-aware resource management, named *DsRM*, is proposed in Chapter 5 to manage resources for homogeneous multicores. *DsRM* leveraged the existence of dark cores to dissipate heat from active cores. That allowed *DsRM* to upscale the V/f levels of active cores that execute high ILP applications, thereby obtaining further performance improvements. Thus *DsRM* was able to obtain significant performance improvements with an average of 34%. This technique, however, could not be extended for heterogeneous multicores. Therefore, in Chapter 6, a resource management technique is proposed that aims at maximizing the performance under a temperature constraint while considering the diverse power, performance and thermal characteristics of heterogeneous cores.

In Chapter 7, a new aging-aware design space is explored enabling the resource management technique from maximizing the performance under temperature and aging constraints. The evaluation of this technique showed the potentials of performance improvements when considering the entire aging-aware design space rather than employing conservative bounds on the design space. An aging-aware, yet efficient, boosting technique is developed in Chapter 8, which was able to maximize the performance while at the same time reduce the delays stemming from both the well-known long-term aging effects and the recently-discovered short-term aging effects. Chapter 9 demonstrated that the dynamic selection of guardband types can significantly improve system performance (up to 32%) compared to employing a single fixed guardband type throughout the processor lifetime, while the same temperature constraint is considered. Concisely, our investigation in Chapter 9 can motivate researchers to move from circuit-level fixed guardbanding to system-level thermal-aware guardbanding in order to be able to consider workload-induced temperatures and optimize for performance accordingly.

A future work for this dissertation will be to develop more dynamic techniques that account for the uncertainty and variability of the execution behavior of applications at runtime. Such uncertainty and variability might result due to the use of shared resources or due to changes in input data. As can be noticed, most of the techniques proposed

in the dissertation depend on power and performance models for the applications that are extracted at design time. On the one hand, considering these profiles enables understanding the application characteristics; their TLP and ILP, thereby maximizing the performance accordingly. On the other hand, if changes in these characteristics happen, it might not be possible to obtain the expected performance gain. In contrast, the technique proposed in Chapter 8 does not depend on any design-time modeling, it depends only on the current values of the power and temperature form the multicores and takes its decisions accordingly. Hence, this technique is not able to optimize for application performance based on their characteristics. As a result, an improvement on these works could be achieved by considering first the design time profiles and attempting to modify it based on runtime observations. This way could combine the benefits of both approaches. To achieve this improvement, machine learning algorithms are good candidates, due to their ability to learn dynamic behaviors.

Bibliography

[1] H. Khdr, H. Amrouch, and J. Henkel, "Aging-Aware Boosting," *IEEE Transactions on Computers*, 2018.

[2] ——, "Dynamic Guardband Selection: Thermal-Aware Optimization for Unreliable Multi-Core Systems," *IEEE Transactions on Computers*, 2018.

[3] H. Khdr, S. Pagani, M. Shafique, and J. Henkel, "Dark-Silicon-Aware Resource Management for Many-Core Systems," in *Proceedings of the 52nd Annual Design Automation Conference*. ACM, 2015, p. 179.

[4] H. Khdr, H. Amrouch, and J. Henkel, "Aging-Constrained Performance Optimization for Multi Cores," in *Proceedings of the 55nd Annual Design Automation Conference*. ACM, 2018.

[5] 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.

[6] 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.

[7] H. Khdr, T. Ebi, M. Shafique, H. Amrouch, and J. Henkel, "mdtm: Multi-objective dynamic thermal management for on-chip systems," in *Proceedings of the conference on Design, Automation & Test in Europe*. European Design and Automation Association, 2014, p. 330.

[8] A. Pathania, H. Khdr, M. Shafique, T. Mitra, and J. Henkel, "Qos-aware stochastic power management for many-cores," 2018.

[9] S. Pagani, H. Khdr, J.-J. Chen, M. Shafique, M. Li, and J. Henkel, "Thermal safe power (tsp): Efficient power budgeting for heterogeneous manycore systems in dark silicon," *IEEE Transactions on Computers*, vol. 66, no. 1, pp. 147–162, 2017.

[10] A. Pathania, H. Khdr, M. Shafique, T. Mitra, and J. Henkel, "Scalable probabilistic power budgeting for many-cores," in *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2017, pp. 864–869.

[11] S. Pagani, H. Khdr, J.-J. Chen, M. Shafique, M. Li, and J. Henkel, "Thermal safe power: efficient thermal-aware power budgeting for manycore systems in dark silicon," in *The Dark Side of Silicon*. Springer, 2017, pp. 125–158.

[12] S. Pagani, L. Bauer, Q. Chen, E. Glocker, F. Hannig, A. Herkersdorf, H. Khdr, A. Pathania, U. Schlichtmann, D. Schmitt-Landsiedel *et al.*, "Dark silicon management: an integrated and coordinated cross-layer approach," *it-Information Technology*, vol. 58, no. 6, pp. 297–307, 2016.

[13] J. Henkel, S. Pagani, H. Khdr, F. Kriebel, S. Rehman, and M. Shafique, "Towards performance and reliability-efficient computing in the dark silicon era," in *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*. EDA Consortium, 2016, pp. 1–6.

[14] S. Pagani, M. Shafique, H. Khdr, J.-J. Chen, and J. Henkel, "seboost: selective boosting for heterogeneous manycores," in *Proceedings of the 10th International Conference on Hardware/Software Codesign and System Synthesis*. IEEE Press, 2015, pp. 104–113.

[15] J. Henkel, H. Bukhari, S. Garg, M. U. K. Khan, H. Khdr, F. Kriebel, U. Ogras, S. Parameswaran, and M. Shafique, "Dark silicon: From computation to communication," in *Proceedings of the 9th International Symposium on Networks-on-Chip*. ACM, 2015, p. 23.

[16] J. Henkel, H. Khdr, S. Pagani, and M. Shafique, "New trends in dark silicon," in *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*. IEEE, 2015, pp. 1–6.

[17] W. Munawar, H. Khdr, S. Pagani, M. Shafique, J.-J. Chen, and J. Henkel, "Peak power management for scheduling real-time tasks on heterogeneous many-core systems," in *Parallel and Distributed Systems (ICPADS), 2014 20th IEEE International Conference on*. IEEE, 2014, pp. 200–209.

[18] S. Pagani, H. Khdr, W. Munawar, J.-J. Chen, M. Shafique, M. Li, and J. Henkel, "TSP: Thermal Safe Power - efficient power budgeting for many-core systems in dark silicon," in *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2014, pp. 10:1–10:10.

[19] J. Henkel, T. Ebi, H. Amrouch, and H. Khdr, "Thermal management for dependable on-chip systems," in *Design Automation Conference (ASP-DAC), 2013 18th Asia and South Pacific*. IEEE, 2013, pp. 113–118.

[20] J. Teich, J. Henkel, A. Herkersdorf, D. Schmitt-Landsiedel, W. Schröder-Preikschat, and G. Snelting, "Invasive computing: An overview," in *Multiprocessor System-on-Chip - Hardware Design and Tool Integration.*, 2011, pp. 241–268. [Online]. Available: https://doi.org/10.1007/978-1-4419-6460-1_11

[21] J. Henkel, A. Herkersdorf, L. Bauer, T. Wild, M. Hübner, R. K. Pujari, A. Grudnitsky, J. Heisswolf, A. Zaib, B. Vogel *et al.*, "Invasive manycore architectures." in *ASP-DAC*, 2012, pp. 193–200.

[22] M. Shafique and J. Henkel, "Agent-based distributed power management for kilo-core processors," in *The IEEE/ACM International Conference on Computer-Aided Design, ICCAD'13, San Jose, CA, USA, November 18-21, 2013*, 2013, pp. 153–160. [Online]. Available: https://doi.org/10.1109/ICCAD.2013.6691112

[23] S. Muddasani, S. Boppu, F. Hannig, B. Kuzmin, V. Lari, and J. Teich, "A prototype of an invasive tightly-coupled processor array," in *Design and Architectures for Signal and Image Processing*, 2012.

[24] S. Pagani, A. Pathania, M. Shafique, J.-J. Chen, and J. Henkel, "Energy efficiency for clustered heterogeneous multicores," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 5, pp. 1315–1330, 2017.

[25] J. Henkel, L. Bauer, J. Becker, O. Bringmann, U. Brinkschulte, S. Chakraborty, M. Engel, R. Ernst, H. Härtig, L. Hedrich *et al.*, "Design and architectures for dependable embedded systems," in *Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. ACM, 2011, pp. 69–78.

[26] H. Amrouch, T. Ebi, and J. Henkel, "Stress balancing to mitigate nbti effects in register files," in *Dependable Systems and Networks (DSN), 2013 43rd Annual IEEE/IFIP International Conference on*. IEEE, 2013, pp. 1–10.

[27] H. Amrouch, S. Mishra, V. van Santen, S. Mahapatra, and J. Henkel, "Impact of bti on dynamic and static power: From the physical to circuit level," in *Reliability Physics Symposium (IRPS), 2017 IEEE International*. IEEE, 2017, pp. CR–3.

[28] V. M. van Santen, H. Amrouch, N. Parihar, S. Mahapatra *et al.*, "Aging-aware voltage scaling," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2016, pp. 576–581.

[29] I. Present, "Cramming more components onto integrated circuits," *Readings in computer architecture*, vol. 56, 2000.

[30] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc, "Design of ion-implanted mosfet's with very small physical dimensions," *IEEE Journal of Solid-State Circuits*, vol. 9, no. 5, pp. 256–268, 1974.

[31] Y. Taur and E. J. Nowak, "Cmos devices below 0.1/spl mu/m: how high will performance go?" in *Electron Devices Meeting, 1997. IEDM'97. Technical Digest., International*. IEEE, 1997, pp. 215–218.

[32] "International technology roadmap for semiconductors 2.0," http://www.itrs2.net/, 2015.

[33] Intel Corporation, "Intel xeon phi processor datasheet," https://ark.intel.com/products/94709/Intel-Xeon-Phi-Processor-7210F-16GB-1.30-GHz-64-core, December 2017.

[34] R. Borkar, M. Bohr, and S. Jourdan, "Advancing moore's law in 2014."

[35] E. Maricau and G. Gielen, "Cmos reliability overview," in *Analog IC Reliability in Nanometer CMOS*. Springer, 2013, pp. 15–35.

[36] S. Hppner, C. Shao, H. Eisenreich, G. Ellguth, M. Ander, and R. Schffny, "A power management architecture for fast per-core dvfs in heterogeneous mpsocs," in *2012 IEEE International Symposium on Circuits and Systems*, May 2012, pp. 261–264.

[37] D. Hackenberg, R. Schöne, T. Ilsche, D. Molka, J. Schuchart, and R. Geyer, "An energy efficiency feature survey of the intel haswell processor," in *Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International*. IEEE, 2015, pp. 896–904.

[38] S. H. Gunther, F. Binns, D. M. Carmean, and J. C.Hall, "Managing the impact of increasing microprocessor power consumption," *Intel Tchnology Journal, Q1*, 2001.

[39] N. Binkert, B. Beckmann, G. Black *et al.*, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[40] S. Li, J.-H. Ahn, R. Strong, J. Brockman, D. Tullsen, and N. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *42nd IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2009, pp. 469–480.

[41] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2008, pp. 72–81.

[42] X. Guo and M. R. Stan, "Work hard, sleep well-avoid irreversible ic wearout with proactive rejuvenation," in *Design Automation Conference (ASP-DAC), 2016 21st Asia and South Pacific*. IEEE, 2016, pp. 649–654.

[43] K. Joshi, S. Mukhopadhyay, N. Goel, and S. Mahapatra, "A consistent physical framework for N and P BTI in HKMG MOSFETs," in *Reliability Physics Symposium (IRPS), IEEE International*, 2012, pp. 5A–3.

[44] Y. Chauhan, S. Venugopalan, M. Karim, S. Khandelwal, N. Paydavosi, P. Thakur, A. Niknejad, and C. Hu, "BSIM - Industry standard compact MOSFET models," in *ESSCIRC*, 2012.

[45] "BSIM Compact MOSFET Models for SPICE Simulation," http://www-device.eecs.berkeley.edu/bsim/?page=BSIM4.

[46] C. Zhou, X. Wang, W. Xu, Y. Zhu, V. J. Reddi, and C. H. Kim, "Estimation of instantaneous frequency fluctuation in a fast dvfs environment using an empirical bti stress-relaxation model," in *2014 IEEE International Reliability Physics Symposium*, June 2014, pp. 2D.2.1–2D.2.6.

[47] M. Ebrahimi, F. Oboril, S. Kiamehr, and M. B. Tahoori, "Aging-aware logic synthesis," in *Proceedings of the International Conference on Computer-Aided Design*, ser. ICCAD '13, 2013.

[48] T. B. Chan, W. T. J. Chan, and A. B. Kahng, "Impact of adaptive voltage scaling on aging-aware signoff," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2013, pp. 1683–1688.

[49] S. Kobbe, L. Bauer, D. Lohmann, W. Schroder-Preikschat, and J. Henkel, "DistRM: Distributed Resource Management for On-Chip Many-Core Systems," in *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2011 Proceedings of the 9th International Conference on*, 2011, pp. 119–128.

[50] A. B. Downey, "A model for speedup of parallel programs," CALIFORNIA UNIV BERKELEY COMPUTER SCIENCE DIV, Tech. Rep., 1997.

[51] I. Anagnostopoulos, V. Tsoutsouras, A. Bartzas, and D. Soudris, "Distributed run-time resource management for malleable applications on many-core platforms," in *50th Design Automation Conference (DAC)*, 2013, pp. 168:1–168:6.

[52] G. Castilhos *et al.*, "Distributed resource management in NoC-based MPSoCs with dynamic cluster sizes," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, Aug 2013, pp. 153–158.

[53] M. Fattah, M. Daneshtalab, P. Liljeberg, and J. Plosila, "Smart hill climbing for agile dynamic mapping in many-core systems," in *50th Design Automation Conference (DAC)*, 2013, pp. 39:1–39:6.

[54] C.-L. Chou, U. Ogras, and R. Marculescu, "Energy- and performance-aware incremental mapping for networks on chip with multiple voltage levels," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 27, no. 10, pp. 1866–1879, Oct. 2008.

[55] K. Ma, X. Li, M. Chen, and X. Wang, "Scalable Power Control for Many-core Architectures Running Multi-threaded Applications," ser. ISCA '11, 2011, pp. 449–460.

[56] X. Wang, B. Zhao, T. Mak, M. Yang, Y. Jiang, M. Daneshtalab, and M. Palesi, "Adaptive power allocation for many-core systems inspired from multiagent auction model," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2014, pp. 1–4.

[57] T. Muthukaruppan, M. Pricopi, V. Venkataramani, T. Mitra, and S. Vishin, "Hierarchical power management for asymmetric multi-core in dark silicon era," in *50th Design Automation Conference (DAC)*, 2013, pp. 174:1–174:9.

[58] C. Hankendi and A. K. Coskun, "Scale & cap: Scaling-aware resource management for consolidated multi-threaded applications," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 22, no. 2, p. 30, 2017.

[59] A. Raghavan, Y. Luo, A. Chandawalla, M. Papaefthymiou, K. P. Pipe, T. F. Wenisch, and M. M. Martin, "Computational sprinting," in *High Performance Computer Architecture (HPCA), 2012 IEEE 18th International Symposium on.* IEEE, 2012, pp. 1–12.

[60] A. Kanduri, M.-H. Haghbayan, A.-M. Rahmani, P. Liljeberg, A. Jantsch, and H. Tenhunen, "Dark silicon aware runtime mapping for many-core systems: A patterning approach," in *Computer Design (ICCD), 2015 33rd IEEE International Conference on.* IEEE, 2015, pp. 573–580.

[61] M. Shafique, D. Gnad, S. Garg, and J. Henkel, "Variability-aware dark silicon management in on-chip many-core systems," in *Design, Automation and Test in Europe (DATE)*, 2015.

[62] M. Kadin and S. Reda, "Frequency and voltage planning for multi-core processors under thermal constraints," in *Computer Design, 2008. ICCD 2008. IEEE International Conference on.* IEEE, 2008, pp. 463–470.

[63] M. Kadin, S. Reda, and A. Uht, "Central vs. distributed dynamic thermal management for multi-core processors: which one is better?" in *Proceedings of the 19th ACM Great Lakes symposium on VLSI.* ACM, 2009, pp. 137–140.

[64] V. Hanumaiah, S. Vrudhula, and K. Chatha, "Performance optimal online dvfs and task migration techniques for thermally constrained multi-core processors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 30, no. 11, pp. 1677–1690, Nov. 2011.

[65] G. Singla, G. Kaur, A. K. Unver, and U. Y. Ogras, "Predictive dynamic thermal and power management for heterogeneous mobile platforms," in *Proceedings of Design, Automation & Test in Europe (DATE)*. IEEE, 2015, pp. 960–965.

[66] "Exynos 5 octa (5422)," http://www.samsung.com/exynos.

[67] Intel Corporation, "Intel turbo boost technology in intel CoreTM microarchitecture (nehalem) based processors," White Paper, November 2008.

[68] "7th generation intel processor families for u/y platforms," ser. January, 2017.

[69] S. Nussbaum, "AMD trinity APU," in *Hot Chips: A Symposium on High Performance Chips*, 2012.

[70] J. Charles, P. Jassi, N. S. Ananth, A. Sadat, and A. Fedorova, "Evaluation of the intel®corei7 turbo boost feature," in *Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC)*, ser. IISWC '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 188–197.

[71] A. Coskun, T. Rosing, K. Whisnant, and K. Gross, "Temperature-aware mpsoc scheduling for reducing hot spots and gradients," in *the Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2008, pp. 49–54.

[72] A. K. Coskun, T. S. Rosing, and K. C. Gross, "Proactive temperature balancing for low cost thermal management in mpsocs," in *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design*. IEEE Press, 2008, pp. 250–257.

[73] Y. Ge *et al.*, "A Multi-Agent Framework for Thermal Aware Task Migration in Many-Core Systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,*, vol. 20, no. 10, pp. 1758–1771, 2012.

[74] Z. Liu *et al.*, "Task migrations for distributed thermal management considering transient effects," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 2, pp. 397–401, 2015.

[75] J. Jahn, M. A. Al Faruque, and J. Henkel, "Carat: Context-aware runtime adaptive task migration for multi core architectures," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*. IEEE, 2011, pp. 1–6.

[76] S. Sharifi, A. K. Coskun, and T. S. Rosing, "Hybrid dynamic energy and thermal management in heterogeneous embedded multiprocessor socs," in *Proceedings of the 2010 Asia and South Pacific Design Automation Conference*, 2010, pp. 873–878.

[77] D. Puschini *et al.*, "Temperature-Aware Distributed Run-Time Optimization on MP-SoC Using Game Theory," in *IEEE Computer Society Annual Symposium on VLSI*, 2008.

[78] T. Ebi, M. Faruque, and J. Henkel, "TAPE: Thermal-aware agent-based power econom multi/many-core architectures," in *Computer-Aided Design - Digest of Technical Papers, 2009. ICCAD 2009. IEEE/ACM International Conference on*, 2009, pp. 302–309.

[79] T. Ebi, D. Kramer, W. Karl, and J. Henkel, "Economic learning for thermal-aware power budgeting in many-core architectures," in *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2011 Proceedings of the 9th International Conference on*, 2011, pp. 189–196.

[80] M. A. Faruque, J. Jahn, T. Ebi, and J. Henkel, "Runtime thermal management using software agents for multi- and many-core architectures," *IEEE Design Test of Computers*, vol. 27, no. 6, pp. 58–68, 2010.

[81] M. H. Haghbayan, A. Miele, A. M. Rahmani, P. Liljeberg, and H. Tenhunen, "Performance/reliability-aware resource management for many-cores in dark silicon era," *IEEE Transactions on Computers*, vol. 66, no. 9, pp. 1599–1612, Sept 2017.

[82] P. Mercati, F. Paterna, A. Bartolini, L. Benini, and T. . 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, Sept 2017.

[83] L. Wang, X. Wang, H.-f. Leung, and T. Mak, "Throughput optimization for lifetime budgeting in many-core systems," in *Proceedings of the on Great Lakes Symposium on VLSI 2017*, ser. GLSVLSI '17. New York, NY, USA: ACM, 2017, pp. 451–454.

[84] V. Rathore, V. Chaturvedi, and T. Srikanthan, "Performance constraint-aware task mapping to optimize lifetime reliability of manycore systems," in *Proceedings of the 26th Edition on Great Lakes Symposium on VLSI*, ser. GLSVLSI '16. New York, NY, USA: ACM, 2016, pp. 377–380.

[85] P. Singh, E. Karl, D. Blaauw, and D. Sylvester, "Compact degradation sensors for monitoring nbti and oxide degradation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 9, pp. 1645–1655, 2012.

[86] J. Sun, R. Lysecky, K. Shankar, A. Kodi, A. Louri, and J. Roveda, "Workload assignment considering nbti degradation in multicore systems," *J. Emerg. Technol. Comput. Syst.*, vol. 10, no. 1, pp. 4:1–4:22, Jan. 2014.

[87] T. R. Mück, Z. Ghaderi, N. D. Dutt, and E. Bozorgzadeh, "Exploiting heterogeneity for aging-aware load balancing in mobile platforms," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 3, no. 1, pp. 25–35, Jan 2017.

[88] A. Das, R. A. Shafik, G. V. Merrett, B. M. Al-Hashimi, A. Kumar, and B. Veeravalli, "Reinforcement learning-based inter- and intra-application thermal optimization for lifetime improvement of multicore systems," in *Proceedings of the 51st Annual Design Automation Conference*, ser. DAC '14. New York, NY, USA: ACM, 2014, pp. 170:1–170:6. [Online]. Available: http://%doi.acm.org/10.1145/2593069.2593199

[89] N. Goel, T. Naphade, and S. Mahapatra, "Combined trap generation and transient trap occupancy model for time evolution of nbti during dc multi-cycle and ac stress," in *2015 IEEE International Reliability Physics Symposium*, April 2015, pp. 4A.3.1–4A.3.7.

[90] A. Tiwari and J. Torrellas, "Facelift: Hiding and slowing down aging in multicores," in *Microarchitecture, 2008. MICRO-41. 2008 41st IEEE/ACM International Symposium on.* IEEE, 2008, pp. 129–140.

[91] M. G. Moghaddam, A. Yamamoto, and C. Ababei, "Investigation of dvfs based dynamic reliability management for chip multiprocessors," in *High Performance Computing & Simulation (HPCS), 2015 International Conference on.* IEEE, 2015, pp. 563–568.

[92] V. Y. Raparti, N. Kapadia, and S. Pasricha, "Artemis: An aging-aware runtime application mapping framework for 3d noc-based chip multiprocessors," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 3, no. 2, pp. 72–85, April 2017.

[93] H. Amrouch, B. Khaleghi, A. Gerstlauer, and J. Henkel, "Reliability-aware design to suppress aging," in *Proceedings of the 53rd Annual Design Automation Conference*, 2016, p. 12.

[94] J. Keane and C. H. Kim, "Transistor aging," *IEEE Spectrum*, 2011.

[95] C. R. Lefurgy, A. J. Drake, M. S. Floyd, M. S. Allen-Ware, B. Brock, J. A. Tierno, and J. B. Carter, "Active management of timing guardband to save energy in power7," in *proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture.* ACM, 2011, pp. 1–11.

[96] Y. Zu, C. R. Lefurgy, J. Leng, M. Halpern, M. S. Floyd, and V. J. Reddi, "Adaptive guardband scheduling to improve system-level efficiency of the power7+," in *Proceedings of the 48th International Symposium on Microarchitecture.* ACM, 2015, pp. 308–321.

[97] V. J. Reddi, S. Kanev, W. Kim, S. Campanoni, M. D. Smith, G.-Y. Wei, and D. Brooks, "Voltage smoothing: Characterizing and mitigating voltage noise in production processors via software-guided thread scheduling," in *Microarchitecture (MICRO), 2010 43rd Annual IEEE/ACM International Symposium on.* IEEE, 2010, pp. 77–88.

[98] T. N. Miller, R. Thomas, X. Pan, and R. Teodorescu, "Vrsync: Characterizing and eliminating synchronization-induced voltage emergencies in many-core processors," in *Computer Architecture (ISCA), 2012 39th Annual International Symposium on.* IEEE, 2012, pp. 249–260.

[99] D. Wentzlaff and A. Agarwal, "Factored operating systems (fos): the case for a scalable operating system for multicores," *ACM SIGOPS Operating Systems Review*, vol. 43, no. 2, pp. 76–85, 2009.

[100] S. Buchwald, M. Mohr, and A. Zwinkau, "Malleable invasive applications." in *Software Engineering (Workshops)*, 2015, pp. 123–126.

[101] L. Mukhanov, D. S. Nikolopoulos, and B. R. de Supinski, "ALEA: fine-grain energy profiling with basic block sampling," *CoRR*, 2015.

[102] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K. Cameron, "Powerpack: Energy profiling and analysis of high-performance systems and applications," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 21, no. 5, pp. 658–671, May 2010.

[103] G. Liu, J. Park, and D. Marculescu, "Procrustes: Power constrained performance improvement using extended maximize-then-swap algorithm." *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1664–1676, 2015.

[104] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. Stan, "HotSpot: a compact thermal modeling methodology for early-stage VLSI design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 5, pp. 501–513, May 2006.

[105] F. Kreith, R. M. Manglik, and M. S. Bohn, *Principles of heat transfer*. Cengage learning, 2012.

[106] N. Parihar, N. Goel, A. Chaudhary, and S. Mahapatra, "A modeling framework for nbti degradation under dynamic voltage and frequency scaling," *IEEE Transactions on Electron Devices*, vol. 63, no. 3, pp. 946–953, March 2016.

[107] "Nangate, Open Cell Library," http://www.nangate.com/.

[108] M. Martins, J. M. Matos, R. P. Ribas, A. Reis, G. Schlinker, L. Rech, and J. Michelsen, "Open cell library in 15nm freepdk technology," in *Proceedings of the 2015 Symposium on International Symposium on Physical Design*, ser. ISPD '15. New York, NY, USA: ACM, 2015, pp. 171–178.

[109] "Synopsys," http://www.synopsys.com/.

[110] H. Amrouch, B. Khaleghi, A. Gerstlauer, and J. Henkel, "Reliability-aware design to suppress aging," in *Proceedings of the 53rd Annual Design Automation Conference*. ACM, 2016, p. 12.

[111] "Predictive Technology Model," http://ptm.asu.edu/.

[112] C. Celio, D. A. Patterson, and K. Asanovi, "The Berkeley Out-of-Order Machine (BOOM): An Industry-Competitive, Synthesizable, Parameterized RISC-V Processor," Berkeley, Tech. Rep., Jun 2015.

[113] "Discussions with Hussam Amrouch with Chair for Embedded Systems (CES), KIT."

[114] R. Kessler, "The alpha 21264 microprocessor," *IEEE Micro*, vol. 19, no. 2, pp. 24–36, Mar. 1999.

[115] F. Hannig, H. Ruckdeschel, H. Dutta, and J. Teich, "PARO: Synthesis of Hardware Accelerators for Multi-Dimensional Dataflow-Intensive Applications," in *Proceedings of the Fourth International Workshop on Applied Reconfigurable Computing (ARC)*, ser. Lecture Notes in Computer Science (LNCS). London, United Kingdom: Springer, Mar. 2008, pp. 287–293.

[116] J. Teich, L. Thiele, and L. Zhang, "Scheduling of partitioned regular algorithms on processor arrays with constrained resources," in *ASAP96- Proc. Int. Conf. on Application-Specific Systems, Architectures, and Processors*, Chicago, U.S.A., Aug. 1996, pp. 131–144.

[117] J. Teich, A. Tanase, and F. Hannig, "Symbolic mapping of loop programs onto processor arrays," *Journal of Signal Processing Systems*, pp. 31–59.

[118] Intel Corporation, "Intel xeon phi coprocessor datasheet," June 2013.

[119] D. Kissler, F. Hannig, A. Kupriyanov, and J. Teich, "A Highly Parameterizable Parallel Processor Array Architecture," in *Proceedings of the IEEE International Conference on Field Programmable Technology (FPT)*. IEEE, Dec. 2006, pp. 105–112.

[120] F. Hannig, V. Lari, S. Boppu, A. Tanase, and O. Reiche, "Invasive tightly-coupled processor arrays: A domain-specific architecture/compiler co-design approach," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 4s, pp. 133:1–133:29.

[121] D. Kissler, A. Strawetz, F. Hannig, and J. Teich, "Power-efficient reconfiguration control in coarse-grained dynamically reconfigurable architectures," *Journal of Low Power Electronics*, pp. 96–105, 2009.

[122] Intel Corporation, "Single-chip cloud computer (SCC)," www.intel.com/content/www/us/en/research/intel-labs-single-chip-cloud-overview-paper.html, 2009.

[123] R. Ubal, D. Schaa, P. Mistry, X. Gong, Y. Ukidave, Z. Chen, G. Schirner, and D. Kaeli, "Exploring the heterogeneous design space for both performance and reliability," in *51st Design Automation Conference (DAC)*, 2014, pp. 1–6.

[124] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Inc., 1990.

[125] P. Brommesson, "Solving the generalized assignment problem by column enumeration based on lagrangian reduced costs," 2006.

[126] J. Casazza, "First the tick, now the tock: Intel microarchitecture (nehalem)," Intel Corporation, White Paper, 2009.

[127] J. Henkel, T. Ebi, H. Amrouch, and H. Khdr, "Thermal management for dependable on-chip systems," in *18th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2013, pp. 113–118.

[128] D. P. Gulati, C. Kim, S. Sethumadhavan, S. W. Keckler, and D. Burger, "Multitasking workload scheduling on flexible-core chip multiprocessors," in *Parallel Architectures and Compilation Techniques (PACT), 2008 International Conference on*. IEEE, 2008, pp. 187–196.

[129] E. A. Burton, G. Schrom, F. Paillet, J. Douglas, W. J. Lambert, K. Radhakrishnan, and M. J. Hill, "Fivr; fully integrated voltage regulators on 4th generation intel; core; socs," in *2014 IEEE Applied Power Electronics Conference and Exposition - APEC 2014*, March 2014, pp. 432–439.

[130] C. Bienia and K. Li, "The parsec benchmark suite tutorial - parsec 2.0," May 2008.

[131] D. Pisinger, "Algorithms for knapsack problems," 1995.

List of Figures

List of Tables

Glossary

A | B | C | D | F | G | H | M | N | P | S | X

A

AgBoost Aging-aware boosting.

AgRM Aging-constrained resource management.

Alpha (Alpha 21264) Superscalar microprocessor with out-of-order execution.

ARM-A15 (ARM Cortex-A15) the big cores of ARM's "big.LITTLE" architecture.

ARM-A7 (ARM Cortex-A7) the small cores of ARM's "big.LITTLE" architecture.

B

blackscholes Application from the PARSEC benchmark suite.

bodytrack Application from the PARSEC benchmark suite.

C

canneal Application from the PARSEC benchmark suite.

D

dedup Application from the PARSEC benchmark suite.

DsRM Dark silicon-aware resource management.

F

ferret Application from the PARSEC benchmark suite.

G

gem5 Multicore simulator.

H

HotSpot Temperature modeling and computation tool.

M

McPAT Power consumption simulator.

N

nMOS N-type Metal-Oxide-Semiconductor Field-Effect Transistor.

P

PARSEC Benchmark suite for multi-threaded applications.

PdRM Power-density-aware resource management.

pMOS P-type Metal-Oxide-Semiconductor Field-Effect Transistor.

S

sGuard Selective guardbanding.

sGuard-DP Selective guardbanding based on dynamic programming.

sGuard-H Selective guardbanding based on heuristic.

SPICE General-purpose analog electronic circuit simulator.

swaptions Application from the PARSEC benchmark suite.

Synopsys Commercial tool for electronic design automation.

X

x264 Application from the PARSEC benchmark suite.

Acronyms

B | D | I | L | M | N | O | P | R | T | V

B

BOOM Berkeley out-of-order Machine.

BSIM Berkeley Short-channel IGFET Model.

BTI Bias Temperature Instability.

D

DPM Dynamic Power Management.

DTM Dynamic Thermal Management.

DVFS Dynamic Voltage and Frequency Scaling.

I

ILP Instruction Level Parallelism.

IPS Instructions Per Second.

L

LUT Look-up-table.

M

MOSFET Metal-Oxide-Semiconductor Field-Effect Transistor.

N

NBTI Negative Bias Temperature Instability.

O

O3 out-of-order.

P

PBTI Positive Bias Temperature Instability.

PE Processing Element.

PTM Predictive Technology Model.

R

RC Resistance-Capacitance.

T

TCPA Tightly-Coupled Processor Array.

TDP Thermal Design Power.

TIM Thermal Interface Material.

TLP Thread Level Parallelism.

TSAP Thermally-Safe Adaptive Power.

TSP Thermal Safe Power.

V

V/f level voltage and frequency level.

Symbols

A | B | D | F | G | H | K | L | M | N | P | Q | R | S | T | V | Z

A

A Matrix of application-to-tile assignment.

B

B Matrix of the thermal conductance values between thermal nodes.

D

d_k Selected number of dark cores of application k.

F

f^{gb} clock frequency of the core after applying the frequency guardband F_GB.

F_GB Frequency guardband type.

f^{nom} Nominal clock frequency of the core.

G

G Matrix of the values of the thermal conductance between each node and the ambient temperature.

GB Table of guardbands.

GLB Global Lower bound of performance.

$Gt_i^{\mathbf{core}}$ Selected guardband type for core i.

Gt_k^{app} Selected guardband type for application k.

H

h_k Selected number of threads of application k.

H_k Maximum thread-level parallelism of application k.

K

K Number of applications.

L

L Number of tiles on the chip.

LLB Local Lower bound of performance.

M

M Matrix of application-to-core mapping.

map_k Set of cores that application k is mapped to.

N

N Number of cores on the chip.

N_ℓ Number of cores in tile ℓ.

P

$Pd^{\mathbf{safe}}$ Thermally-safe power density constraint.

p_i Power consumption of core i.

P_k Power table of application k.

Q

q_k Selected power budget of application k.

R

R_k Performance table of application k.

S

S_i Selected power state for core i.

T

T Matrix of the core temperatures.

T_{amb} Ambient temperature.

t_{clk} Processor's clock delay after adding timing guardband t_{GB} specified at design time.

t_{clk}^{nom} Nominal processor's clock delay specified at design time.

$t_{critpath}$ Processor's critical path delay at runtime.

T_{crit} Critical temperature of the chip.

t_{GB} Timing guardband.

T_i Steady-state temperature of core i.

V

V_{dd} Supply voltage of a core.

V_{dd}^{max} Maximum supply voltage on the chip.

$vf_i^{\mathbf{core}}$ Selected V/f level for core i.

vf_k Selected V/f level for the cores of application k.

$vf_\ell^{\mathbf{tile}}$ Selected V/f level for tile ℓ.

$VF_{\ell,y}$ Maximum V/f level of tile ℓ.

VF_Y Maximum V/f level of the cores.

Y Number of available V/f levels on the chip.

V_{dd}^{gb} Supply voltage of the core after applying the voltage guardband V_GB.

V_GB Voltage guardband type.

V_{dd}^{nom} Nominal supply voltage of the core.

V_{th} Threshold voltage of transistors.

ΔV_{th}^{m} Constraint for the increase in the threshold voltage of transistors.

ΔV_{th} Increase in the threshold voltage of transistors.

Z

Z Number of the thermal nodes in the RC thermal Network.