Karlsruher Institut für Technologie

Power, Energy, and Thermal Management for Clustered Manycores

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

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

genehmigte Dissertation

von

Santiago Pagani

aus Buenos Aires, Argentinien

Tag der mündlichen Prüfung:	24.11.2016
Erster Gutachter:	Prof. Dr.-Ing. Jörg Henkel
Zweiter Gutachter:	Prof. Dr. Jian-Jia Chen
Dritter Gutachter:	Prof. Dr. Petru Eles

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my advisors Prof. Henkel and Prof. Chen, for all of their support during this long journey. Undertaking this Ph.D. has been a life-changing experience which was only made possible because they believed in me from the start, making me an active part of their research groups. They have shared with me their vast knowledge, continuously motivated me to continue on this path of becoming a researcher, given me the freedom and trust to pursue my own ideas and allowed me to fulfill them as I saw fit, and (most importantly) have offered me their time, which is our most precious commodity. Without their advise it would not have been possible for me to get this far.

Secondly, my sincere thanks also goes to (recently made) Prof. Shafique, who has also advised me greatly during the last two years of my Ph.D. research. I dearly appreciate all the help he has given me, and I send him my west wishes in the new path he has just started.

Thirdly, I would also like to thank the external reviewers and defense committee, particularly, Prof. Eles, Prof. Beigl, Prof. Karl, Prof. Bellosa, and Prof. Beckert, for their efforts and time invested in reviewing my dissertation and attending my defense.

Last but not least, I would like to thank my family. My beloved wife Carolina, who motivated me to pursue my Ph.D. in the first place, and who has been by my side ever since, during sleepless nights working to meet countless deadlines, far away from family and friends for years, just so I could do what was best for my career. Without her caring and support this would not have been possible. And my lovely son Maximiliano, for being such a good little rascal this past year, always putting a smile on our faces and brighting up our days.

List of publications

This dissertation summarizes the results published in several conferences, journals, workshops, and book chapters, as follows:

- [68] Santiago Pagani and Jian-Jia Chen. Energy efficiency analysis for the single frequency approximation (SFA) scheme. In *Proceedings of the 19th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 82–91, August 2013. **[Best Paper Award]**.
- [69] Santiago Pagani and Jian-Jia Chen. Energy efficient task partitioning based on the single frequency approximation scheme. In *Proceedings of the 34th IEEE Real-Time Systems Symposium (RTSS)*, pages 308–318, December 2013.
- [77] Santiago Pagani, Heba Khdr, Waqaas Munawar, Jian-Jia Chen, Muhammad Shafique, Minming Li, and Jörg Henkel. TSP: Thermal Safe Power - Efficient power budgeting for many-core systems in dark silicon. In *Proceedings of the 9th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 10:1–10:10, October 2014. **[Best Paper Award]**.
- [70] Santiago Pagani and Jian-Jia Chen. Energy efficiency analysis for the single frequency approximation (SFA) scheme. *ACM Transactions on Embedded Computing Systems (TECS)*, 13(5s):158:1–158:25, November 2014.
- [73] Santiago Pagani, Jian-Jia Chen, Muhammad Shafique, and Jörg Henkel. MatEx: Efficient transient and peak temperature computation for compact thermal models. In *Proceedings of the 18th Design, Automation and Test in Europe (DATE)*, pages 1515–1520, March 2015.
- [72] Santiago Pagani, Jian-Jia Chen, and Minming Li. Energy efficiency on multi-core architectures with multiple voltage islands. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 26(6):1608–1621, June 2015.
- [71] Santiago Pagani, Jian-Jia Chen, and Jörg Henkel. Energy and peak power efficiency analysis for the single voltage approximation (SVA) scheme. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 34(9):1415–1428, September 2015.
- [80] Santiago Pagani, Muhammad Shafique, Heba Khdr, Jian-Jia Chen, and Jörg Henkel. seBoost: Selective boosting for heterogeneous manycores. In *Proceedings of the 10th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 104–113, October 2015.
- [74] Santiago Pagani, Jian-Jia Chen, Muhammad Shafique, and Jörg Henkel. Thermal-aware power budgeting for dark silicon chips. In *Proceedings of the 2nd Workshop on Low-Power Dependable Computing (LPDC) at the International Green and Sustainable Computing Conference (IGSC)*, December 2015.
- [76] Santiago Pagani, Heba Khdr, Jian-Jia Chen, Muhammad Shafique, Minming Li, and Jörg Henkel. Thermal Safe Power (TSP): Efficient power budgeting for heterogeneous manycore systems in dark silicon. *IEEE Transactions on Computers (TC)*, 66(1):147–162, Jan 2017. **[Feature Paper of the Month]**.
- [78] Santiago Pagani, Anuj Pathania, Muhammad Shafique, Jian-Jia Chen, and Jörg Henkel. Energy efficiency for clustered heterogeneous multicores. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2017.
- [75] Santiago Pagani, Heba Khdr, Jian-Jia Chen, Muhammad Shafique, Minming Li, and Jörg Henkel. Thermal Safe Power: Efficient thermal-aware power budgeting for manycore systems in dark silicon. In Amir M. Rahmani, Pasi Liljeberg, Ahmed Hemani, Axel Jantsch, and Hannu Tenhunen, editors, *The Dark Side of Silicon*. Springer, 2017.

Furthermore, following is a list of other co-authored works:

- [40] Janmartin Jahn, Sebastian Kobbe, Santiago Pagani, Jian-Jia Chen, and Jörg Henkel. Work in Progress: Malleable software pipelines for efficient many-core system utilization. In *Proceedings of the 6th Many-core Applications Research Community (MARC) Symposium at ONERA*, pages 30–33, July 2012.
- [41] Janmartin Jahn, Sebastian Kobbe, Santiago Pagani, Jian-Jia Chen, and Jörg Henkel. Runtime resource allocation for software pipelines. In *Proceedings of the 16th International Workshop on Software and Compilers for Embedded Systems (M-SCOPES)*, pages 96–99, June 2013.
- [43] Janmartin Jahn, Santiago Pagani, Sebastian Kobbe, Jian-Jia Chen, and Jörg Henkel. Optimizations for configuring and mapping software pipelines in manycore. In *Proceedings of the 50th IEEE/ACM Design Automation Conference (DAC)*, pages 130:1–130:8, June 2013. **[HiPEAC Paper Award]**.
- [42] Janmartin Jahn, Santiago Pagani, Jian-Jia Chen, and Jörg Henkel. MOMA: Mapping of memory-intensive software-pipelined applications for systems with multiple memory controllers. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 508–515, November 2013.
- [62] Waqaas Munawar, Heba Khdr, Santiago Pagani, Muhammad Shafique, Jian-Jia Chen, and Jörg Henkel. Peak power management for scheduling real-time tasks on heterogeneous many-core systems. In *Proceedings of the 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, December 2014.
- [44] Janmartin Jahn, Santiago Pagani, Sebastian Kobbe, Jian-Jia Chen, and Jörg Henkel. Runtime resource allocation for software pipelines. *ACM Transactions on Parallel Computing (TOPC)*, 2(1):5:1–5:23, May 2015.
- [48] Heba Khdr, Santiago Pagani, Muhammad Shafique, and Jörg Henkel. Thermal constrained resource management for mixed ILP-TLP workloads in dark silicon chips. In *Proceedings of the 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 179:1–179:6, June 2015. **[HiPEAC Paper Award]**.
- [27] Jörg Henkel, Heba Khdr, Santiago Pagani, and Muhammad Shafique. New trends in dark silicon. In *Proceedings of the 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 119:1–119:6, June 2015. **[HiPEAC Paper Award]**.
- [81] Anuj Pathania, Santiago Pagani, Muhammad Shafique, and Jörg Henkel. Power management for mobile games on asymmetric multi-cores. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, pages 243–248, July 2015.
- [28] Jörg Henkel, Santiago Pagani, Heba Khdr, Florian Kriebel, Semeen Rehman, and Muhammad Shafique. Towards performance and reliability-efficient computing in the dark silicon era. In *Proceedings of the 19th Design, Automation and Test in Europe (DATE)*, March 2016.
- [101] Anas Toma, Santiago Pagani, Jian-Jia Chen, Wolfgang Karl, and Jörg Henkel. An energy-efficient middleware for computation offloading in real-time embedded systems. In *Proceedings of the 22nd IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, August 2016.
- [49] Heba Khdr, Santiago Pagani, Éricles Sousa, Vahid Lari, Anuj Pathania, Frank Hannig, Muhammad Shafique, Jürgen Teich, and Jörg Henkel. Power density-aware resource management for heterogeneous tiled multicores. *IEEE Transactions on Computers (TC)*, 2016.
- [67] Santiago Pagani, Lars Bauer, Qingqing Chen, Elisabeth Glocker, Frank Hannig, Andreas Herkersdorf, Heba Khdr, Anuj Pathania, Ulf Schlichtmann, Doris Schmitt-Landsiedel, Mark Sagi, Éricles Sousa, Philipp Wagner, Volker Wenzel, Thomas Wild, and Jörg Henkel. Dark silicon management: An integrated and coordinated cross-layer approach. *Special Issue of (De Gruyter Oldenbourg) Information Technology on Invasive Computing*, 2017.
- [79] Santiago Pagani, Muhammad Shafique, and Jörg Henkel. Design space exploration and run-time adaptation for multi-core resource management under performance and power constraints. In Soonhoi Ha and Jürgen Teich, editors, *Handbook of Hardware/Software Codesign*. Springer, 2017.

Abstract

Efficient and effective system-level power, energy, and thermal management are very important issues in modern computing systems, e.g., to reduce the packaging cost, to prolong the battery lifetime of embedded systems, or to avoid the chip from possible overheating. These are some of the main motivations why computing systems have moved from single-core to multicore/manycore platforms, mainly to balance the power consumption and computation performance. Furthermore, clustered architectures with multiple voltage islands, where the voltage on a cluster can change independently and all cores in a cluster share the same supply voltage at any given time, are an expected compromise between global and per-core Dynamic Voltage and Frequency Scaling (DVFS) for modern manycore systems. In this dissertation, we focus on two of the most relevant problems for such architectures, specifically, optimizing performance under power/thermal constraints, and minimizing energy under performance constraints.

For performance optimization, we first present a novel thermal-aware power budgeting concept, called Thermal Safe Power (TSP), which is an abstraction that provides safe power and power density constraints as a function of the number of active cores. TSP conceptually changes the typical design that uses a single and constant value as power budget, e.g., the Thermal Design Power (TDP), and can also serve as a fundamental tool for guiding task partitioning and core mapping decisions. Secondly, we show that runtime decisions typically used to optimize resource usages (e.g., task migration, power gating, DVFS, etc.) can result in transient temperatures much higher than the normally considered steady-state scenarios. In order to be thermally safe, it is important to evaluate the transient peaks before making resource management decisions. To this end, we present a lightweight method for computing these transient peaks, called MatEx, based on analytically solving the system of thermal differential equations by using matrix exponentials and linear algebra, instead of using regular numerical methods. Thirdly, we present an efficient and lightweight runtime boosting technique based on transient temperature estimation, called seBoost. Traditional boosting techniques select the boosting levels (for boosted cores) and the throttle-down levels (for non-boosted cores) arbitrarily or through step-wise control approaches, and might result in unnecessary performance losses for the non-boosted cores or may fail to satisfy the required runtime performance surges. Contrarily, seBoost relies on MatEx to select the boosting levels, and thus it guarantees meeting the required runtime performance surges, while maximizing the boosting time with minimum performance losses for the non-boosted cores.

In regards to energy minimization, we first focus on a single cluster, and we propose to use the Double Largest Task First (DLTF) strategy for partitioning tasks to cores based on load balancing and idle energy reduction, combined with either the Single Frequency Approximation (SFA) scheme or the Single Voltage Approximation (SVA) scheme for deciding the DVFS levels for execution. Furthermore, we provide thorough theoretical analysis of both solutions, in terms of energy efficiency and peak power reduction, against the optimal task partitioning and optimal DVFS schedule, especially for the state-of-the-art designs, that have a limited number of cores inside each cluster. In SFA, all the cores in a cluster run at a single voltage and frequency, such that all tasks meet their performance constraints. In SVA, all the cores in a cluster also run at the same single voltage as in SFA; however, the frequency of each core is individually chosen, such that the tasks in each core can meet their performance constraints, but without running at unnecessarily high frequencies. Finally, we extend our analysis for systems with multiple clusters, and present two task-to-core mapping solutions when using SFA on individual clusters, specifically, a dynamic programming algorithm that derives optimal solutions for homogeneous manycores, and a lightweight and efficient heuristic for heterogeneous manycores.

Contents

Chapter 1

Introduction

Over the past decade, single-core systems have reached a practical upper-limit in regards to their maximum operational frequency, mostly due to power dissipation. This is one of the main motivations why chip manufacturers shifted their focus towards designing chips with multiple cores that operate at lower voltages and frequencies than their single-core counterparts, potentially achieving the same computational performance while consuming less power. Moreover, the performance demands of modern applications have considerably increased and can no longer be satisfied only by raising the frequency of a single-core system. Therefore, modern computing systems require many cores in the same chip, and their number is expected to increase every year [39].

The ever-increasing transistor integration and the observed limits on voltage scaling for next-generation technology nodes are starting to result in high power densities and temperatures on manycore systems, which are the causes behind the emerging *dark silicon problem* [95]. Particularly, when a common cooling solution is used for several scaling generations (i.e., the cooling costs are kept constant), all the cores on a chip can no longer be simultaneously active at the nominal operation levels without violating the chip's thermal constraints [19, 95, 99]. Such an effect challenges the viability of further cost-effective technology scaling, given that it can slow down the current performance gain trends between generations [27]. Therefore, efficient and effective power and thermal management techniques are now all the more relevant, especially for optimizing performance, as they promise to maintain technology scaling trends feasible, without incurring in high cooling costs, while avoiding the chip from possible overheating. Furthermore, in order to prolong the battery lifetime of embedded systems, or to cut the power bills in servers, energy management for energy minimization under performance (or real-time) constraints is another relevant (almost dual) problem.

In this dissertation, we focus on two of the most relevant problems related to power management on multicore and manycore systems. Specifically, one part of the dissertation focuses on maximizing/optimizing computational performance under power or thermal constraints, while another part focuses on minimizing energy consumption under performance (or real-time) constraints.

1.1 Optimization Goals and Constraints

1.1.1 Computational Performance

Computational performance refers to how quickly the system can execute an application or a given set of applications. The performance of an application can be measured in, e.g., execution time, throughput, Instructions per Cycle (IPC), Instructions per Second (IPS), speed-up factor (normalized to a known reference), etc. For a set of applications, the term *overall system performance* is commonly used, which is a generic term that can, e.g., refer to maximizing the summation of the weighted throughput of all applications (weights add some sort of priority to the applications), minimize the longest execution time among all applications (i.e., the *makespan*), etc.

The resulting performance of an application depends on how the application is executed, e.g., in how many threads the application is parallelized, the types of cores to which the application is mapped, how other applications are simultaneously using the shared resources (e.g., caches, the Network on Chip (NoC), etc.), the execution frequency of the cores, etc. The characteristics of individual applications also play a major role in its resulting performance, e.g., their Thread-Level Parallelism (TLP) or their Instruction-Level Parallelism (ILP), and directly impact how an application's performance scales in regards to the number of parallel threads or executed frequency. Applications with high TLP normally scale well when they are parallelized in many threads, while applications with high ILP normally scale well with increasing frequencies. For example, Figure 1.1 shows the total number of executed cycles, total execution time, and speed-up factors, of one instance of five applications from the PARSEC benchmark suite [4] (with *simsmall* input) with respect to the execution frequency, when executing a single thread on different types of cores.

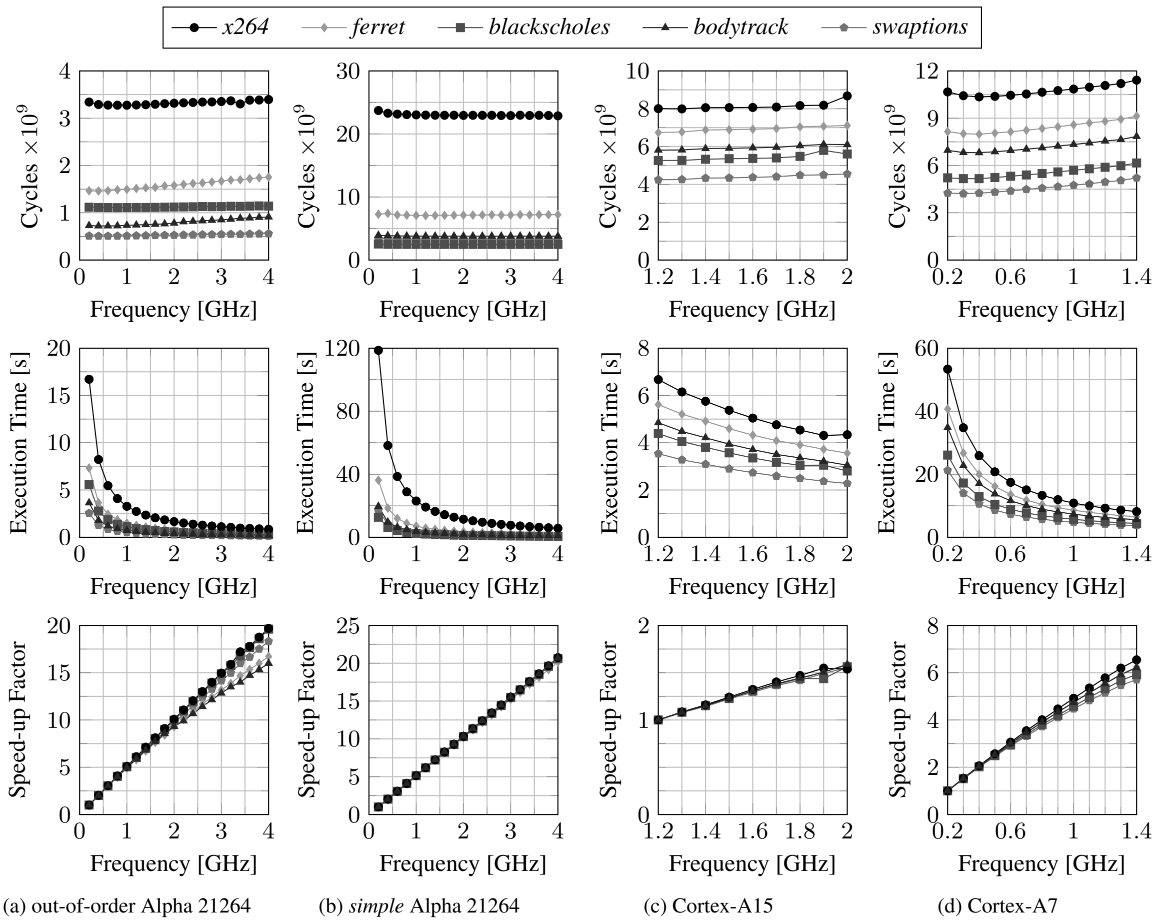


Figure 1.1: Execution cycles, execution time, and speed-up factors (normalized to the execution time of each application running at the lowest frequency on each type or core) of one instance of five applications from the PARSEC benchmark suite [4], executing a single thread with *simsmall* input, based on simulations in gem5 [5] and McPAT [57] (for 22 nm), and measured on the Exynos 5 Octa (5422) processor [92].

Maximizing the *overall system performance* is generally the most commonly pursued optimization goal. Nevertheless, for applications with real-time deadlines, meeting the deadlines can be formulated as satisfying performance requirements, and therefore for such cases performance is considered as a constraint.

1.1.2 Power and Energy Consumption

Every core doing some computation consumes power (unit *Watt* $[W]$), and this power consumption produces heat. Power consumption is an instantaneous metric that changes through time. Particularly, a core executing a certain application thread will consume different amounts of power at different time instants. For example, Figure 1.2 shows the resulting power consumption values of simulations conducted with Sniper [8] and McPAT [57], for a PARSEC *bodytrack* application executing 4 parallel threads on a quad-core Intel Nehalem cluster running at 2.2 GHz. The power consumption observed on a core at a given time point depends on several parameters, e.g., the technology node, the underlying architecture of the core, the execution mode (e.g., active, idle, sleep, etc.), the voltage and frequency settings, the current temperature, the current application phase, etc.

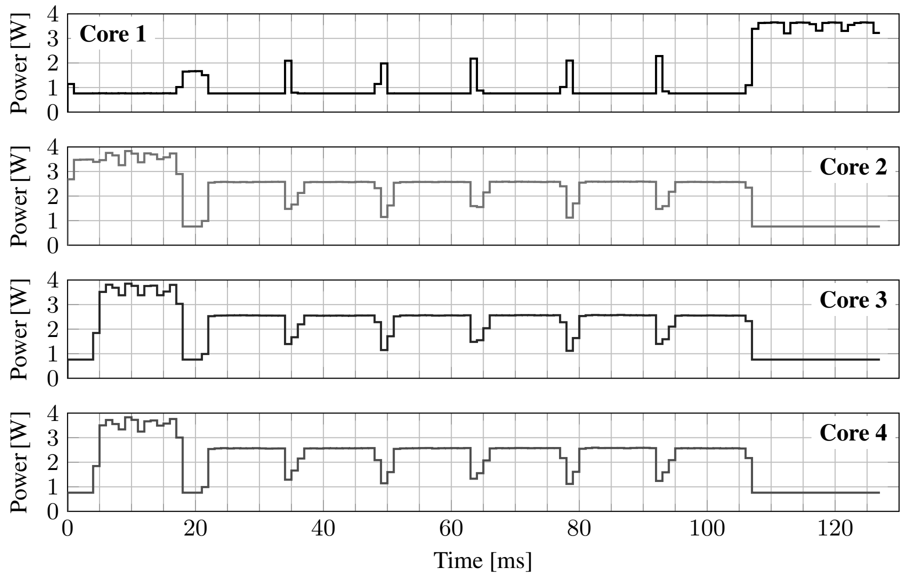


Figure 1.2: Power consumption of a PARSEC *bodytrack* application with *simsmall* input, executing 4 threads at 2.2 GHz on a quad-core Intel Nehalem cluster, based on simulations using Sniper [8] and McPAT [57].

On the other hand, energy is the integration of power over time (units *Joule* $[J]$ or *Watt second* $[Ws]$). Namely, when plotted, the energy consumed between two time points is equal to the area below the power curve between those two time points. Hence, energy can be associated to a time window, e.g., an application instance. For example, when the power consumption between two time points remains constant, the energy consumed between those points can be simply computed by multiplying the power consumption by the time elapsed between the points. Figure 1.3 presents average power and energy consumption values for one instance of five applications from the PARSEC benchmark suite [4] executing a single thread on an out-of-order (OOO) Alpha 21264 core.

Minimizing the overall energy consumption under performance or timing constraints is a common optimization goal for mobile embedded systems that desire to prolong the battery lifetime. Contrarily, it is rare to optimize for power consumption, and thus power is mostly considered as a constraint, for example, to run the system under a given per-chip power budget.

1.1.3 Temperature

As mentioned in Section 1.1.2, when some part of the chip consumes power, it also generates heat. However, although the changes in power consumption can be considered to be instantaneous for practical purposes, changes in temperature do not occur instantaneously, as there is a thermal capacitance associated to every element on a chip. After enough time elapses without a change in power consumption (depending on the values of the thermal capacitances), the temperature throughout the chip eventually reaches its *steady-state*. The intermediate temperature values that are visible until the steady-state temperatures are achieved are defined as the *transient temperatures*, as shown in the example in Figure 1.4.

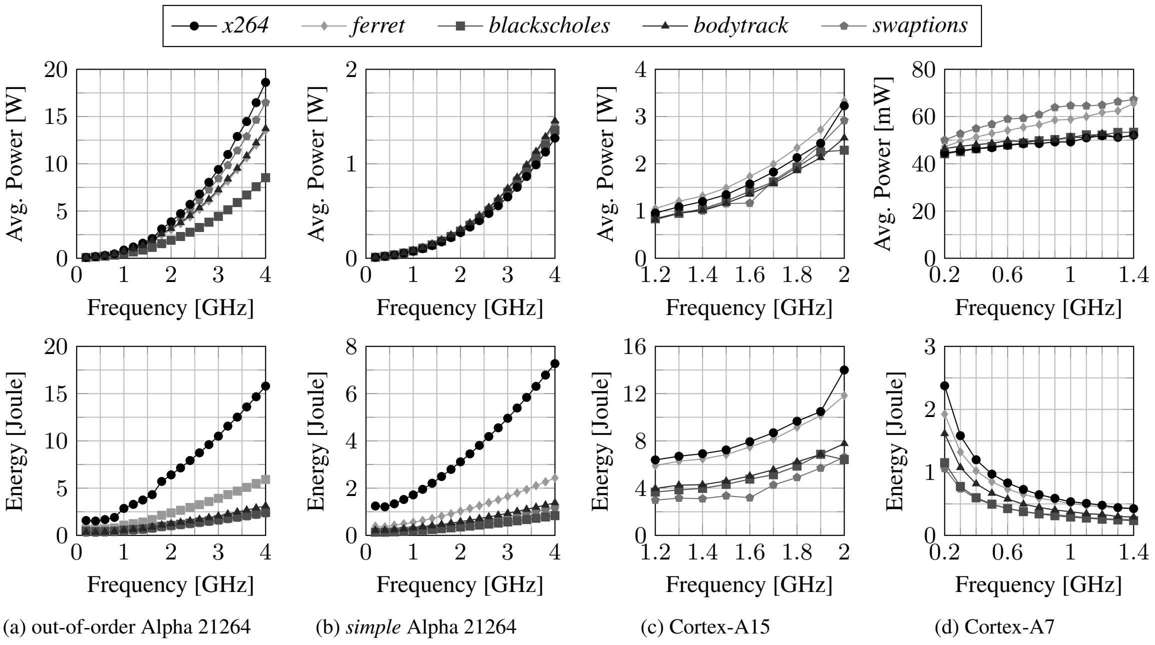


Figure 1.3: Average power and energy consumption values of one instance of five applications from the PARSEC benchmark suite [4] executing a single thread with *simsmall* input, based on simulations in gem5 [5] and McPAT [57] (for 22 nm), and measured on the Exynos 5 Octa (5422) processor [92].

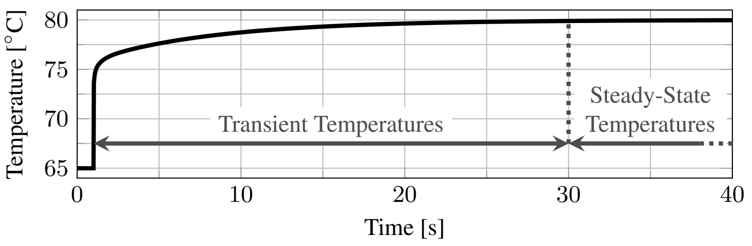


Figure 1.4: Example of transient and steady-state temperatures for a change in power at $t = 1$ s.

Maintaining the temperature throughout the chip under a certain critical value (or thermal threshold) is of paramount importance. Otherwise, excessively high temperatures on a chip may cause permanent failures in transistors. In order to dissipate this power and temperature, chips are provided with a cooling solution (e.g., the combination of the thermal paste, the heat spreader, the heat sink, the cooling fan, etc.). To aid in the design of such a cooling solution, the common industry practice is to provide system designers with the Thermal Design Power (TDP) of a specific chip, defined as "the highest expected sustainable power while running known power intensive real applications" [34]. Therefore, since TDP should be a safe power level to run the system, manufacturers generally recommend to design the cooling solution to dissipate TDP, such that the cooling solution is not over-dimensioned. However, given that TDP is not the maximum achievable power, chips are also normally provided with some kind of Dynamic Thermal Management (DTM) technique. DTM techniques are mostly reactive (i.e., are triggered after the critical temperature is reached or exceeded) and can power-down cores, gate their clocks, reduce their supply voltages and execution frequencies, boost-up the fan speed, etc. Namely, if some part of the chip heats up above a critical value (condition identified using temperature sensors distributed on the chip), then DTM is triggered in order to reduce the temperature. An abstract example of a standard reactive control-based closed-loop DTM technique [34] based on voltage and frequency scaling can be seen in Figure 1.5.

Thermal constraints are generally the biggest limiting factor for performance optimization, especially in modern computing platforms with very high power densities due to the dark silicon problem.

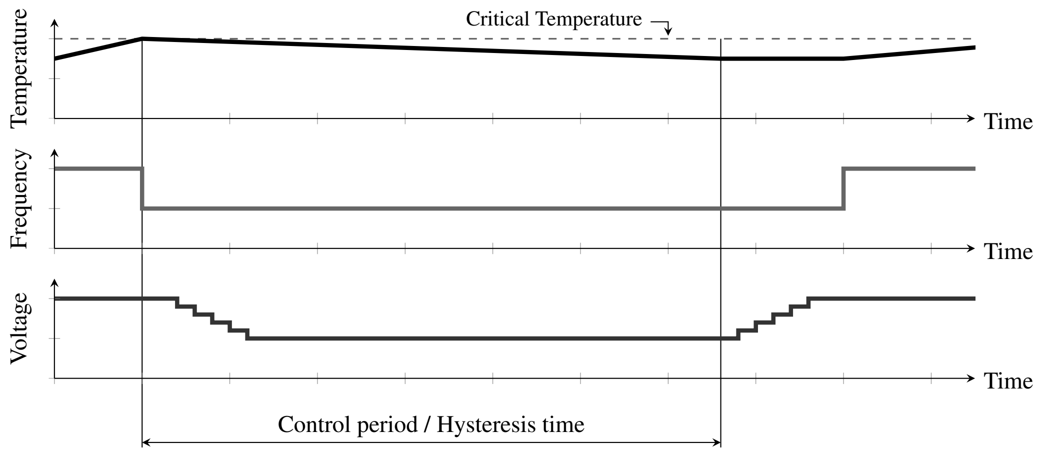


Figure 1.5: Abstract example of a standard reactive control-based closed-loop DTM technique based on voltage and frequency scaling. When the critical temperature is violated in some part of the chip, the voltage and frequency levels of all cores are reduced. After a control period (or hysteresis time), and if the maximum temperature in the chip is below the critical value, the voltage and frequency levels of the cores can be brought back to the nominal operation settings.

1.2 Optimization Knobs

1.2.1 Core Heterogeneity

The continuous increase in power density and performance demands have led to the emergence of heterogeneous multicore and manycore systems composed by different types of cores, where each type of core may have different performance, power, and energy characteristics [98], as already seen in the examples on Figure 1.1 and Figure 1.3. The distinct architectural features of each type of core can be thus potentially exploited to meet the system's goals and to satisfy its constraints (e.g., computational performance, power consumption, energy consumption, temperature, etc.). Therefore, for power and energy efficiency, heterogeneous multicore and manycore systems are a promising alternative over their homogeneous counterparts, as an application may witness large improvements in performance and/or power when mapped to an appropriate type of core. An example of a heterogeneous system is the Exynos 5 Octa (5422) processor (simple block diagram shown in Figure 1.6) based on ARM's big.LITTLE architecture [21].

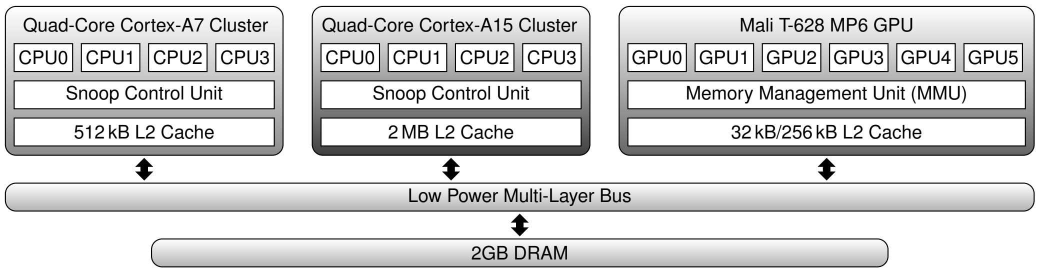


Figure 1.6: Exynos 5 Octa (5422) processor based on ARM's big.LITTLE architecture.

1.2.2 Task-to-core Assignment/Mapping

Task-to-core assignment (or mapping) involves deciding to which specific core a thread is mapped to, both in terms of the type of core and the physical location of the core in the chip. Namely, due to the characteristics of the different cores (as explained in Section 1.2.1), the type of the core to which a thread is mapped to affects its resulting execution time and power/energy consumption. Nevertheless, the physical location to which a thread is mapped to is also a non-trivial issue, given that this impacts the execution time (due to communication latencies among cores, potential link congestions, simultaneous utilization of shared resources by other threads, etc.), as well as on the resulting temperature distribution throughout the chip (due to the heat transfer that occurs among cores, which can potentially create or avoid hot spots).

1.2.3 Dynamic Power Management (DPM)

Dynamic Power Management (DPM) refers to dynamically selecting the power states of individual cores. In general terms, cores could be individually set to the execution/active mode, or they could be set to some low-power mode when they are inactive (e.g., idle/clock-gated, sleep, power-gated, etc.). The available number of low-power modes depends on the chip, and each different low-power mode has associated a power consumption, as well as different latencies for transitioning from one mode to another.

1.2.4 Dynamic Voltage and Frequency Scaling (DVFS)

Dynamic Voltage and Frequency Scaling (DVFS) refers to the ability of dynamically scaling the voltage and/or frequency of cores. The voltage value supplied to a core limits the maximum frequency at which it can be stably executed. Higher voltages allow for stable execution at higher frequencies. As shown in [82], the relationship between the supply voltage of a core and the maximum frequency for stable execution can be modeled according to Equation (1.1),

$$f_{\text{stable}} = k \cdot \frac{(V_{\text{dd}} - V_{\text{th}})^2}{V_{\text{dd}}} \tag{1.1}$$

where V_{dd} is the supply voltage of the core, f_{stable} is the maximum stable frequency, V_{th} is the threshold voltage for the given technology, and k is an architecture-dependent fitting factor. For a given V_{dd}, running at frequencies lower than f_{stable} is stable, but power/energy inefficient, and therefore generally avoided (further details about this statement are later presented in Chapter 3.4). Figure 1.7 uses Equation (1.1) to model the voltage and frequency relationship necessary for stable execution on a 28 nm x86-64 microprocessor [22].

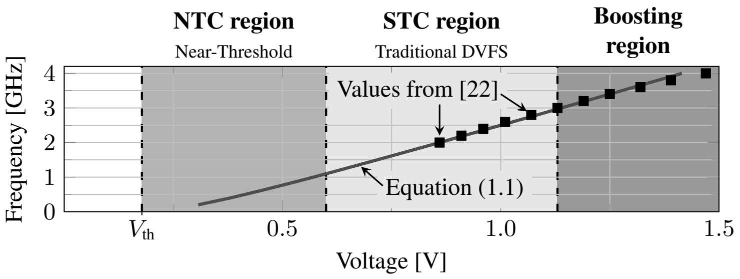


Figure 1.7: Supply voltage vs. maximum stable frequency relationship, modeled with Equation (1.1) for the experimental results of a 28 nm x86-64 microprocessor developed in [22].

The granularity at which voltage scaling and frequency scaling are available may vary across chips. For example, we could have:

- **Global voltage and frequency scaling:** All cores in the chip share a common voltage and frequency.
- **Global voltage scaling:** All cores in the chip share a common voltage, but individual cores can change their frequencies independently.
- **Voltage and frequency islands/clusters:** Different groups of cores share a common voltage and frequency.
- **Voltage islands/clusters:** Different groups of cores share a common voltage, but individual cores can change their frequencies independently.
- **Per-core voltage and frequency scaling:** The voltage and frequency of all cores is selected individually.

Given that there is a quadratic relationship between the power consumption of a core and its supply voltage (explained in detail later in Chapter 3.3), having a system with only one global supply voltage for all cores can be power/energy/thermal inefficient. The reason behind this inefficiency is that the voltage of all cores in the chip is then determined by the core requiring the highest frequency, while other cores requiring lower frequencies are also forced to execute at a high voltage, thus consuming more power/energy and producing more heat than necessary. On the other hand, having an individual voltage for every core can be very power/energy/thermal efficient, since every core could be supplied with the lowest stable voltage for its required frequency. However, based on Very Large-Scale Integration (VLSI) circuit simulations, it has been suggested in [29] that it may be costly for implementation as it can suffer from complicated design problems. Therefore, cluster-based architectures with multiple voltage islands are a promising compromise for multicore and manycore systems, as different clusters/islands can run at different voltages at any time point. For example, in Intel's Single Chip Cloud computer (SCC) [36], cores are clustered into groups of eight cores sharing a common voltage, while the frequency can be selected for every tile of two cores, such that there can be up to four different frequencies inside every cluster of eight cores sharing a voltage. On the other hand, in the Exynos 5 Octa (5422) processor [92] (Figure 1.6), all the cores inside every cluster share both their voltage and frequency, and therefore only frequency settings are exposed to the software level, while the voltage is automatically selected according to the chosen frequency.

1.3 Performance Optimization under Power or Thermal Constraints

As explained in Section 1.1.1, maximizing the overall system performance is one of the most commonly pursued optimization goals on manycore systems. However, the temperature on the chip needs to remain below safe margins at all times without incurring in high cooling costs. For such a purpose, the most common approaches are to directly use temperature as a constraint, or to consider some kind of power budgets (either at a per-chip or per-core level). In either case, there is also a physical power constraint on every chip (not an abstraction like a power budget) which can, e.g., be determined by the inner chip's wire thickness or by the supply voltage.

Power budgets are abstractions that allow system designers to indirectly handle temperature issues. The main idea is that running the system under a given power budget should presumably avoid violations of the temperature constraints. In line with this idea, a power budget aimed to serve as a thermal abstraction can be considered *safe*, when satisfying the power budget guarantees the satisfaction of the thermal constraints. Furthermore, a power budget can be considered *efficient* if, when consuming the entire budget, the resulting peak temperature is very close to the critical temperature, i.e., there is no unnecessary thermal headroom. An abstract example of a *safe* and *efficient* power budget is shown in Figure 1.8, in which, when the system consumes the entire power budget, the maximum temperature among all elements in the chip at any time point remains just under the critical temperature.

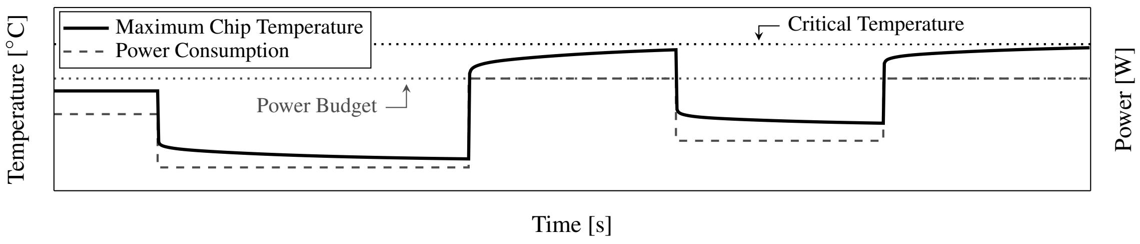


Figure 1.8: Abstract example of a *safe* and *efficient* power budget.

Optimizing performance under a power budget constraint therefore involves assigning threads/tasks to cores considering the core heterogeneity available on the chip, and then applying DPM and DVFS. Developing algorithms to solve such a problem is potentially less complex than directly dealing with temperature, because of two main reasons. First, transient temperature effects are implicitly ignored, and satisfying the power budget should be thermally safe from a transient perspective. Secondly, the heat transfer among cores is also implicitly ignored, meaning that this approach mostly deals with task-to-core *assignment*, rather than task-to-core *mapping*. That is, the physical location of the cores to which tasks are assigned is mostly relevant due to communication costs/latencies, but not due to thermal effects. Therefore, as long as the power budgets are satisfied, it should not matter whether all tasks are mapped to cores close together in a concentrated manner, or if the mapping is evenly distributed across the chip.

Optimizing performance using power budgets as constraints is potentially simpler than directly dealing with temperature by considering thermal constraints. Nevertheless, the reduced complexity should be transfered to the techniques that derive such power budgets. Otherwise, using very simplistic power budgets would result in poor performance and/or thermally unsafe behavior. In other words, efficient performance optimization under power budget constraints should be intended to partition the problem in two subproblems, and not merely to reduce the overall complexity.

The major issue with techniques based on power budget constraints is how to guarantee whether the power budget is met or not. In practice, if the chip is equipped with one power meter for every core, meeting a power budget can be simply achieved by measuring the actual power consumption on every core and acting accordingly, e.g., applying DVFS such that the measured power is as close as possible to the power budget but without exceeding it, thus, running as fast as possible for optimizing the core's performance. In case that power meters are not available in the hardware or in case their granularity is not suitable (e.g., only one power meter for the entire chip), then design-time application profiling (e.g., [48]) or runtime power estimation through performance counters (e.g., [38, 58, 83, 106]) are reasonable alternatives as an intermediate step to relate core settings (i.e., DPM states and DVFS levels) and power consumption values.

The alternative to using power budgets as constraints is to optimize performance by dealing directly with temperature, either by using thermal sensors or through thermal models. In comparison to using power budget constraints, directly dealing with temperature accounts for the effects of heat transfer among cores (hence, the location of active cores *does* matter) and transient temperatures, thus avoiding possible pessimistic or unsafe scenarios that can exist when using power budgets as thermal abstractions.

Using thermal sensors can generally be more accurate and easier to implement than using thermal models, since it is very common to find several thermal sensors inside modern chips, possibly even one thermal sensor for every core, as is the case in the Exynos 5 Octa (5422) processor [92]. Nevertheless, the problem with such an approach is that doing proactive management without using some kind of thermal model to predict future temperatures is not feasible. Therefore, thermal management techniques that only rely on thermal sensors are generally reactive in nature, avoiding thermal violations or exploiting the available thermal headroom at runtime, with little involvement in terms of predictability or timing guarantees.

On the other hand, optimizing performance using thermal models is more complex and challenging than using thermal sensors or power budget constraints. However, it enables efficient and thermally safe *proactive* management, as thermal models can predict the future thermal behavior (not possible when only using thermal sensors) while accounting for the effects of heat transfer among cores and transient temperatures (not possible on power budgeting approaches). Similar to any other approach based on modeling, it can be potentially affected by accuracy losses, e.g., due to modeling errors or the dynamic behavior of applications. Nevertheless, thermal modeling errors can normally be handled by using some modeling method that relies on real measurements on a specific chip and cooling solution, e.g., [17]. Moreover, such an option is also well suited to deal with process variations, heat sink imperfections, aging effects, multiple fan speeds, etc.

1.4 Energy Minimization under Performance Constraints

For real-time tasks with hard deadlines or for performance-constrained applications, rather than maximizing performance, the main objective is to guarantee that all applications meet their timing/performance requirements/constraints. If all timing and performance constraints are satisfied, then the system can focus on further optimization, such as minimizing the overall energy consumption to prolong the battery lifetime of embedded systems or to cut the power bills in servers. Minimizing energy under performance constraints involves partitioning the threads/tasks into task sets (such that several tasks can be assigned to the same core, handled with preemption, but having no more task sets than cores on the chip), assigning every task set to a core/cluster, and then applying a DPM and DVFS policy on each core and cluster. Even for chips with no DVFS capabilities, obtaining the *optimal task partitioning* for energy minimization, i.e., the task-to-core assignment that results in the lowest possible energy consumption, is already an $\mathcal{NP}$-hard problem [3, 18].

Minimizing energy is related to minimizing average power consumption; however, it is not precisely equivalent. For example, as seen in Figure 1.1 and Figure 1.3, when the DVFS level of a core increases, generally, the average power consumption of the core raises and the application's execution time is reduced. Given that energy is the integration of power through time (i.e., for a constant power, it can be computed by multiplying power with time), when the ratio of increase in average power consumption is larger than the decrease in the execution time, executing at low DVFS levels results in low energy consumptions. However, when the opposite happens, decreasing the DVFS levels will not further reduce the energy consumption, as in such cases the increase of the execution time becomes more significant than the reduction of the average power consumption (normally due to high leakage power consumptions). This effect can for example be observed in Figure 1.3b and Figure 1.3d for *x264*, where it is not energy efficient to execute applications at frequencies lower than $0.4\,\mathrm{GHz}$ and $1.4\,\mathrm{GHz}$, respectively.

Therefore, minimizing energy under performance constraints is tightly related to maximizing performance under power and thermal constraints. Particularly, they are almost dual problems, both using the same optimization knobs and requiring a very similar background knowledge. Nevertheless, due to the above mentioned differences, they should be approached as separate problems.

1.5 Summary of the State-of-the-art, Problems, and Challenges

For **performance optimization under power budget constraints**, the most common and traditional approach is to consider a *single and constant per-chip power budget* as an abstraction from thermal problems. The main idea is that the summation of the power consumptions of all individual elements on the chip should not exceed the per-chip budget. Particularly, the most widely used per-chip power budget is TDP, and there are several researches in the literature aiming at performance optimization for such a case [54, 64, 87, 93]. A similar approach is to have a *single and constant per-core power budget*, where the power consumption of individual cores cannot exceed the per-core budget. The value of the per-core power budget for such cases could be computed by dividing TDP by the number of cores on the chip. However, using a single and constant value as a power constraint, either at a per-chip or per-core level, can easily result in thermal violations or significantly underutilized resources on multicore/manycore systems (a motivational example justifying this statement is later presented in Chapter 5.1.1). The challenge here is therefore to derive a novel thermal-aware power budgeting technique that does not result in thermal violations or underutilized resources.

With respect to **performance optimization under thermal constraints based on thermal modeling**, most state-of-the-art techniques have high time complexity, making them well suited for design-time decisions (e.g., [25, 48]), but not so well suited for runtime thermal management, especially when considering transient thermal effects that can produce transient temperatures that are higher than the associated steady-state values (a motivational example about this issue is later presented in Chapter 6.1.1). The challenge here is thus to find methods to speed-up current thermal management techniques, e.g., with the same thermal-aware power budgeting techniques as mentioned above, and with lightweight techniques to estimate whether a mapping/scheduling decision might result in a thermal violation due to transient temperatures.

For **performance optimization under thermal constraints using thermal sensors**, *boosting techniques* have been widely adopted in commercial multicore and manycore systems, e.g., Intel's Turbo Boost [9, 10, 35, 91] and AMD's Turbo CORE [66]. Specifically, boosting techniques leverage any available thermal headroom at runtime by allowing the system to execute cores at high DVFS levels during short time intervals while ignoring the standard operating power budgets, e.g., TDP. Given that running cores at high DVFS levels increases their power consumption, applying boosting will normally result in an increment of the temperature through time. Therefore, after the temperature in some part of the chip reaches the critical value, the system must return to the nominal operation levels (requiring some cool-down time before boosting again), or use a closed-loop control-based approach to oscillate around the critical temperature (thus prolonging the boosting time indefinitely). The problem with these kinds of techniques is that, due to their reactive nature and lack of thermal modeling, they are unsuitable for providing predictability or timing/performance guarantees, and these are challenges for new boosting techniques.

In regards to **energy minimization under performance constraints (or hard real-time constraints)**, task scheduling, task partitioning, and DVFS policies have been explored in the academia and industry in the past decades. Most researches assume to have either per-core DVFS (e.g., [3, 12, 14, 108]), or global DVFS (e.g., [15, 50, 94, 109]). However, aside from a few examples (e.g., [18, 24, 53, 65, 107]), there is not much research for cluster-based architectures with multiple voltage islands. Given that obtaining the *optimal task partitioning* for energy minimization is an $\mathcal{NP}$-hard problem [3, 18], most task partitioning strategies are based on polynomial time greedy algorithms like the Largest Task First (LTF) strategy [109]. With respect to DVFS, once tasks are assigned to cores and clusters, it is necessary to choose a policy that selects the voltage of the cluster and the frequencies of the cores. For energy minimization of periodic real-time tasks or performance-constrained applications, the simplest and most intuitive scheme (adopted by several researchers in the past, e.g., [15, 65]) is to use a single voltage and frequency for execution, specifically, the lowest voltage and frequency that satisfies the timing and performance constraints, denoted as the Single Frequency Approximation (SFA) scheme. Combining a task partitioning strategy like LTF with a DVFS policy like SFA is clearly not the optimal solution for energy minimization; however, it is a practical and easy to implement approach, with little or no DVFS overheads at runtime. Nevertheless, deriving the worst-case efficiency in terms of energy consumption for combining these two schemes is an open problem. Furthermore, when also considering core heterogeneity, the energy consumption and execution time of a task changes both with the DVFS levels and the type of the core to which the task is assigned to, as already observed in Figure 1.1 and Figure 1.3. Therefore, state-of-the-art solutions for homogeneous systems [24, 53, 65, 107] will not derive good solutions for the heterogeneous case, while state-of-the-art solutions for heterogeneous systems [18, 63] are not yet in a mature state.

1.6 Dissertation Contributions

This dissertation makes the following contributions for two of the most relevant problems related to power management on multicore and manycore systems, specifically, optimizing performance under power/thermal constraints, and minimizing energy under performance constraints. An overview of the interactions and relations of these contributions in a common manycore system are shown in Figure 1.9.

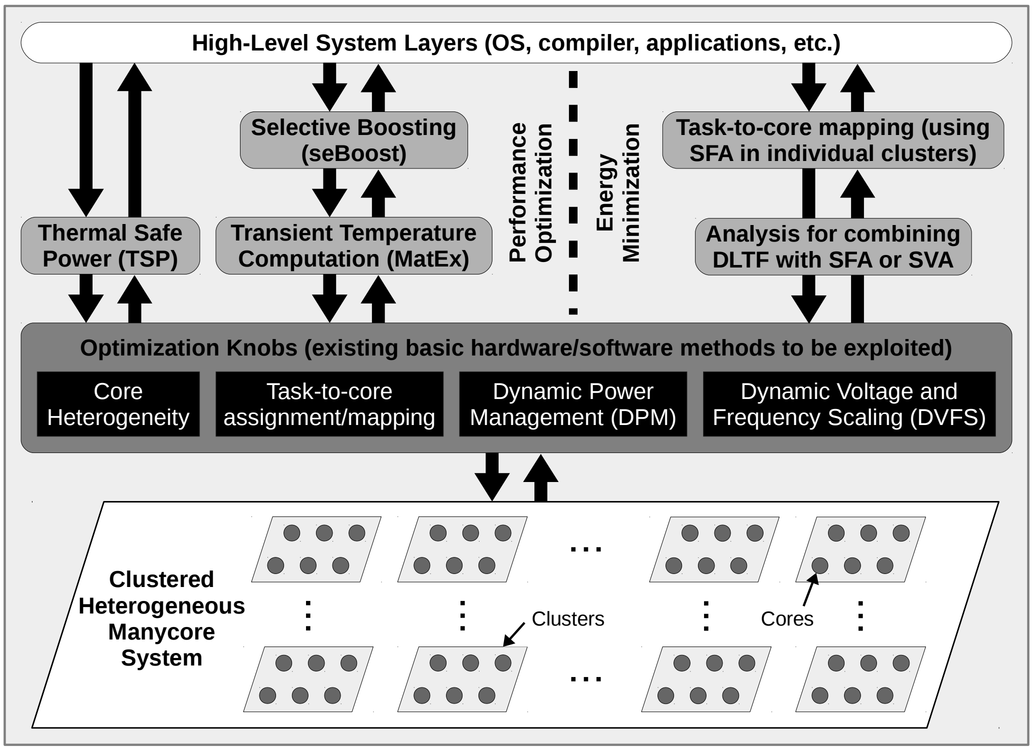

High-Level System Layers (OS, compiler, applications, etc.)
Selective Boosting (seBoost)
Performance Optimization
Energy Minimization
Task-to-core mapping (using SFA in individual clusters)
Thermal Safe Power (TSP)
Transient Temperature Computation (MatEx)
Analysis for combining DLTF with SFA or SVA
Optimization Knobs (existing basic hardware/software methods to be exploited)
Core Heterogeneity
Task-to-core assignment/mapping
Dynamic Power Management (DPM)
Dynamic Voltage and Frequency Scaling (DVFS)
Clustered Heterogeneous Manycore System
Clusters
Cores

Figure 1.9: Interactions and relations of the contributions presented in this dissertation.

1.6.1 Performance Optimization under Power and Thermal Constraints

Efficient Thermal-Aware Power Budgeting

We present a novel thermal-aware power budgeting concept, called Thermal Safe Power (TSP), which is an abstraction that provides safe (but efficient) power and power density constraints as a function of the number of simultaneously active cores. Based on the thermal model of a specific chip and its cooling solution, we derive a polynomial-time method for computing the Thermal Safe Power (TSP) values for the worst-case mappings. That is, we derive an algorithm for computing a per-core power budget for a given number of simultaneously active cores, that will be thermally safe for any possible mapping with that number of active cores, and that will also have little pessimism which could lead to underutilized resources and a large thermal headroom. Furthermore, we derive a second algorithm that is able to compute the TSP values at runtime for a particular mapping of active cores and ambient temperature, further reducing any possible thermal headroom.

TSP conceptually changes the typical design that uses a single and constant value as per-chip or per-core power budget, e.g., the Thermal Design Power (TDP). Moreover, TSP can also serve as a fundamental tool for guiding task partitioning, core mapping, DPM, and DVFS algorithms on their attempt to achieve high predictable performance under thermal constraints.

Analytical Transient and Peak Temperature Computation

We show that runtime decisions typically used to optimize resource usages (e.g., task migration, DPM, DVFS, etc.) can result in transient temperatures much higher than the normally considered steady-state scenarios. In order to be thermally safe, it is important to evaluate the transient peaks before making resource management decisions. To this end, we present a lightweight method for computing these transient peaks, called MatEx. MatEx works with any compact thermal model composed by a system of first-order differential equations, e.g., the thermal model used by HotSpot [33]. In contrast to traditional numerical methods, MatEx is based on matrix exponentials and linear algebra, thus allowing us to derive an analytical expression that can be used for computing any future transient temperatures without requiring incremental computations or incurring in accuracy losses. Moreover, the peaks in the transient temperatures can be then computed simply by analyzing and differentiating such an expression, which is something not possible to do when solving the system of differential equations using traditional numerical methods.

Selective Boosting

We present an efficient and lightweight runtime boosting technique based on transient temperature estimation (specifically, based on MatEx), called seBoost. Traditional boosting techniques select the boosting levels (for boosted cores) and the throttle-down levels (for non-boosted cores) arbitrarily or through step-wise control approaches, and might result in unnecessary performance losses for the non-boosted cores or in failing to satisfy the required runtime performance surges for the boosted cores. Contrarily, seBoost relies on MatEx to select the boosting levels. Therefore, seBoost can guarantee to meet the required runtime performance surges for the boosted cores, while maximizing the boosting time with minimum performance losses for the non-boosted cores.

1.6.2 Energy Minimization under Real-time/Performance Constraints

Energy and Peak Power Efficiency Analysis for Simple Approximation Schemes

Focusing on an individual cluster of a chip, we present the Double Largest Task First (DLTF) scheme (an extension of the LTF scheme) and propose to use it to partition the tasks assigned to a cluster into task sets (assigned to cores). Then, we propose to use the Single Frequency Approximation (SFA) or the Single Voltage Approximation (SVA) scheme for selecting the DVFS levels on the corresponding cluster. Combining DLTF with either SFA or Single Voltage Approximation (SVA) are two simple and practical solutions for energy minimization that incur in little or no DVFS overheads at runtime. In SFA, all the cores in a cluster run at a single voltage and frequency, such that all tasks meet their performance constraints. Furthermore, since all the cores run at a single frequency and no frequency alignment for DVFS between cores is needed, any uni-core

DPM technique for reducing the energy consumption for idling can be easily incorporated individually on each core in the cluster. In SVA, all the cores in the cluster also run at the same single voltage as in SFA; however, the frequency of each core is individually chosen, such that the tasks in each core can meet their performance constraints, but without running at unnecessarily high frequencies. Hence, for the worst-case execution times of tasks, all the cores are executing tasks at all times, and there is no need for any DPM technique to reduce the energy consumption for idling.

Most importantly, we provide thorough theoretical analysis of both solutions, in terms of energy efficiency and peak power reduction, against the optimal task partitioning and optimal DVFS schedule, especially for the state-of-the-art designs that have just a few number of cores per cluster.

Energy-Efficient Task-to-core Assignment for Clustered Manycores

We extend the analysis of SFA for systems with multiple clusters and present two task-to-core mapping solutions when using SFA on individual clusters, specifically, an optimal dynamic programming algorithm for homogeneous manycores, and a lightweight and efficient heuristic for heterogeneous manycores.

1.7 Dissertation Outline

The rest of this dissertation is organized as follows:

- Chapter 2 summarizes the background and related work concerning performance optimization under power/thermal constraints and energy minimization under performance (or hard real-time) constraints.
- Chapter 3 summarizes the system model used throughout the dissertation.
- Chapter 4 summarizes the common simulation framework used for the experimental evaluations conducted throughout the dissertation.
- Chapter 5 presents the details of the thermal-aware TSP power budgeting technique.
- Chapter 6 presents the details of the MatEx transient and peak temperature computation technique.
- Chapter 7 presents the details of the seBoost boosting technique.
- Chapter 8 presents thorough theoretical analysis for combining DLTF with either SFA or SVA, in terms of energy efficiency and peak power reduction, against the optimal task partitioning and optimal DPM/DVFS schedule, especially for the state-of-the-art designs that have just a few number of cores per cluster.
- Chapter 9 extends the analysis of SFA for homogeneous multicore systems clustered in multiple voltage islands, and presents the details of an optimal dynamic programming technique for energy-efficient task-to-core assignment when using SFA on individual clusters.
- Chapter 10 presents the details of a lightweight and efficient heuristic technique for energy-efficient task-to-core assignment for heterogeneous multicore systems clustered in multiple voltage islands.
- Chapter 11 summarizes this dissertation and discusses the future work.

1.8 Orientation within Funding Projects

1.8.1 Invasive Computing

Figure 1.10 illustrates the operational flow of an integrated and coordinated cross-layer dark silicon management method and its interactions with other layers of a heterogeneous computing system, as part of the *Power-Efficient Invasive Loosely-Coupled MPSoCs (B3)* subproject, in the context of the Transregional Collaborative Research Centre *Invasive Computing* [SFB/TR 89] supported by the German Research Foundation (DFG). The chapters focusing on performance optimization under power/thermal constraints (i.e., Chapters 5, 6, and 7), are concrete contributions to the dark silicon management layer of this subproject.

After reading the raw data from the hardware sensors, the **sensing layer** collects and correlates any relevant information from the chip to be used as a basis for the dark silicon management decisions. This relevant data are hardware monitor values (e.g., power consumption of cores and their temperature), as well as probes that capture traces (e.g., thread execution time) or evaluate system-specific information (e.g., thread generation rates). These power and workload statistics are forwarded from the sensing layer to the dark silicon

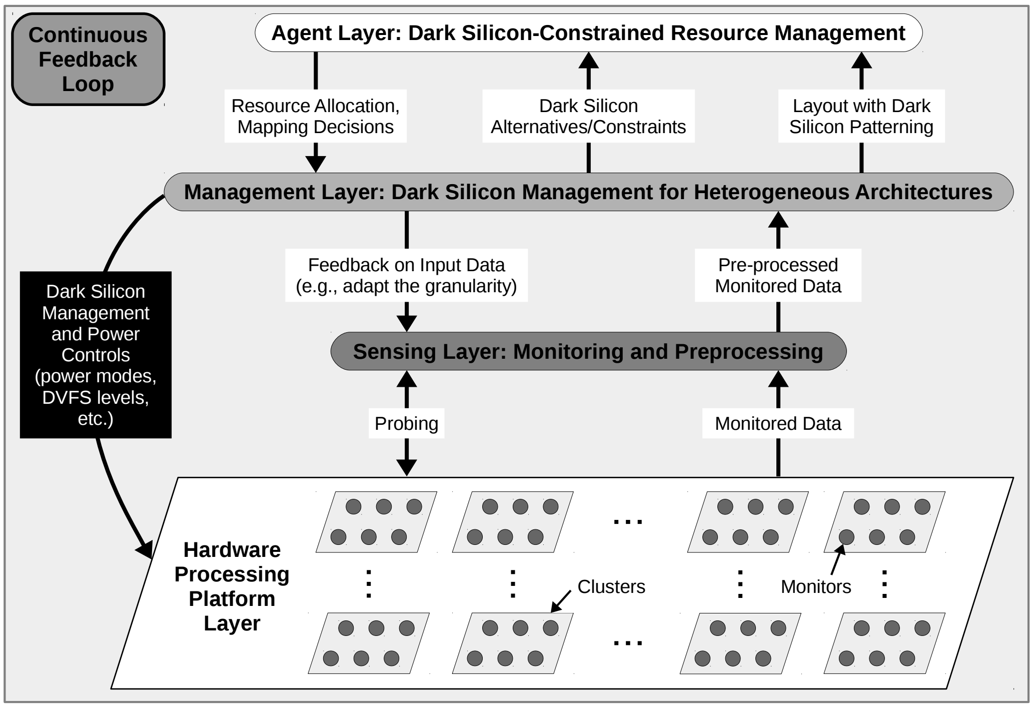

Continuous Feedback Loop
Agent Layer: Dark Silicon-Constrained Resource Management
Resource Allocation, Mapping Decisions
Dark Silicon Alternatives/Constraints
Layout with Dark Silicon Patterning
Management Layer: Dark Silicon Management for Heterogeneous Architectures
Feedback on Input Data (e.g., adapt the granularity)
Pre-processed Monitored Data
Dark Silicon Management and Power Controls (power modes, DVFS levels, etc.)
Sensing Layer: Monitoring and Preprocessing
Probing
Monitored Data
Hardware Processing Platform Layer
Clusters
Monitors

Figure 1.10: Overview of a distributed dark silicon management system illustrating the interaction of different system components across several hardware and software layers.

management layer, for a certain spatial granularity (cores, clusters, or group of clusters) and temporal granularity (time elapsed between two inputs from the sensing layer). The granularity of the statistics is selected based on feedback from the dark silicon management layer.

The **dark silicon management layer** performs various optimizations, particularly, dark silicon patterning (i.e., mapping alternatives for active cores), DPM (i.e., deciding the power state of cores), DVFS, boosting decisions, and coarse-/fine-grained power budgeting. These optimization decisions are based on the pre-processed monitored data received from the sensing layer, accounting for the heterogeneous nature of the different compute fabrics in a clustered architecture. Then, the chosen dark silicon patterning and DVFS levels are forwarded to the agent system as a set of thermally-safe mapping alternatives. Moreover, the dark silicon management layer provides feedback to the sensing layer in order for it to adapt the collection of platform data based on the achieved power and energy efficiency. These adaptations may be changing the spatial and temporal granularity of observation, or selecting/filtering specific data sources.

The **agent layer** then performs resource allocation and dark silicon-aware mapping based on the inputs received from the dark silicon management layer, in order to maximize the performance-per-power efficiency in a distributed manner. The agent layer is both power and temperature agnostic, and thus thermal violations would occur frequency without the existence of the dark silicon management layer. Therefore, power-efficient resource management under dark silicon constraints (i.e., power and temperature constraints) is enabled by the continuous feedback between the dark silicon management layer and the agent layer. The dark silicon management layer has a physical view of the available resources and the system state (e.g., temperature, power consumption, etc.), such that it can define thermally-safe mapping alternatives that are forwarded to the agent layer. Contrarily, the agent layer has an application level view and it attempts to provide a good performance-per-power efficiency for the applications. Namely, the agent layer optimizes performance under the physically defined constraints provided by the dark silicon management layer.

1.8.2 Power Management for Multicore Architecture with Voltage Islands

The chapters focusing on energy minimization under real-time/performance constraints (i.e., Chapters 8, 9, and 10), are concrete contributions to projects *Power Management for Multicore Architecture with Voltage Islands (PM^{4}MAVI)* (part of the Baden Württemberg MWK Juniorprofessoren-Programms) and *Energy Efficient Schedules for Multi-core Architectures with Voltage Islands* (part of DAAD exchange projects).

Chapter 2

Background and Related Work

This chapter summarizes the background and related work concerning performance optimization under power/thermal constraints and energy minimization under performance (or hard real-time) constraints.

2.1 Performance Optimization under Power or Thermal Constraints

2.1.1 Techniques using Per-chip Power Constraints

There are many works in the literature that focus on performance optimization under a per-chip power budget constraint, e.g., [16, 54, 86, 87, 93].

Sartori and Kumar [93] propose three design-time techniques for maximizing the overall performance, specifically, mapping the power management problem to a knapsack problem, mapping the problem to a genetic search problem, and mapping the problem to a simple learning problem with confidence counters. Their work shows that these techniques prevent the total power consumption from exceeding the given per-chip power budget, and they enable the placement of more cores on a die than would be normally allowed by the budget.

Kultursay *et al.* [54] build a 32-core TFET-CMOS heterogeneous multicore chip and present a runtime scheme to optimize the performance of applications executing on such cores, while operating under a given per-chip power budget. The runtime scheme combines thread-to-core mapping on a heterogeneous system, dynamic work partitioning, and dynamic power partitioning.

Ebi *et al.* [16] present a distributed agent-based[1] power management technique aiming at peak temperature reduction for soft real-time tasks on a homogeneous system with per-core DVFS. The thread-to-core mapping is assumed to be given, and the DVFS level selection of independent cores is managed by agents (one agent per core), based on trading power units among adjacent cores, compared to a classical supply and demand model of computational economics. The total amount of power units is fixed at design-time by considering a per-chip power budget.

Raghunathan *et al.* [87] attempt to exploit process variations among cores in order to choose the most suitable cores for each application, for overall performance optimization on homogeneous systems. The results of their experiments suggest that, mostly due to the proportional increment of the process variations on a chip, the overall performance can be potentially increased along with the increment of the dark silicon area.

2.1.2 Techniques using Thermal Constraints

Khdr *et al.* [48] propose a design-time resource management technique based on dynamic programming, denoted as *DsRem*, which aims at maximizing the overall system performance by distributing the available cores among different applications, such that the maximum steady-state temperature among all cores remains below the critical temperature. By relying on extensive design-time application profiling, *DsRem* determines

[1] An agent is an autonomous computational entity (its behavior at least partially depends on its own experience) that perceives and acts upon its environment [105].

the number of active cores and their DVFS levels considering the TLP and ILP characteristics of every application. Particularly, for applications that have a high TLP, *DsRem* will try to execute them by using a large number of cores running at low DVFS levels (thus exploiting the parallelism of such applications). Contrarily, applications with high ILP will try to be executed by using a small number of cores running at high DVFS levels. If there exist applications that exhibit both high TLP and high ILP, *DsRem* will attempt to execute them by using a large number of cores running at high DVFS levels. In more detail, *DsRem* consist of three phases: (1) it first distributes the cores among applications (considering their TLP and ILP characteristics) and chooses the DVFS levels to maximize the overall performance under a TDP constraint, (2) it then greedily maps applications with a high average power consumption to cores with a low steady-state temperature (estimating the new temperature after mapping every task) attempting to reduce the peak steady-state temperature, and (3) it heuristically adapts the previous decisions in an attempt to reduce the peak steady-state temperature or exploit available thermal headroom.

The work in [96] proposes a variability-aware dark silicon manager, called *DaSiM*. The idea behind *DaSiM* is to exploit the heat transfer among cores by greedily deriving a thread-to-core mapping that minimizes the peak temperature on the chip, assuming a model for core-to-core leakage power variations. An efficient thread-to-core mapping will directly influence the temperature distribution on the chip due to improved heat dissipation, potentially enabling the activation of more cores (beneficial for high TLP applications), and/or boosting certain cores (beneficial for high ILP applications). To enable runtime optimizations, *DaSiM* also presents a lightweight steady-state temperature prediction mechanism that can estimate the resulting temperature distribution for a candidate solution.

Hanumaiah *et al.* [25] propose a thermal management technique based on RC thermal networks which attempts to minimize the latest completion time among the applications for homogeneous multicores with per-core DVFS. The technique is separated in two parts: first, the task-to-core allocation is determined at the beginning of a migration interval (typically every 50 ms to 100 ms), and secondly, the DVFS levels of the cores are adjusted independently (typically every 5 ms to 10 ms). The proposed technique uses convex optimization to derive an optimal solution to the stated problem, but with very high time complexity which is only suited for design-time decisions. Nevertheless, authors then exploit the structure of certain matrices in the RC thermal network in order to have an approximate solution which is reportedly 20 times faster than the convex optimization approach. Authors also consider the dependency between leakage power consumption and temperature in their formulation through a piece-wise linear approximation. The implementation of such a technique for runtime adaptations is however debatable, as the experiments in [25] show that it may require more than 15 ms to compute the DVFS levels of *each* core for a given mapping, and more than 120 ms to decide the corresponding mapping solution, which is generally not fast enough for runtime usage in manycore systems.

Intel's Turbo Boost [9, 10, 34, 35, 91] enables a group of cores to execute at high DVFS levels when there exists headroom within the power, current, and temperature constraints. More specifically, whenever the temperature, power, and current are below their predefined constraints, the cores increase (i.e., *boost*) their DVFS levels one step at a time (within a control period, e.g., 1 ms) until either the temperature, power, or current reaches its predefined upper limit (which could also depend on the number of active cores). Similarly, whenever the temperature, power, or current exceed their associated constraints, the cores reduce their DVFS levels one step at a time (also within a control period, e.g., 1 ms) until the corresponding constraints are satisfied, or until the nominal DVFS levels are reached. Naturally, as already seen in Figure 1.3, boosting to high DVFS levels will result in high power consumptions. Nevertheless, although the temperature will raise due to the increments in power consumption, Turbo Boost exploits the thermal capacitances inside the chip by knowing that this temperature raise will not occur immediately after a change in power, but it will rather require some time to reach or exceed the critical temperature. Figure 2.1 shows an example of the behavior of Intel's Turbo Boost for a critical temperature of 80°C, based on simulations conducted with gem5 [5], McPAT [57], and HotSpot [33], when executing two applications from the PARSEC benchmark suite [4] (each application executing 8 parallel dependent threads, one thread per core) on a system with 16 OOO Alpha 21264 cores.

Computational sprinting [85] is a different type of boosting technique that proposes to optimize performance at runtime by exploiting parallelism. Particularly, sprinting is based on activating cores that are normally inactive (e.g., idle or in a low-power mode) during short bursts of time (typically shorter than 1 s). Motivated by the almost cubic relationship between the DVFS levels of execution and the power consumption

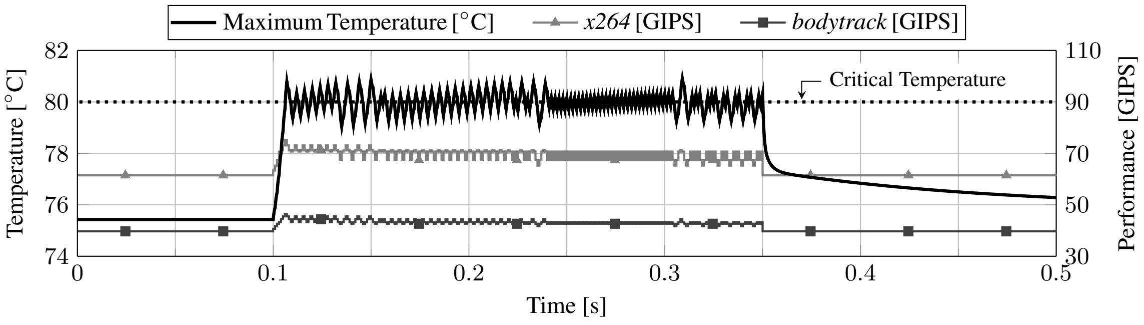


Figure 2.1: Intel's Turbo Boost [9] example. The bold line represents the maximum temperature among all elements in the chip at a given time point (left axis). The performance of the applications is measured in Giga Instructions per Second (GIPS) (right axis).

on a core (as seen in Figure 1.3), computational sprinting intentionally discourages boosting through DVFS. On the contrary, sprinting is motivated by the ideally linear relationship between computational performance and power consumption which is expected when activating several cores running at the same DVFS levels. However, although this is an interesting approach, only applications with very high ILP will scale their performance linearly when activating more cores, while the performance increments of normal applications can become quickly saturated due to what is known as the parallelism wall, as seen in the example in Figure 2.2. Furthermore, the latencies associated to waking up cores from low-power modes, as well as the correspondent thread migrations, can potentially result in large overheads, especially when considering the short duration of the sprinting periods. Because of these reason, Intel's Turbo Boost will generally achieve a higher overall performance than computational sprinting.

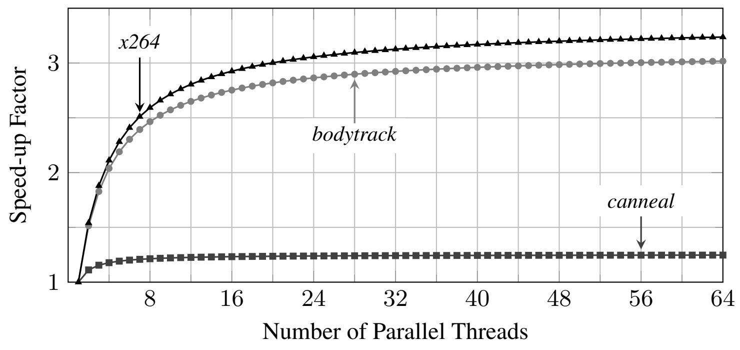


Figure 2.2: Speed-up factors (with respect to the number of parallel threads, normalized to the execution time of running a single thread) of three applications from the PARSEC benchmark suite [4] running at 2 GHz on an OOO Alpha 21264 core, based on simulations conducted with gem5 [5] (up to 8 threads) and Amdahl's law (for more than 8 threads).

2.1.3 Temperature Estimation

There are significant works in the literature for temperature estimation/computation in integrated circuits. Most techniques limit themselves to steady-state computations, e.g., [56, 84, 112]. However, there are also several techniques capable of dealing with transient temperature estimations, e.g., [2, 32, 33, 51, 89, 103, 104, 114].

Wang and Chen [104] present a 3-D transient temperature simulator which is based on the Alternating Direction Implicit (ADI) method, having linear time complexity and a linear memory requirement. The problem with this method is that it cannot model different packaging components with detailed temperature distribution information. Klab *et al.* [51] propose to couple a gate-level logic simulator with an analytical solution. However, their model is only able to consider the dynamic power consumption of cores, assuming a negligible leakage power, which is not a realistic assumption. Moreover, the analytical solution limits the approach only to dice geometries, since it is computationally exhausting for complex geometries with composite materials. Ardestani and Renau [2] propose a time-based sampling simulator, called *ESESC*, that enables integrated power and temperature evaluations in statistically sampled simulations for multicore systems. The problem with *ESESC* is that the temperature simulations cannot be easily separated from the complete system simulations, which denies the ability to have fast temperature estimations based on power consumption values. Rai *et al.* [89] present a technique for temperature estimation based on application-specific calibrations using the available built-in sensors, such that it does not need information about the chip's floorplan or power consumption of the cores. Their work shows that every application has a unique temperature profile, thus allowing to derive a lightweight method for temperature estimation that combines mapping and scheduling information. The limitation of [89] is that it assumes that an application consumes constant power during its entire execution, which is not necessarily a practical assumption, as already seen in Figure 1.2.

The most widely used temperature simulator is HotSpot [33]. Based on the popular stacked-layer packaging scheme of modern VLSI systems (as seen in the example in Figure 2.3), HotSpot constructs a compact thermal model that results in an RC thermal network (further details in Chapter 3.5). Aside from modeling silicon and several packaging layers made of different materials, HotSpot also includes a high-level on-chip interconnect self-heating power and thermal model that allows it to take in consideration the thermal impacts of the interconnects. For transient temperature estimations, HotSpot solves the system of first-order differential equations associated to the RC thermal network of a specific chip (more details in Chapter 3.5) by using a fourth-order *Runge-Kutta* numerical method with an adaptive number of iterations.

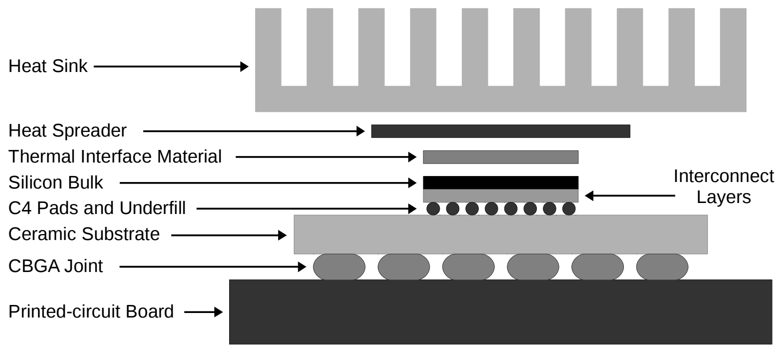


Figure 2.3: Stacked layers in a typical Ceramic Ball Grid Array (CBGA) package (adapted from a figure in [33]).

There exist a few techniques that currently compete with HotSpot, e.g., [32, 103, 114]. *Power Blurring* [114] is a matrix convolution method, where the distribution of temperature throughout the chip is represented as a blurred image of the power consumption map when it is blurred with a filter mask for the impulse response. The temperature profile for a given power consumption map is therefore computed by convolving the power map with the filter mask. The filter mask for the impulse response is obtained using a Finite Element Analysis (FEA) tool such as *ANSYS* [1]. Wang *et al.* [103] present a compact thermal modeling technique that derives a composable thermal model. The composable model is derived from detailed structures for each basic block on the chip by using the finite difference method. The complexity of the model is then reduced, and the technique attempts to merge the boundary nodes of blocks to improve the reduction efficiency, thus leading to different space discretizations for the whole thermal system. By using Generalized Integral Transforms (GIT), Huang and Lee [32] present an analytical temperature simulator that estimates the temperature distribution on a chip with a truncated set of spatial bases that only needs a small number of truncation points.

Coskun *et al.* [13] present a thermal management approach that is based on temperature estimation using Autoregressive Moving Average (ARMA) modeling, which is able to estimate future temperatures based on past temperature measurements. The ARMA model is updated at runtime whenever a workload change is detected. However, this technique does not directly model the relationship between power consumption, DVFS levels, and the workload, and more importantly, it does not consider a thermal model that accounts for the transfer of heat among cores. Therefore, the work from [13] is not well suited to estimate future temperatures *before* a change in workload occurs, and thus it is not able to evaluate the potential benefits of a candidate task migration or change of the DVFS levels.

2.2 Energy Minimization under Real-time/Performance Constraints

2.2.1 Per-core DVFS Techniques

For per-core DVFS, power-aware and energy-efficient scheduling for homogeneous multicore and manycore systems has been widely explored, especially for real-time systems with hard deadlines, e.g., [3, 11, 12, 14, 60, 108]. Chen and Thiele [12] have shown that using the LTF strategy for energy-efficient task partitioning results in solutions with upper-bounded approximation factors, in which the absolute values of the approximation factors depend on the specific hardware platforms. Particularly, by power gating cores to reduce the energy consumption, Xu *et al.* [108] and Chen *et al.* [12] present polynomial-time algorithms to derive task partitions that attempt to execute at the most energy-efficient DVFS levels. Moreover, de Langen and Juurlink [14] provide some heuristics for energy-aware scheduling, while Moreno and de Niz [60] propose an algorithm that, with polynomial-time complexity, computes the optimal DVFS setting for systems with uniform DVFS steps and negligible leakage power consumption.

There are also several works in the literature focusing on heterogeneous multicore systems with per-core DVFS, e.g., [110]. The work in [110] presents a dynamic programming algorithm that uses trimming by rounding, in which the rounding factor trades the quality of the derived solution (in terms of energy consumption) with the total execution time of the algorithm.

2.2.2 Global DVFS Techniques

For homogeneous systems with global DVFS, negligible leakage power consumption, and negligible overheads for entering/leaving low-power modes, Yang *et al.* [109] present an optimal DVFS schedule for energy minimization when executing periodic *frame-based real-time tasks* (i.e., all the tasks have the same arrival time, deadline, and period). Their solution is based on an accelerating DVFS schedule and the *deep sleeping property* (which states that every core in the system is put to sleep after executing all the workload in its ready queue), as shown in the example in Figure 2.4. For a system with M cores, after the task partitioning is finished, the cores are ordered increasingly according to their assigned workload, and every periodic frame is then divided into M fragments. In the i-th fragment, all active cores are executing at frequency f_i during time t_i. After time t_i elapses, the i-th core finishes all its workload for the current frame and it goes to sleep, such that there are $M - i + 1$ active cores in the i-th fragment. The problem with the approach in [109] is that it is highly restrictive and it cannot be easily extended for systems with non-negligible leakage power consumption, or to handle periodic real-time tasks (i.e., tasks have different arrival times and periodicity).

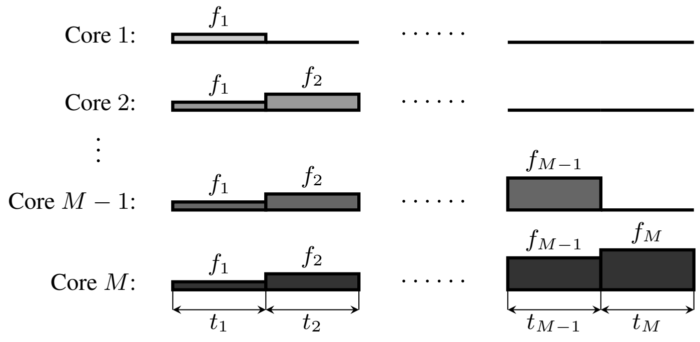


Figure 2.4: An accelerating schedule for frame-based real-time tasks satisfying the *deep sleeping property*.

The studies in [15, 94] relax some of the assumptions made in [109] by considering periodic real-time tasks with non-negligible leakage power consumption, as well as non-negligible overhead for entering/leaving low-power modes. The technique in [15] first partitions tasks to choose the number of active cores, and it then decides the DVFS levels of the active cores for execution, using SFA when tasks do not complete earlier than the estimated worst-case execution times. Nevertheless, [15] lacks the theoretical analysis that supports the effectiveness of their approach for energy minimization in the worst cases. Seo *et al.* [94] propose to dynamically balance the task loads of multiple cores in order to optimize the power consumption during execution, and to repartition tasks to adjust the number of active cores in order to reduce the leakage power consumption under low load conditions.

2.2.3 Clustered Manycores / Multiple Voltage Islands

In the literature, there exist some heuristic solutions for energy minimization on clustered homogeneous manycores with multiple voltage islands, e.g., [24, 53, 65, 107]. Furthermore, among these examples, [65] and [107] use the SFA scheme to choose the DVFS levels on individual clusters.

Particularly, Kong *et al.* [53] propose a heuristic that selects the number of active clusters and then partitions the tasks according to the LTF scheme. The task model from [53] is then extended by Han *et al.* [24] in order to consider shared resources. Moreover, the work in [24] also presents a modified (synchronized) version of LTF, that is used to partition the tasks by assuming that only one task can access a particular resource at any given time. Nikitin and Cortadella [65] propose an Extremal Optimization Heuristic (EOH) assuming that tasks are modeled as task graphs, considering the corresponding communication costs between tasks. However, it considers that only one task can be assigned to each core, and it fails to provide theoretical analysis for the energy efficiency of the their solution. Wu *et al.* [107] propose a heuristic based on genetic algorithms that assigns real-time tasks to clusters, in which the selection, crossover, and mutation operators gradually optimize the energy consumption through an iterative process.

To the best of our knowledge, there exist only a handful of works in the literature that target either average power consumption minimization or energy minimization on clustered heterogeneous manycores with multiple voltage islands, e.g., [18, 63, 64]. Muthukaruppan *et al.* [64] present a hierarchical power management framework based on control theory, which aims at minimizing the overall average power consumption while satisfying the required Quality of Service (QoS) of the applications (i.e., their performance constraints), while using TDP as a per-chip power constraint. Task migrations and DVFS are used at different levels, particularly, at a task level, at a cluster level, and on the chip controllers. The controllers are coordinated in such a way that they can throttle down the power consumption in case TDP is exceeded, and to do task-to-core mapping in order to optimize the overall performance. The work in [63] presents a hierarchical power management technique based on price theory (following supply and demand based market mechanisms) for clustered heterogeneous multicores with multiple voltage islands, which aims at minimizing the average power consumption (not energy) while satisfying application performance goals under a per-chip power constraint, specifically, TDP. As this approach is not targeted for real-time systems, it does not guarantee a feasible schedule for real-time tasks with hard deadlines. The technique from [63] is implemented as a collection of autonomous agents (specifically, task agents, core agents, cluster agents, and a chip agent), where each agent represents a transactional body in the market (earning, bidding, purchasing, and distributing computational resources). However, since minimizing the power consumption does not always translate into minimizing energy, the techniques from [64] and [63] could potentially result in high energy consumption values when executing at low frequencies even if the performance constraints are satisfied. Contrarily, Elewi *et al.* [18] do target energy minimization and propose a simple greedy task partitioning scheme called Equally-Worst-Fit-Decreasing (EWFD), which focuses on balancing the total workload assigned to each cluster. After the task partitioning is completed, EWFD uses SFA to select the DVFS levels on individual clusters.

Chapter 3

System Model

This chapter summarizes the system models used throughout the dissertation.

3.1 Application Model

For the chapters targeting performance optimization (i.e., Chapters 5, 6, and 7), we do not consider any specific application model for our contributions. Furthermore, with the exception of just a few cases, all the experiments/examples conducted/presented throughout the dissertation consider multi-threaded applications from the PARSEC benchmark suite [4], where every executed application runs several parallel dependent threads. Nevertheless, the chapters targeting energy minimization (i.e., Chapters 8, 9, and 10) focus on real-time embedded systems, for which some application model is required.

Particularly, for Chapters 8, 9, and 10, we consider that every thread of an application (for the case of multi-threaded applications) can be modeled as a periodic real-time task with an implicit hard deadline, such that there are in total R periodic tasks, denoted as $\{\tau_1, \tau_2, \ldots, \tau_R\}$. We assume that there is no data dependency among tasks, i.e., we consider independent tasks. *Note that although this seems like a limiting assumption, it is in fact a practical assumption when considering closed intellectual property multi-threaded applications, for which is not generally possible to accurately model data dependency among threads. Furthermore, there exist methods to model a set of dependent tasks as a set of independent ones [52].* Every task τ_n releases an infinite number of task instances with period (and relative deadline) d_n and every instance has worst-case execution cycles $e_{q,n}$ when being executed on a core of type q. For example, Figure 1.1 already presented experimental results showing the total execution cycles of several applications from the PARSEC benchmark suite [4], where it can be observed that, for those applications, the execution cycles of a task remain constant (for practical purposes) in regards to the DVFS levels for execution, but a certain task needs a different amount of cycles when running on different types of cores. Moreover, note that for memory bound applications, the worst-case execution cycles of a task might grow with the execution frequency, as memory operations do not scale with the core's frequency. Thus, for memory bound applications, the maximum worst-case execution cycles (normally occurring at the highest DVFS levels) should be considered.

In regards to scheduling, we consider partitioned scheduling, where every task is assigned to one core. When a new task instance (also denoted as *job*) arrives to the system and is ready for execution, the task is executed on the assigned core. Moreover, in each independent core, we use Earliest Deadline First (EDF) [59] to schedule the tasks, such that the task instance (job) with the earliest absolute deadline on each core has the highest priority and preempts other tasks, as seen in the example in Figure 3.1. The least common multiple among all periods of all tasks is called the *hyper-period*, denoted as D. The schedule decided by the selected scheduling policy (e.g., EDF), is repeated every hyper-period. A special (more restricted) case of this task model are periodic framed-base tasks, where all R tasks arrive at the same time and have the same period and relative deadline, e.g., as is the case for the work in [109] shown in the example in Figure 2.4.

After the task partitioning is completed by considering M cores, all R tasks are grouped into M disjoint task sets, denoted as $\{\mathbf{S}_1, \mathbf{S}_2, \ldots, \mathbf{S}_M\}$, such that there are no more task sets to map than available cores in the chip. We assume that if the task partitioning strategy groups the tasks into less than M task sets, dummy empty task sets (i.e., task sets without assigned tasks that do not need to execute any workload) are included in

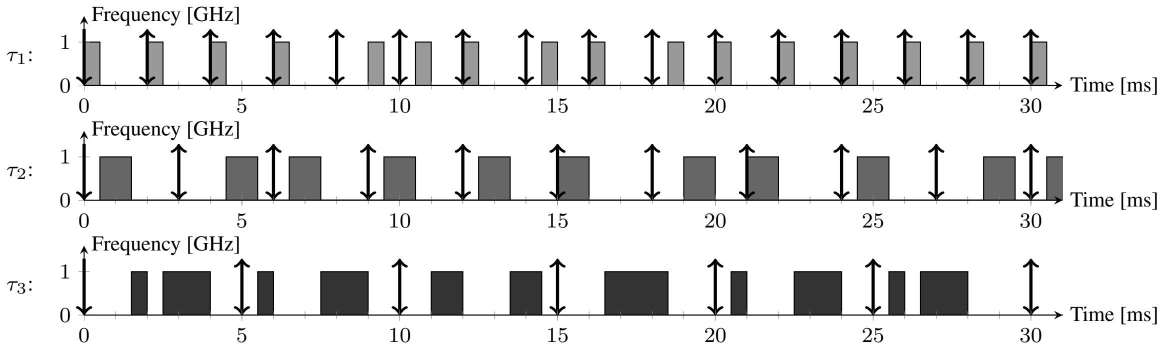


Figure 3.1: Example of an EDF schedule for three periodic tasks, i.e., τ_1, τ_2, and τ_3, with periods (and deadlines) $d_1 = 2\,\text{ms}$, $d_2 = 3\,\text{ms}$, and $d_3 = 5\,\text{ms}$, hyper-period $D = 30\,\text{ms}$, and worst-case execution cycles $e_{q,1} = 0.5 \cdot 10^6$ cycles, $e_{q,2} = 1 \cdot 10^6$ cycles, and $e_{q,3} = 2 \cdot 10^6$ cycles, assigned to one core of type q running at 1 GHz. Symbol $\downarrow$ represents the arrival time of a new task instance (job) ready for execution. Symbol $\uparrow$ represents the period and deadline of a task instance (job). For periodic tasks with implicit deadlines, $\downarrow$ and $\uparrow$ overlap, i.e., $\updownarrow$. In this example, in case two or more tasks have the same absolute deadline and one of these tasks was already being executed, then we continue to run it in order to reduce the number of preemptions. Otherwise, we run the task with lowest index.

order to have a total of M task sets. We define the *cycle utilization* of task set $\mathbf{S}_i$ running on a core of type q as $w_{q,i} = \sum_{\tau_n \in \mathbf{S}_i} \frac{e_{q,n}}{d_n}$, with unit $\frac{\text{cycles}}{\text{second}}$. Empty task sets without assigned tasks have a resulting cycle utilization of 0 for any type of core. The *total cycle utilization* among all tasks for a given core of type q, can be computed as $\sum_{n=1}^{R} \frac{e_{q,n}}{d_n} = \sum_{i=1}^{M} w_{q,i}$. By defining $w_{q,0} = 0$ for all cores of type q, for simplicity of presentation and without loss of generality, we order the task sets such that $0 = w_{j,0} \leq w_{j,1} \leq w_{j,2} \leq \cdots \leq w_{j,M}$, where j is an arbitrary type of core used for reference when ordering the task sets, e.g., the lowest-power core. *It has been well studied, e.g., in [59], that executing task set $\mathbf{S}_i$ on a core of type q at frequencies higher than or equal to $w_{q,i}$ with EDF guarantees that all the tasks inside task set $\mathbf{S}_i$ will meet their timing constraints.*

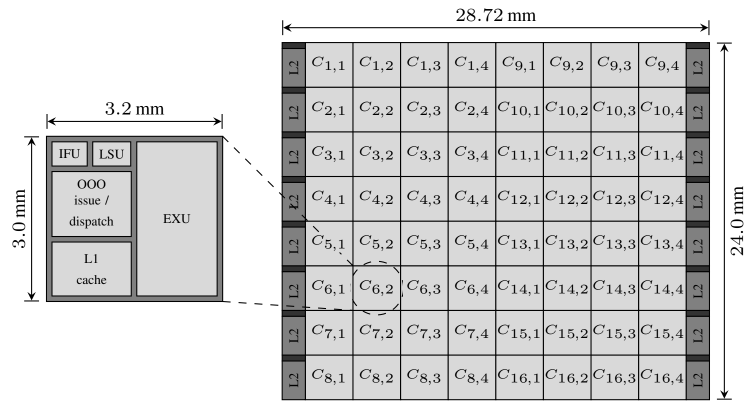
In this way, when using EDF to schedule tasks on individual cores (as is the case in this dissertation), satisfying the hard real-time constraints of all tasks translates to satisfying the cycle utilizations (i.e., performance constraints) of all task sets. This is the reason why throughout the dissertation we always refer to *real-time or performance* constraints, rather than one or the other. Furthermore, the application model described in this section is not restricted to the real-time domain, and any general performance-constrained applications that requires to recurrently execute a certain amount of computation in a specific time interval can be modeled as described above.

3.2 Hardware Model

Throughout this dissertation we focus on homogeneous and heterogeneous multicore/manycore systems clustered in multiple voltage islands, where there are in total M cores in the chip (among all types of cores). We assume that the cores inside a cluster/island are all of the same type, but different islands can have different types and/or number of cores. Particularly, we assume that the system has Q types of cores and a total of V clusters/islands, such that there is at least one cluster/island for each core type, but there can be several clusters/islands of the same core type, i.e., $Q \leq V$. Every cluster k is composed of M_k^{cluster} cores, such that the total number of cores in the system is $M = \sum_{k=1}^{V} M_k^{\text{cluster}}$. Moreover, there are in total M_q^{type} cores of type q in the chip, such that $M = \sum_{q=1}^{Q} M_q^{\text{type}}$. Depending on convenience, we use two different notations to identify the cores. For the general case in which the cluster to which a core belongs to is not relevant, cores are simply indexes as $C_1, C_2, \ldots, C_M$. When the cluster is relevant, the cores in cluster k are denoted as $C_{k,1}, C_{k,2}, \ldots, C_{k,M_k^{\text{cluster}}}$. We identify the type of core of each cluster through indexes $Q_1^{\square}, Q_2^{\square}, \ldots, Q_V^{\square}$, such that if cluster k is composed of cores of type q then we have that $Q_k^{\square} = q$.

With respect to DVFS, we assume that every core of type q has $\hat{F}_q^{\text{type}} + 1$ available frequencies, denoted as $\left\{F_{q,0}^{\text{type}}, F_{q,1}^{\text{type}}, F_{q,2}^{\text{type}}, \ldots, F_{q,\hat{F}_q^{\text{type}}}^{\text{type}}\right\}$, such that $F_{q,\hat{F}_q^{\text{type}}}^{\text{type}}$ is the maximum frequency for a core of type q, while an inactive core (i.e., idle or in a low-power mode) is said to be set at frequency $F_{q,0}^{\text{type}}$. A similar notation can be defined for individual cores, such that core i has $\hat{F}_i^{\text{core}} + 1$ available frequencies, denoted as $\left\{F_{i,0}^{\text{core}}, F_{i,1}^{\text{core}}, F_{i,2}^{\text{core}}, \ldots, F_{i,\hat{F}_i^{\text{core}}}^{\text{core}}\right\}$, where $F_{i,\hat{F}_i^{\text{core}}}^{\text{core}}$ is the maximum frequency for core i, while when inactive (i.e., idle or in a low-power mode) it is said to be set at frequency $F_{i,0}^{\text{core}}$. In regards to the granularity of DVFS, we assume that all the cores in a cluster share a common voltage and frequency at any given time point (i.e., we consider voltage and frequency clusters). Particularly, in order to be power/energy efficient, for running the cores in a cluster at a desired frequency, the voltage of the cluster is set to the minimum value that supports stable execution (as discussed in Chapter 1.2.4 and later elaborated in Section 3.4). For example, the Exynos 5 Octa (5422) processor based on ARM's big.LITTLE architecture [21] (shown in Figure 1.6) may be described by such a model.

The geometry of a particular architecture is described by its *floorplan*, which specifies the areas and spatial location of the functional blocks composing a chip. The granularity of the blocks that compose the floorplan of a chip can be freely chosen, and may vary for different objectives. For example, consider the homogeneous 64-core system shown in Figure 3.2, based on simulations conducted using gem5 [5] and McPAT [57], for OOO Alpha 21264 cores in $22\,\text{nm}$ technology. According to the simulations, every core has an area of $9.6\,\text{mm}^2$ and is composed by several units: an Instruction Fetch Unit (IFU), an Execution Unit (EXU), a Load and Store Unit (LSU), an out-of-order (OOO) issue/dispatch, and a private L1 cache. Moreover, every cluster is composed of 4 cores, a shared 2 MB L2 cache, and a memory controller. The combined area of the shared L2 cache and memory controller is $4.7\,\text{mm}^2$, which is comparable (in magnitude) to the area of a core. Thus, for deriving a thermal model based on the floorplan of such a chip (later discussed in Section 3.5), a practical approach would be to consider one block for every core (not one block for every internal unit inside each core, especially considering that the power consumptions of the internal units of the cores are tightly related with their DVFS levels), while having independent blocks for the L2 cache and the memory controllers.



28.72 mm
24.0 mm
3.2 mm
3.0 mm
IFU
LSU
OOO issue / dispatch
EXU
L1 cache
L2
$c_{1,1}$ $c_{1,2}$ $c_{1,3}$ $c_{1,4}$ $c_{9,1}$ $c_{9,2}$ $c_{9,3}$ $c_{9,4}$
$c_{2,1}$ $c_{2,2}$ $c_{2,3}$ $c_{2,4}$ $c_{10,1}$ $c_{10,2}$ $c_{10,3}$ $c_{10,4}$
$c_{3,1}$ $c_{3,2}$ $c_{3,3}$ $c_{3,4}$ $c_{11,1}$ $c_{11,2}$ $c_{11,3}$ $c_{11,4}$
$c_{4,1}$ $c_{4,2}$ $c_{4,3}$ $c_{4,4}$ $c_{12,1}$ $c_{12,2}$ $c_{12,3}$ $c_{12,4}$
$c_{5,1}$ $c_{5,2}$ $c_{5,3}$ $c_{5,4}$ $c_{13,1}$ $c_{13,2}$ $c_{13,3}$ $c_{13,4}$
$c_{6,1}$ $c_{6,2}$ $c_{6,3}$ $c_{6,4}$ $c_{14,1}$ $c_{14,2}$ $c_{14,3}$ $c_{14,4}$
$c_{7,1}$ $c_{7,2}$ $c_{7,3}$ $c_{7,4}$ $c_{15,1}$ $c_{15,2}$ $c_{15,3}$ $c_{15,4}$
$c_{8,1}$ $c_{8,2}$ $c_{8,3}$ $c_{8,4}$ $c_{16,1}$ $c_{16,2}$ $c_{16,3}$ $c_{16,4}$

Figure 3.2: Floorplan of a homogeneous 64-core system (16 quad-core clusters) based on simulations in gem5 [5] and McPAT [57], where every core is composed by several units: an IFU, an EXU, an LSU, an OOO issue/dispatch, and a private L1 cache.

For simplicity of presentation, throughout the dissertation we assume that there are a total of Z blocks in the floorplan, such that M is the total number of cores in the chip and $Z - M \geq 0$ is the number of blocks corresponding to other types of components (e.g., L2 caches and memory controllers). For a given type of core, the area of all cores of type q is denoted as $\text{area}_q^{\text{type}}$. For a specific core, the area of core m (among all M in the chip) is defined as $\text{area}_m^{\text{core}}$. Finally, for a given floorplan, when simultaneously activating m cores, there are $\binom{M}{m}$ combinations of *which* specific cores in the floorplan to activate. Throughout the dissertation, we refer to a specific decision of *which* cores to activate as *core mapping* or *mapping of cores*. We define column vector $\mathbf{X} = [x_i]_{M\times 1}$ for a particular mapping of *active* cores (among all cores, ignoring the types of the cores). Vector $\mathbf{X}$ is a binary vector: $x_i = 1$ means that core i corresponds to an *active* core, while $x_i = 0$ means that core i corresponds to an *inactive* core. For considering core heterogeneity, we also define column vector $\mathbf{X}^q = [x_i^q]_{M_q^{\text{type}}\times 1}$ for all types of cores $q = 1, 2, \ldots Q$, which represents a particular mapping of *active* cores of type q. Vector $\mathbf{X}^q$ is a binary vector: $x_i^q = 1$ means that core i corresponds to an *active* core of type q, while $x_i^q = 0$ means that core i corresponds to an *inactive* core of type q.

3.3 Power Model

The algorithms presented in Chapters 5, 6, 7, 9, and 10, do not require to rely on any specific power model. Specifically, in these chapters we simply consider that the power consumed on a core of type q while executing task τ_n at frequency index j (such that $0 \leq j \leq \hat{F}_q^{\text{type}}$), is denoted as $P_q^{\tau_n}\left(F_{q,j}^{\text{type}}\right)$. For notational brevity, upon convenience, the value of $P_q^{\tau_n}\left(F_{q,j}^{\text{type}}\right)$ might represent the average power consumption throughout the entire execution of the task (or for a given execution phase of the task), or the peak power consumption throughout the entire execution of the task (or for a given execution phase of the task), or the instantaneous power consumption at a certain time instant. Furthermore, for a specific core, when core m (among all M in the chip) is inactive (i.e., idle or in a low-power mode, not in execution mode), we assume that it consumes $P_{\text{inact}m}^{\text{core}}$. Similarly, for a given type of core, the inactive power of all cores of type q is denoted as $P_{\text{inact}q}^{\text{type}}$. For the special case of homogeneous systems, all the cores in the system consume the same power when they are inactive, which we define as $P_{\text{inact}}^{\text{core}}$ for simplicity in the notation. Moreover, there is also a maximum chip power constraint, denoted as $P_{\max}$, and a maximum chip current constraint, denoted as $I_{\max}$. Both $P_{\max}$ and $I_{\max}$ are electrical constraints that cannot be exceeded (e.g., from the capacity of the power supply or the wire thickness), and not abstractions like is the case for power budgets (as discussed in Chapter 1.3).

Nevertheless, a more specific power model is required for our theoretical analysis in Chapter 8. Generally, as detailed in [27], the power consumption of a generic CMOS core at a given time point can be modeled as shown in Equation (3.1),

$$P_{\text{core}}\left(V_{\text{dd}}, f, T, t\right) = u_{\tau_n}(t) \cdot C_{\text{eff}} \cdot {V_{\text{dd}}}^2 \cdot f + V_{\text{dd}} \cdot I_{\text{leak}}\left(V_{\text{dd}}, T\right) + P_{\text{ind}} \tag{3.1}$$

where $u_{\tau_n}(t)$ represents the instantaneous activity factor of the core for task τ_n at time t (thus considering the task dependent part of the power consumption), C_{eff} represents the effective switching capacitance of the core, V_{dd} represents the supply voltage, f represents the execution frequency of the core, I_{leak} represents the leakage current (which depends on the supply voltage and the temperature of the core T, such that high temperatures cause high leakage currents), and P_{ind} represents the independent power consumption attributed to maintaining the core in execution mode (i.e., the voltage and frequency independent part of the power consumption). In Equation (3.1), $u_{\tau_n}(t) \cdot C_{\text{eff}} \cdot {V_{\text{dd}}}^2 \cdot f$ represents the dynamic power consumption on the core (mainly generated by switching activities), while $V_{\text{dd}} \cdot I_{\text{leak}}\left(V_{\text{dd}}, T\right)$ represents the leakage power consumption (mainly generated by leakage currents). Thus, running cores at low voltages and frequencies reduces the power consumption.

Furthermore, Equation (3.1) can be further approximated with three considerations. First, given that in order to be power/energy efficient, the voltage of a cluster is set to the minimum value that supports stable execution for the highest execution frequency among all cores in the cluster (as discussed in Chapter 1.2.4 and Section 3.2), according to Equation (1.1), if $V_{\text{dd}} \gg V_{\text{th}}$, we can have a linear approximation relating the supply voltage and the highest execution frequency among all cores in the cluster, defined as f_{cluster}, i.e., it holds that $V_{\text{dd}} \propto f_{\text{cluster}}$. Secondly, we can consider *safe margins* for I_{leak} with regards to the temperature by always modeling the value of I_{leak} for a temperature on the core that reaches the critical temperature, such that I_{leak} only depends on the supply voltage. Thirdly, given that the analysis in Chapter 8 targets energy, we can limit the focus of our power model to average power consumption (rather than instantaneous and time dependent power consumption), and therefore consider the average activity factor on the core instead of the instantaneous activity factor. Moreover, in order to allow for theoretical analysis and given the similarities of most of the average power consumptions values observed in Figure 1.1, we assume that all tasks have similar average activity factors, resulting in a single power model for all types of tasks. Therefore, with these three considerations, for the general case of having voltage scaling at a cluster level and frequency scaling at a core level, the average power consumption on a CMOS core can be approximated as shown in Equation (3.2),

$$P_{\text{core}}\left(f_{\text{cluster}}, f\right) = \alpha \cdot {f_{\text{cluster}}}^{\gamma-1} \cdot f + \beta \cdot f_{\text{cluster}} + \kappa \tag{3.2}$$

where f is the execution frequency of the core, $\gamma > 1$ is a constant related to the hardware (in CMOS processors, γ is generally modeled equal to 3 [15, 109], such that ${V_{\text{dd}}}^2 \propto {f_{\text{cluster}}}^{\gamma-1}$), and $\alpha > 0$ is also constant (including the effective switching capacitance, the average activity factor of the core, and a scaling factor for the linear relationship between the voltage of the cluster and the highest frequency in the cluster).

The dynamic power consumption is now represented by $\alpha \cdot {f_{\text{cluster}}}^{\gamma-1} \cdot f \geq 0$, the leakage power consumption is represented by $\beta \cdot f_{\text{cluster}} \geq 0$, and $\kappa \geq 0$ represents the independent power consumption, all with the same physical interpretations as in Equation (3.1). Furthermore, in case the core runs at the same frequency which sets the voltage of the cluster, i.e., if $f = f_{\text{cluster}}$, then the power consumption can be rewritten as shown in Equation (3.3).

$$P_{\text{core}}(f) = \alpha \cdot f^{\gamma} + \beta \cdot f + \kappa \tag{3.3}$$

For example, Figure 3.3 presents power consumption values for a 22 nm OOO Alpha 21264 core, based on simulations conducted on gem5 [5] and McPAT [57], for an *x264* application from the PARSEC benchmark suite [4] running a single thread (i.e., a subset of the values presented in Figure 1.3a). As shown in Figure 3.3, these experimental power values can be modeled using Equation (3.3) with power parameters $\gamma = 3$, $\alpha = 0.27 \frac{\text{W}}{\text{GHz}^3}$, $\beta = 0.52 \frac{\text{W}}{\text{GHz}}$, and $\kappa = 0.5\,\text{W}$, resulting in a goodness of fit of: *Sum of Squares due to Error (SSE)* of 1.058, a *Square of the correlation between the response values and the predicted response values (R-square)* of 0.9992, an *Adjusted R-square* of 0.9992 and a *Root Mean Squared Error (RMSE)* of 0.1626.

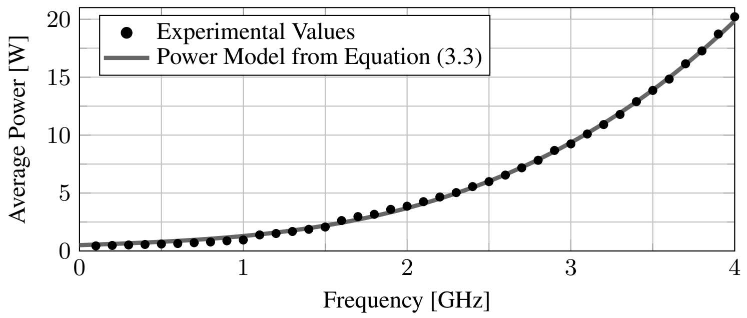


Figure 3.3: Experimental results for a 22 nm OOO Alpha 21264 core, based on simulations using gem5 [5] and McPAT [57] for an *x264* application from the PARSEC benchmark suite [4] running a single thread, and the power model from Equation (3.3) when $\gamma = 3$, $\alpha = 0.27 \frac{\text{W}}{\text{GHz}^3}$, $\beta = 0.52 \frac{\text{W}}{\text{GHz}}$, and $\kappa = 0.5\,\text{W}$.

Similarly, we can use Equation (3.3) to model the experimental results from [30] (research paper on Intel's SCC), which developed a manycore system integrating 48 cores. Figure 12 (Frequency vs. voltage) and Figure 13 (Measured power vs. voltage) from [30] are of special interest, and we summarize its values in Figure 3.4a and Figure 3.4b. Particularly, Figure 3.4a relates several execution frequencies for the cores and their minimum voltages for stable execution, while Figure 3.4b presents power consumption values for running all cores at certain voltages (and at their corresponding maximum frequencies for each voltage). Following, we approximate the table in Figure 3.4a by using a quadratic function. Therefore, having a function that relates frequency and voltage for this chip, we are able to compute new values relating frequency with power based on the values from Figure 3.4b. These new results are divided by 48 (given that the experiments in [30] were conducted for the entire chip, but we are interested in the power of individual cores) and shown in Figure 3.4c. Finally, we approximate the experimental values in Figure 3.4c with the power model from Equation (3.3) using parameters $\gamma = 3$, $\alpha = 1.76 \frac{\text{W}}{\text{GHz}^3}$, $\beta = 0 \frac{\text{W}}{\text{GHz}}$, and $\kappa = 0.5\,\text{W}$, resulting in a goodness of fit of: *Sum of Squares due to Error (SSE)* of 0.05041, a *Square of the correlation between the response values and the predicted response values (R-square)* of 0.9958, an *Adjusted R-square* of 0.9958 and a *Root Mean Squared Error (RMSE)* of 0.07938.

3.4 Energy Model

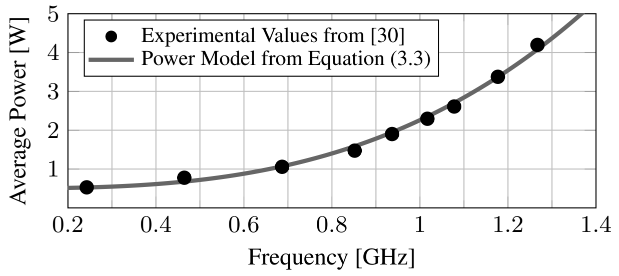
The algorithms presented in Chapter 9 and Chapter 10 only require a very general energy model. As briefly discussed in Chapter 1.1.2, energy is the integration of power through time. Therefore, when a core of type q executes task τ_n at frequency $F_{q,j}^{\text{type}}$ (such that $0 \leq j \leq \hat{F}_q^{\text{type}}$) while consuming constant power during time interval Δt, the energy consumed by the core during interval Δt is computed as $P_q^{\tau_n}\left(F_{q,j}^{\text{type}}\right) \cdot \Delta t$. Moreover, during time interval Δt, a core running at frequency $F_{q,j}^{\text{type}}$ executes a certain amount Δc of core cycles, such that $\Delta t = \frac{\Delta c}{F_{q,j}^{\text{type}}}$. If we now consider that Δc is equal to $e_{q,n}$, then the energy consumption for executing one instance of τ_n in a core of type q at frequency $F_{q,j}^{\text{type}}$ is computed as $P_q^{\tau_n}\left(F_{q,j}^{\text{type}}\right) \cdot \frac{e_{q,n}}{F_{q,j}^{\text{type}}}$. In this way, given that τ_n is executed $\frac{D}{d_n}$ times during one hyper-period, the total energy consumed by τ_n during one hyper-period,

Voltage	Frequency
0.73 V	301.48 MHz
0.75 V	368.82 MHz
0.85 V	569.45 MHz
0.94 V	742.96 MHz
1.04 V	908.92 MHz
1.14 V	1077.11 MHz
1.23 V	1223.37 MHz
1.32 V	1303.79 MHz

(a) Frequency vs. voltage [30]

Voltage	Total Power
0.70 V	25.38 W
0.80 V	37.26 W
0.91 V	50.76 W
1.00 V	70.73 W
1.05 V	91.25 W
1.10 V	110.15 W
1.14 V	125.27 W
1.21 V	161.99 W
1.28 V	201.40 W

(b) Power vs. voltage [30]



(c) Power model for a single core

Figure 3.4: Experimental results from the 48-core system in [30] and the power model from Equation (3.3), when $\gamma=3$, $\alpha=1.76\frac{\text{W}}{\text{GHz}^3}$, $\beta=0\frac{\text{W}}{\text{GHz}}$, and $\kappa=0.5\,\text{W}$.

which we denote as $E_q^{\tau_n}\left(F_{q,j}^{\text{type}}\right)$, is presented in Equation (3.4).

$$E_q^{\tau_n}\left(F_{q,j}^{\text{type}}\right) = \frac{D}{d_n} \cdot P_q^{\tau_n}\left(F_{q,j}^{\text{type}}\right) \cdot \frac{e_{q,n}}{F_{q,j}^{\text{type}}} \tag{3.4}$$

As shown by previous work in the literature (e.g., [11, 45]), when the voltage of a core is always set to the minimum value that achieves stable execution for a given frequency, there exists a *critical frequency* for every application, and this *critical frequency* minimizes the energy consumption for execution when the overhead for entering/leaving a low-power mode can be considered negligible. The critical frequency of task τ_n running on a core of type q is denoted as ${f_{\text{crit}}}_q^{\tau_n}$. Namely, executing at low voltages and frequencies reduces the power consumption, as already seen in Figure 1.3, and as suggested by the power model of Equation (3.3), mainly due to the cubic relationship between the dynamic power consumption and the frequency. However, executing at low frequencies also prolongs the execution time of an application, as already seen in Figure 1.1. Therefore, for task τ_n running on core type q, the critical frequency ${f_{\text{crit}}}_q^{\tau_n}$ represents the frequency below which the energy savings achieved by reducing the power consumption (mainly savings in dynamic energy) are less significant than the associated increments to the energy consumption for prolonging the execution time (mainly due to leakage effects). In simple terms, this means that, for the corresponding type of core, executing an application below its critical frequency is not energy efficient, and it should hence be preferably avoided when the optimization goal is minimizing energy, even if it reduces the power consumption and meets the timing and performance constraints. For example, Figure 1.1 and Figure 1.3 presented execution time, average power consumption, and energy consumption values for executing one instance of five applications from the PARSEC benchmark suite [4], where we can observe the presence of a critical frequency. Particularly, it is energy inefficient to execute both Alpha cores below $0.4\,\text{GHz}$, and to execute the Cortex-A7 cores below $1.4\,\text{GHz}$. Contrarily, since the Cortex-A15 cores cannot execute slower than $1.2\,\text{GHz}$, there is no critical frequency observable in the figure for them, and thus they can be executed as slow as possible while satisfying the timing and performance constraints. Nevertheless, a critical frequency would also be observable for this type of cores if they could be executed at slower frequencies.

Similarly as with the power model described in Section 3.3, a more specific energy model than the described above is required for our theoretical analysis in Chapter 8. Therefore, by considering the power model from Equation (3.2), the energy consumption of a core running at frequency f during time interval Δt and executing a certain amount Δc of core cycles, such that $\Delta t = \frac{\Delta c}{f}$, can be computed as presented in Equation (3.5).

$$E_{\text{core}}\left(f_{\text{cluster}}, f\right) = \left(\alpha \cdot {f_{\text{cluster}}}^{\gamma-1} \cdot f + \beta \cdot f_{\text{cluster}} + \kappa\right) \frac{\Delta c}{f} \tag{3.5}$$

Furthermore, for the case in which the core runs at the frequency which decides the voltage of the cluster, i.e., $f = f_{\text{cluster}}$, the energy consumption on the core can be written as shown in Equation (3.6), which is a convex function with respect to f.

$$E_{\text{core}}\left(f\right) = \left(\alpha \cdot f^{\gamma} + \beta \cdot f + \kappa\right) \frac{\Delta c}{f} \tag{3.6}$$

Therefore, by setting the first-order derivative of Equation (3.6) with respect to f to zero, the minimum value for $E_{\text{core}}(f)$ can be found when f is equal to $\sqrt[\gamma]{\frac{\kappa}{(\gamma-1)\alpha}}$. In this way, the value of the critical frequency for the energy model in Equation 3.6, simply denoted as f_{crit} (given that in Chapter 8 we focus on homogeneous systems and assume that all tasks have similar average activity factors), is computed as shown in Equation (3.7).

$$f_{\text{crit}} = \sqrt[\gamma]{\frac{\kappa}{(\gamma-1)\,\alpha}} \tag{3.7}$$

For example, Figure 3.5 illustrates function $E_{\text{core}}(f)$ from Equation (3.6) for a single core executing 10^9 computer cycles with the same power parameters derived from the experimental results in [30] for Figure 3.4c, where we can observe that there exist a critical frequency at $0.52\,\text{GHz}$.

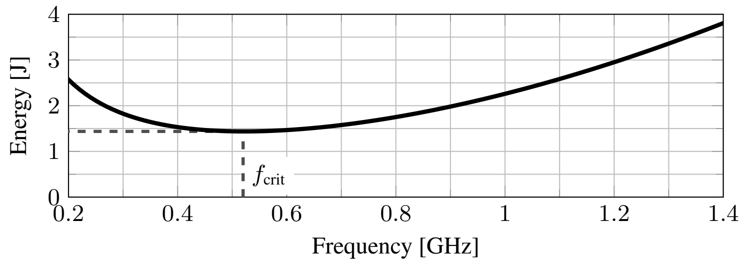


Figure 3.5: Energy consumption for a single core from [30] executing 10^9 computer cycles, when using the energy model from Equation (3.6) with $\gamma = 3$, $\alpha = 1.76 \frac{\text{W}}{\text{GHz}^3}$, $\beta = 0 \frac{\text{W}}{\text{GHz}}$, and $\kappa = 0.5\,\text{W}$, resulting in a critical frequency of $0.52\,\text{GHz}$.

With respect to the voltage of the cluster, so far in this dissertation we have claimed that it is energy inefficient to execute cores at voltages higher than the minimum required voltage for stable execution of the desired frequency. Following, we justify such a statement based on the power and energy models of Equation (3.1) and Equation (3.5). According to Equation (3.1), the dynamic power consumption has a quadratic relationship with the supply voltage and a linear relationship with the execution frequency. Therefore, for a fixed supply voltage, reducing the execution frequency of a core will linearly decrease the dynamic power consumption while linearly incrementing the execution time, but it will have *no effect on the dynamic energy consumption*. Contrarily, when modeling I_{leak} for the critical temperature, the leakage power consumption depends only on the voltage (not on the frequency), while the independent power consumption is both voltage and frequency independent. In this way, reducing the execution frequency without changing the voltage will merely result in an increment of the leakage and independent energy consumptions, mainly due to the increase in the execution time while not changing the leakage and independent power. To illustrate this discussion, focusing now on Equation (3.5), Figure 3.6 shows several examples of energy consumption values with respect to the frequency when considering different values of f_{cluster} (which is equivalent to considering different supply voltages, since f_{cluster} sets the voltage of the cluster).

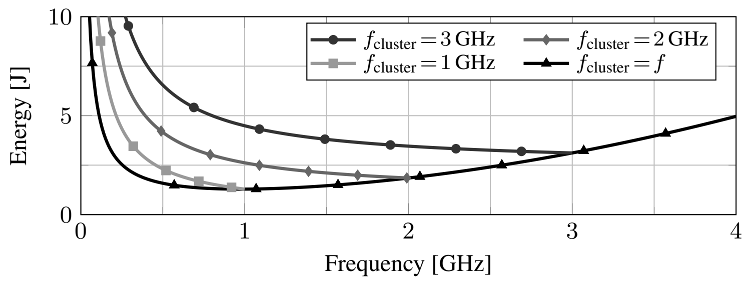


Figure 3.6: Energy consumption examples for a core executing $\Delta c = 10^9$ compute cycles, considering four different cases for the voltage (i.e., different values for f_{cluster}), for the model in Equation (3.5) with $\gamma = 3$, $\alpha = 0.27 \frac{\text{W}}{\text{GHz}^3}$, $\beta = 0.52 \frac{\text{W}}{\text{GHz}}$, and $\kappa = 0.5\,\text{W}$.

Finally, every core consumes some energy during the transition process of entering and leaving a low-power mode. Given that Chapters 8, 9, and 10 focus on periodic tasks, which after entering a low-power mode will always go back to execution mode after a certain amount of time, we define the *overheads for sleeping* as the summation of the energy consumption for both for entering and leaving a low-power mode. Therefore, when the duration of the waiting interval between the time at which a task instance finished all the workload in its ready queue and the time at which a new task instance arrives to the system is short enough, keeping a core idle is more energy efficient than entering/leaving a low-power mode. On the other hand, when the waiting interval is sufficiently long, entering (and later leaving) a low-power mode results in higher energy savings than idling. Hence, we define the *break-even time* as the time such that the energy consumption for keeping a core idling is equal to the energy overheads for sleeping.

3.5 Thermal Model

The most widely adopted model used in electronics for thermal modeling are RC thermal networks, which are motivated by the well-known duality between thermal and electrical circuits [33], which we also adopt in this dissertation. In an RC thermal network, the different parts of the chip and cooling solution are represented by N thermal nodes (electrical nodes in an electrical circuit), such that there are at least as many thermal nodes as blocks in the floorplan, i.e., $N \geq Z$. The temperature associated to each thermal node (with unit *Kelvin* $[K]$) is represented by the voltage on the node. Thermal nodes are interconnected between each other through thermal conductances (with units *Watts per Kelvin* $\left[\frac{W}{K}\right]$). Heat transfer (or heat flow) among cores and other elements of the chip is represented by the currents flowing through the thermal conductances. There is a thermal capacitance associated to every thermal node which accounts for the transient thermal effects. The ambient temperature is represented by another thermal node denoted as T_{amb}, and there is no thermal capacitance associated with it as the ambient temperature is considered to be constant for long periods of time. The power consumption of the cores and other elements on the chip correspond to sources of heat on the chip (with unit *Watt* $[W]$), represented by current sources in the electrical circuit. With these considerations, the temperatures throughout the chip are modeled as a function of the ambient temperature, the power consumptions inside the chip, and by considering the heat transfer among neighboring elements on the chip.

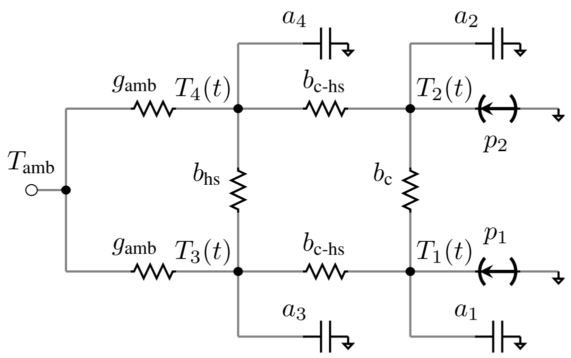


Figure 3.7: (from [75]) Simplified RC thermal network example for 2 cores, where we assume that cores are in immediate contact with the heat sink, and there is no other connection between a core and the ambient temperature. In a more detailed example, we would have several layers between a core and the heat sink (e.g., as seen in Figure 2.3, the ceramic packaging substrate, the thermal interface material, the heat spreader, etc.), and there would also exist more paths that lead to the ambient temperature (e.g., through the circuit board).

Figure 3.7 shows a simplified example of an RC thermal network for a chip with two cores. In Figure 3.7, $T_1(t)$ and $T_2(t)$ are the voltages representing the temperatures on core 1 and core 2, respectively. Voltages $T_3(t)$ and $T_4(t)$ represent the temperatures on the heat sink nodes directly above core 1 and core 2, respectively. Current supplies p_1 and p_2 represent the power consumptions on core 1 and core 2, respectively. For the heat transfer among thermal nodes, b_{c} represents the thermal conductance between core 1 and core 2, $b_{\text{c-hs}}$ represents the thermal conductance between a core and the heat sink, b_{hs} represents the thermal conductance between nodes of the heat sink, and g_{amb} represents the thermal conductance between a heat sink node and the ambient temperature. Finally, the thermal capacitances of thermal node i is represented by capacitor a_i. By using Kirchoff's first law and linear algebra, we can derive a system of first-order differential equations for the example in Figure 3.7. Particularly,

$$\begin{cases} p_1 - (T_1(t) - T_3(t))\, b_{\text{c-hs}} + (T_2(t) - T_1(t))\, b_{\text{c}} - a_1 \dfrac{\mathrm{d}T_1(t)}{\mathrm{d}t} = 0 \\ p_2 - (T_2(t) - T_4(t))\, b_{\text{c-hs}} - (T_2(t) - T_1(t))\, b_{\text{c}} - a_2 \dfrac{\mathrm{d}T_2(t)}{\mathrm{d}t} = 0 \\ (T_1(t) - T_3(t))\, b_{\text{c-hs}} + (T_4(t) - T_3(t))\, b_{\text{hs}} - a_3 \dfrac{\mathrm{d}T_3(t)}{\mathrm{d}t} - (T_3(t) - T_{\text{amb}})\, g_{\text{amb}} = 0 \\ (T_2(t) - T_4(t))\, b_{\text{c-hs}} - (T_4(t) - T_3(t))\, b_{\text{hs}} - a_4 \dfrac{\mathrm{d}T_4(t)}{\mathrm{d}t} - (T_4(t) - T_{\text{amb}})\, g_{\text{amb}} = 0. \end{cases}$$

This system of first-order differential equations can be rewritten in matrix and vector form as

$$\begin{bmatrix} (b_{\text{c-hs}} + b_{\text{c}}) & -b_{\text{c}} & -b_{\text{c-hs}} & 0 \\ -b_{\text{c}} & (b_{\text{c-hs}} + b_{\text{c}}) & 0 & -b_{\text{c-hs}} \\ -b_{\text{c-hs}} & 0 & (b_{\text{c-hs}} + b_{\text{hs}} + g_{\text{amb}}) & -b_{\text{hs}} \\ 0 & -b_{\text{c-hs}} & -b_{\text{hs}} & (b_{\text{c-hs}} + b_{\text{hs}} + g_{\text{amb}}) \end{bmatrix} \begin{bmatrix} T_1(t) \\ T_2(t) \\ T_3(t) \\ T_4(t) \end{bmatrix} +$$

$$\begin{bmatrix} a_1 & 0 & 0 & 0 \\ 0 & a_2 & 0 & 0 \\ 0 & 0 & a_3 & 0 \\ 0 & 0 & 0 & a_4 \end{bmatrix} \begin{bmatrix} T_1'(t) \\ T_2'(t) \\ T_3'(t) \\ T_4'(t) \end{bmatrix} = \begin{bmatrix} p_1 \\ p_2 \\ 0 \\ 0 \end{bmatrix} + T_{\text{amb}} \begin{bmatrix} 0 \\ 0 \\ g_{\text{amb}} \\ g_{\text{amb}} \end{bmatrix}.$$

Therefore, in condensed matrix and vector form, the system of first-order differential equations of an RC thermal network with N thermal nodes can be expressed as shown in Equation (3.8)

$$\mathbf{A}\mathbf{T}' + \mathbf{B}\mathbf{T} = \mathbf{P} + T_{\text{amb}}\mathbf{G} \tag{3.8}$$

where matrix $\mathbf{A} = [a_{i,j}]_{N \times N}$ contains the thermal capacitance values (generally a diagonal matrix, since thermal capacitances are modeled to ground), matrix $\mathbf{B} = [b_{i,j}]_{N \times N}$ contains the thermal conductance values between vertical and lateral neighboring nodes, column vector $\mathbf{T} = [T_i\,(t)]_{N \times 1}$ represents the temperatures on the thermal nodes, column vector $\mathbf{T}' = [T_i'\,(t)]_{N \times 1}$ accounts for the first-order derivative of the temperature on each thermal node with respect to time, column vector $\mathbf{P} = [p_i]_{N \times 1}$ contains the values of the power consumption on every node, and column vector $\mathbf{G} = [g_i]_{N \times 1}$ contains the values of the thermal conductances between each node and the ambient temperature. In case that thermal node i is not in contact with the ambient temperature, e.g., the temperature of a core, the value of g_i is set to zero. Furthermore, Equation (3.8) can be rephrased as

$$\mathbf{T}' = \mathbf{C}\mathbf{T} + \mathbf{A}^{-1}\mathbf{P} + T_{\text{amb}}\mathbf{A}^{-1}\mathbf{G} \qquad \text{with } \mathbf{C} = -\mathbf{A}^{-1}\mathbf{B}. \tag{3.9}$$

In order to solve the system of first-order differential equations in Equation (3.8) or Equation (3.9), we need to have a set of initial conditions (as is the case when solving any differential equation). For such a purpose, we define column vector $\mathbf{T}_{\text{init}} = [T_{\text{init}\,i}]_{N \times 1}$ which contains the initial temperatures on all nodes at time zero, i.e., $t = 0\,\text{s}$.

When only considering the steady-state temperatures, Equation (3.8) becomes

$$\mathbf{B}\mathbf{T}_{\text{steady}} = \mathbf{P} + T_{\text{amb}}\mathbf{G} \quad \text{or} \quad \mathbf{T}_{\text{steady}} = \mathbf{B}^{-1}\mathbf{P} + T_{\text{amb}}\mathbf{B}^{-1}\mathbf{G}$$

where column vector $\mathbf{T}_{\text{steady}} = \left[T_{\text{steady}_i}\right]_{N \times 1}$ represents the steady-state temperatures on all thermal nodes, and $\mathbf{B}^{-1} = [\tilde{b}_{i,j}]_{N \times N}$ is the inverse of matrix $\mathbf{B}$. Furthermore, when only focusing on the steady-state temperature of node i, denoted T_{steady_i}, we have that

$$T_{\text{steady}_i} = \sum_{j=1}^{N} \tilde{b}_{i,j} \cdot p_j + T_{\text{amb}} \cdot \sum_{j=1}^{N} \tilde{b}_{i,j} \cdot g_j \tag{3.10}$$

where $\tilde{b}_{i,j} \cdot p_j$ represents the amount of heat contributed by thermal node j into the steady-state temperature of node i.

In practice, for a specific floorplan and cooling solution, the parameters of the associated thermal network can be computed by using a modeling tool like HotSpot [33] (which would derive a detailed RC thermal network equivalent to the one used internally by HotSpot to conduct thermal simulations). Furthermore, as discussed in Chapter 1.3, a more practical alternative would be through thermal profiling, by using some thermal modeling method that relies on real measurements on a specific chip and cooling solution, e.g., [17]. Such an option is well suited to deal with modeling errors, process variations on the chip, imperfections on heat sink, etc. Similarly, given that having different fan speeds changes the thermal conductivity of the heat sink, a method like the one presented in [17] could also help to adapt the thermal model at runtime, or to derive multiple thermal models to choose from (one for every fan speed).

In regards to vector $\mathbf{P}$, we can divide it into three column vectors, specifically, $\mathbf{P}^{\text{cores}}$, $\mathbf{P}^{\text{blocks}}$, and $\mathbf{P}^{\text{rest}}$, such that $\mathbf{P} = \mathbf{P}^{\text{cores}} + \mathbf{P}^{\text{blocks}} + \mathbf{P}^{\text{rest}}$. Column vector $\mathbf{P}^{\text{cores}} = \left[p_i^{\text{cores}}\right]_{N\times 1}$ represents the power consumption on the cores. Column vector $\mathbf{P}^{\text{blocks}} = \left[p_i^{\text{blocks}}\right]_{N\times 1}$ represents the power consumption on other blocks in the floorplan that do not correspond to cores, (e.g., the L2 caches for the manycore system in Figure 3.2, as explained in Section 3.2). Column vector $\mathbf{P}^{\text{rest}} = \left[p^{\text{rest}}\right]_{N\times 1}$ represents the power consumption of the rest of the thermal nodes (e.g., internal nodes of the heat sink), for which it holds that $p_i^{\text{rest}} = 0$ for all i. In this way, we have that

$$\mathbf{T}_{\text{steady}} = \mathbf{B}^{-1}\mathbf{P}^{\text{cores}} + \mathbf{B}^{-1}\mathbf{P}^{\text{blocks}} + T_{\text{amb}}\mathbf{B}^{-1}\mathbf{G}$$

and

$$T_{\text{steady}_i} = \sum_{j=1}^{N} \tilde{b}_{i,j} \cdot p_j^{\text{cores}} + \sum_{j=1}^{N} \tilde{b}_{i,j} \left(p_j^{\text{blocks}} + T_{\text{amb}} \cdot g_j\right). \tag{3.11}$$

For notational brevity, we define set $\mathbf{L} = \{\ell_1, \ell_2, \ldots, \ell_Z\}$, such that the elements in set $\mathbf{L}$ include all the indexes of the thermal nodes that correspond to blocks in the floorplan (as opposed to thermal nodes that represent the heat sink, internal nodes of the heat spreader, the thermal interface material, etc.), and thus for block ℓ_j it should hold that $1 \leq \ell_j \leq N$. For example, if thermal node i corresponds to block j in the floorplan, then the steady-state temperature on node i (i.e., T_{steady_i}) is also indexed as $T_{\text{steady}_{\ell_j}}$. Similarly, we define set $\mathbf{K} = \{k_1, k_2, \ldots, k_M\}$, such that set $\mathbf{K}$ contains all the indexes of the thermal nodes that correspond to cores (among all cores, ignoring the types of the cores), and thus for core j it should hold that $1 \leq k_j \leq N$. For example, if thermal node i corresponds to core j, then the steady-state temperature on node i (i.e., T_{steady_i}) is also indexed as $T_{\text{steady}_{k_j}}$. For considering core heterogeneity, we also define sets $\mathbf{K}^q = \left\{k_1^q, k_2^q, \ldots, k_{M_q^{\text{type}}}^q\right\}$ for all core types $q = 1, 2, \ldots, Q$, such that set $\mathbf{K}^q$ contains the indexes of the thermal nodes that correspond to cores of type q.

Chapter 4

Experimental Framework

Throughout this dissertation we conduct several experimental evaluations based on a common simulation framework, for which Figure 4.1 presents an overview. The framework has two modes: (1) a *detailed mode* for evaluations requiring detailed transient temperature information and a runtime feedback loop, mostly used for evaluating performance optimization techniques; and (2) a *high-level mode* that does not include the runtime feedback loop, but which can simulate a very large number of applications, mostly used for evaluating energy optimization techniques.

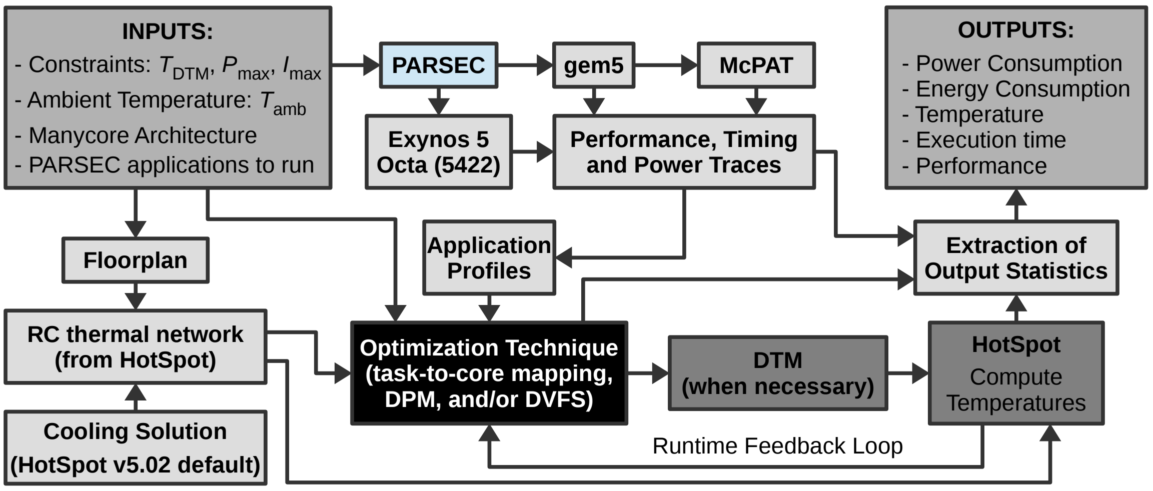

INPUTS:
- Constraints: T_{DTM}, $P_{\max}$, $I_{\max}$
- Ambient Temperature: T_{amb}
- Manycore Architecture
- PARSEC applications to run
PARSEC
gem5
McPAT
OUTPUTS:
- Power Consumption
- Energy Consumption
- Temperature
- Execution time
- Performance
Exynos 5 Octa (5422)
Performance, Timing and Power Traces
Floorplan
Application Profiles
Extraction of Output Statistics
RC thermal network (from HotSpot)
Optimization Technique (task-to-core mapping, DPM, and/or DVFS)
DTM (when necessary)
HotSpot Compute Temperatures
Cooling Solution (HotSpot v5.02 default)
Runtime Feedback Loop

Figure 4.1: Overview of our simulation framework.

4.1 Setup

4.1.1 Detailed Mode Setup

In the detailed mode, our experimental evaluations are conducted considering detailed transient thermal effects, such that we can conduct accurate full system simulations. Particularly, our simulation framework integrates gem5 [5], McPAT [57], real measurements of an Exynos 5 Octa (5422) processor [92] conducted on the Odroid-XU3 [26] mobile platform, and HotSpot [33], in a closed-loop transient simulator with a runtime feedback loop. For the simulations with gem5 and McPAT, applications from the PARSEC benchmark suite [4] are first executed in gem5 for all possible frequency configurations on the different types of cores, as well as by considering different number of parallel threads for each application. The output statistics from gem5 are then parsed, stored as traces for later use, and also used to generate the *xml* files required as inputs by McPAT, point at which we add the voltage information to the *xml* files (not required before by gem5).

Following, we also parse the output results from McPAT in order to obtain the necessary power consumption data for every part of the chip, again stored as traces for later use. For the measurements on the Odroid-XU3 platform, we simply execute the PARSEC applications for all possible frequency configurations and by considering different number of parallel threads for each application, and we measure the resulting execution time and power consumption values. Figure 1.1 and Figure 1.3 have already shown experimental results obtained with this simulation framework, where it can be observed that different applications have different average power consumptions, and the specific power values depend on the application, the DVFS levels, and the selected number of threads (this last point not shown in Figures 1.1 and 1.3). Finally, all this power data is then fed to HotSpot such that we can conduct the transient thermal simulations.

For each architecture under consideration (later detailed in Section 4.2), we consider one thermal block for each core and independent thermal blocks for other hardware (e.g., L2 caches, memory controllers, NoC routers, etc.). We then obtain the values for $\mathbf{A}$, $\mathbf{B}$, $\mathbf{B}^{-1}$, $\mathbf{G}$, and $\mathbf{C}$, by using HotSpot v5.02 with its default configuration in the block model mode. Namely, a chip thickness of $0.15\,\text{mm}$, a silicon thermal conductivity of $100\,\frac{\text{W}}{\text{m}\cdot\text{K}}$, a silicon specific heat of $1.75\cdot 10^6\frac{\text{J}}{\text{m}^3\cdot\text{K}}$, a heat sink of $6\times 6\,\text{cm}$ and $6.9\,\text{mm}$ thick, a heat sink convection capacitance of $140.4\,\frac{\text{J}}{\text{K}}$, a heat sink convection resistance of $0.1\,\frac{\text{K}}{\text{W}}$, a heat sink and heat spreader thermal conductivity of $400\,\frac{\text{W}}{\text{m}\cdot\text{K}}$, a heat sink and heat spreader specific heat of $3.55\cdot 10^6\,\frac{\text{J}}{\text{m}^3\cdot\text{K}}$, a heat spreader of 3×3 cm and 1 mm thick, an interface material thickness of 20 µm, an interface material thermal conductivity of $4\,\frac{\text{W}}{\text{m}\cdot\text{K}}$, and an interface material specific heat of $4\cdot 10^6\,\frac{\text{J}}{\text{m}^3\cdot\text{K}}$. Furthermore, we consider that the ambient temperature is 45°C, the maximum electrical power constraint for the chip (i.e., P_{max}) is $250\,\text{W}$, and the critical threshold temperature that triggers DTM (i.e., T_{DTM}) is 80°C.

Whenever we evaluate a technique that does not have its own dynamic thermal control using thermal sensors (e.g., when we evaluate a technique other than Intel's Turbo Boost), we consider that the chip is integrated with a standard reactive control-based closed-loop DTM technique [34], which is triggered when T_{DTM} is exceeded anywhere in the chip. Particularly, similar to the DTM technique detailed in [34], when DTM is triggered, it reduces the DVFS levels of all active cores, one step at a time, by using a control period of 1 ms. Once the maximum temperature in the chip drops back below T_{DTM}, the DVFS levels of the cores are increased one step at a time, by using the same 1 ms control period, until all cores reach their nominal operation levels. In our simulations, this voltage and frequency reduction/increment is achieved by changing the DVFS levels in gem5 and McPAT or in the Odroid-XU3 platform, or more specifically, by reading the trace information for the corresponding DVFS levels. Therefore, these changes in the DVFS levels translate to different execution times for each application thread and also on different power consumption values. Thanks to the runtime feedback loop in our simulation framework, these new power consumption data is fed to HotSpot for the next simulation steps, hence changing the resulting transient temperatures. On the other hand, when evaluating Intel's Turbo Boost, the transient simulations are conducted in a very similar manner. The only difference is that Turbo Boost will always attempt to increase the DVFS levels whenever the temperature everywhere in the chip is below the critical threshold temperature, not stopping at the nominal operation levels, as already explained in detail in Chapter 2.1.2.

As an additional comment with respect to the implemented DTM technique for our simulations, it should be noted that although decreasing the DVFS levels of *all* the cores in the chip when maybe only *one* core has a temperature above T_{DTM} might seem a pessimistic approach, in most practical cases it will not be as pessimistic as it seems. For example, for the floorplan illustrated in Figure 3.2, Figure 4.2 presents a temperature snapshot for a specific mapping with 12 active cores, in which each active core has a power consumption of 18.75 W (225 W in total). For such a case, the highest steady-state temperature among all cores is 102.9°C. Nevertheless, the lowest steady-state temperature among the active cores is 97.0°C, which is still much higher than the critical temperature of 80.0°C. Therefore, in our experiments we conservatively assume that DTM is triggered for all active cores without incurring in much pessimism.

4.1.2 High-level Mode Setup

In the high-level mode of our simulation framework, we do not include the runtime feedback loop for the transient temperature simulations. Namely, the execution time and power traces for the different frequency configurations and number of parallel threads for each application are obtained using gem5, McPAT, and the Odroid-XU3 mobile platform, in the same way as described in Section 4.1.1. This collection of traces information is then used to compute the peak power consumption, energy consumption, and steady-state

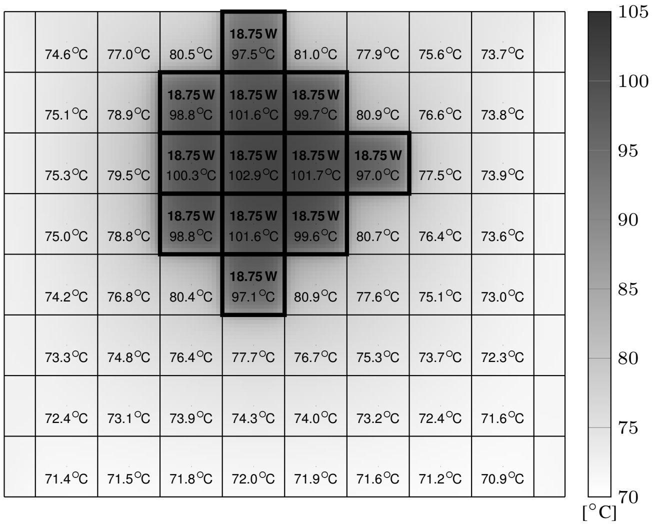


Figure 4.2: (from [76]) Temperature snapshot for the floorplan illustrated in Figure 3.2, for a specific mapping with 12 active cores in which each active core has a power consumption of 18.75 W (225 W in total), resulting in a highest steady-state temperature of 102.9°C and a lowest steady-state temperature (among the active cores) of 97.0°C. Top numbers are the power consumptions of each active core (boxed in black). Bottom numbers are the temperatures in the center of each core. Detailed temperatures are shown according to the color bar.

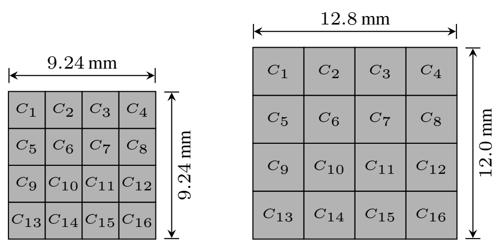
temperatures of any mapping, scheduling, and DVFS decision which we wish to evaluate. Therefore, the high-level mode merely avoids conducting the transient temperature simulations, which are the most computational intensive part of our simulations once the execution time and power traces have been obtained. In this way, the high-level mode is able to run simulations for a very large number of applications and workload scenarios in a reasonable time (just a few seconds for each simulation), which is not possible in the detailed mode due to the long time required to finish a single simulation (in the order of several hours).

4.2 Architectures

In this section we describe the architectures used throughout the dissertation for most of the presented motivational examples and experimental evaluations.

4.2.1 Homogeneous architectures

For the motivational examples in Chapters 5, 6, 7, we consider two simplified hypothetical manycore systems with 16 homogeneous cores simulated with gem5 and McPAT, arranged in 4 rows and 4 columns, as shown in Figure 4.3. The chip shown in Figure 4.3a considers *simple in-order* Alpha 21264 cores in 45 nm technology with a resulting size of 2.31×2.31 mm, while the chip shown in Figure 4.3b considers OOO Alpha 21264 cores in 22 nm technology with a resulting size of 3.2 mm $\times$ 3.0 mm.


9.24 mm
9.24 mm
C_1 C_2 C_3 C_4
C_5 C_6 C_7 C_8
C_9 C_{10} C_{11} C_{12}
C_{13} C_{14} C_{15} C_{16}
12.8 mm
12.0 mm
C_1 C_2 C_3 C_4
C_5 C_6 C_7 C_8
C_9 C_{10} C_{11} C_{12}
C_{13} C_{14} C_{15} C_{16}

(a) *simple* Alpha 21264 cores (45 nm)

(b) OOO Alpha 21264 cores (22 nm)

Figure 4.3: Floorplans of two homogeneous 16-core systems used for the motivational examples in Chapters 5, 6, 7, based on simulations in gem5 [5] and McPAT [57].

For our experiments on clustered homogeneous architectures, we consider a system very similar to Intel's SCC [36], particularly, the homogeneous 64-core system already illustrated in Figure 3.2 and briefly discussed in Chapter 3.2, based on simulations conducted on gem5 [5] and McPAT [57] for OOO Alpha 21264 cores in 22 nm technology. We assume a DVFS granularity at a cluster level and consider that every cluster is composed of 4 cores, a shared 2 MB L2 cache, and a memory controller. According to our simulations and as shown in Figure 3.2, every core has area of $9.6\,\text{mm}^2$, and the combined area of the shared L2 cache and memory controller in each cluster is $4.7\,\text{mm}^2$. All the clusters are connected to a 512 MB RAM (executing at 1 GHz, with 73 GB/s memory bandwidth). When using HotSpot to derive the RC thermal network associated to the floorplan of such a chip, we consider one block for every core (not one block for every internal unit inside each core), while having independent blocks for the L2 cache and the memory controllers. Finally, we assume that every cluster has available frequencies $\{0.2, 0.4, \ldots, 4.0\}$ GHz, and consider that the minimum voltages for stable execution of each frequency are taken from the work in [22], as already presented in Figure 1.7.

4.2.2 Heterogeneous architectures

For our experiments on clustered heterogeneous architectures, we consider the 72-core system presented in Figure 4.4, consisting of 24 OOO Alpha 21264 cores (arranged in three clusters of eight cores) and 16 *simple in-order* Alpha 21264 cores (arranged in four clusters of four cores), based on the simulations on gem5 [5] and McPAT [57] for 22 nm technology, and 16 *in-order* Cortex-A7 cores and 16 OOO Cortex-A15 cores (both cases arranged in four clusters of four cores), based on an Odroid-XU3 [26] mobile platform with an Exynos 5 Octa (5422) [92] chip with ARM's big.LITTLE architecture. According to our simulations, each OOO Alpha core has an area of $9.6\,\text{mm}^2$, and every OOO Alpha cluster with eight cores has a shared 2 MB L2 cache and a memory controller. Moreover, each *simple* Alpha core has an area of $1.6\,\text{mm}^2$, and every *simple* Alpha cluster with four cores has a shared 2 MB L2 cache and a memory controller. All OOO and *simple* Alpha clusters are connected to a 512 MB RAM (executing at 1 GHz, with 73 GB/s memory bandwidth). The areas of the Cortex-A7 and Cortex-A15 cores are estimated from die figures of the Exynos 5 Octa (5422) [92] chip as $0.8\,\text{mm}^2$ and $5.0\,\text{mm}^2$, respectively. There is a shared 512 kB L2 cache in every Cortex-A7 cluster with four cores, and a shared 2 MB L2 cache in every Cortex-A15 cluster with four cores, all connected through two low-power multi-layer 32 bit buses to a 2 GB LPDDR3 RAM PoP (executing at 933 MHz, with 14.9 GB/s memory bandwidth). For the OOO and *simple* Alpha clusters, we assume available frequencies $\{0.2, 0.4, \ldots, 4.0\}$ GHz, and consider that the minimum voltages for stable execution of each frequency are taken from the work in [22], as presented in Figure 1.7. For the Cortex-A7 and Cortex-A15 clusters, the available frequencies in the Odroid-XU3 platform are $\{0.2, 0.3, \ldots, 1.4\}$ GHz and $\{1.2, 1.3, \ldots, 2.0\}$ GHz, respectively, and the voltages for each frequency are automatically selected by the chip.

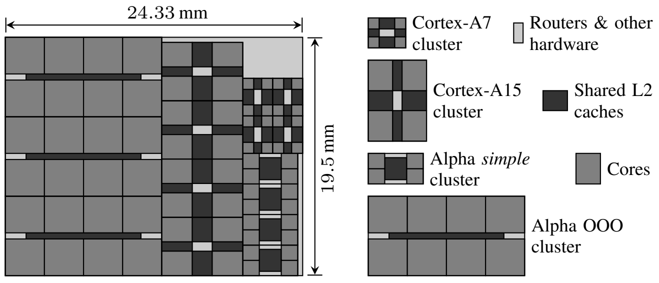


Figure 4.4: (from [76]) Floorplan of a heterogeneous 72-core system based on simulations conducted using gem5 [5] and McPAT [57], and an Odroid-XU3 [26] mobile platform with an Exynos 5 Octa (5422) [92] chip. For a given cluster type, every cluster is identified as a, b, c, and d, from left to right and from top to bottom.

4.3 Benchmarks

For benchmarks, we use realistic applications from the PARSEC benchmark suite [4], where every application can run $1, 2, \ldots, 8$ parallel dependent threads. For all our experiments, in order to maintain the system busy, we consider that every time an application instance finishes, another instance is immediately executed.

When focusing on real-time tasks or performance-constrained applications, we assume that every task consists of one randomly-selected PARSEC application running a single thread (i.e., using a uniform distribution, we randomly choose which specific application, among all PARSEC applications, is executed by each task), to which we randomly assign a deadline and period, such that every application instance must finish before its deadline.

Chapter 5

Thermal Safe Power (TSP)

5.1 Overview

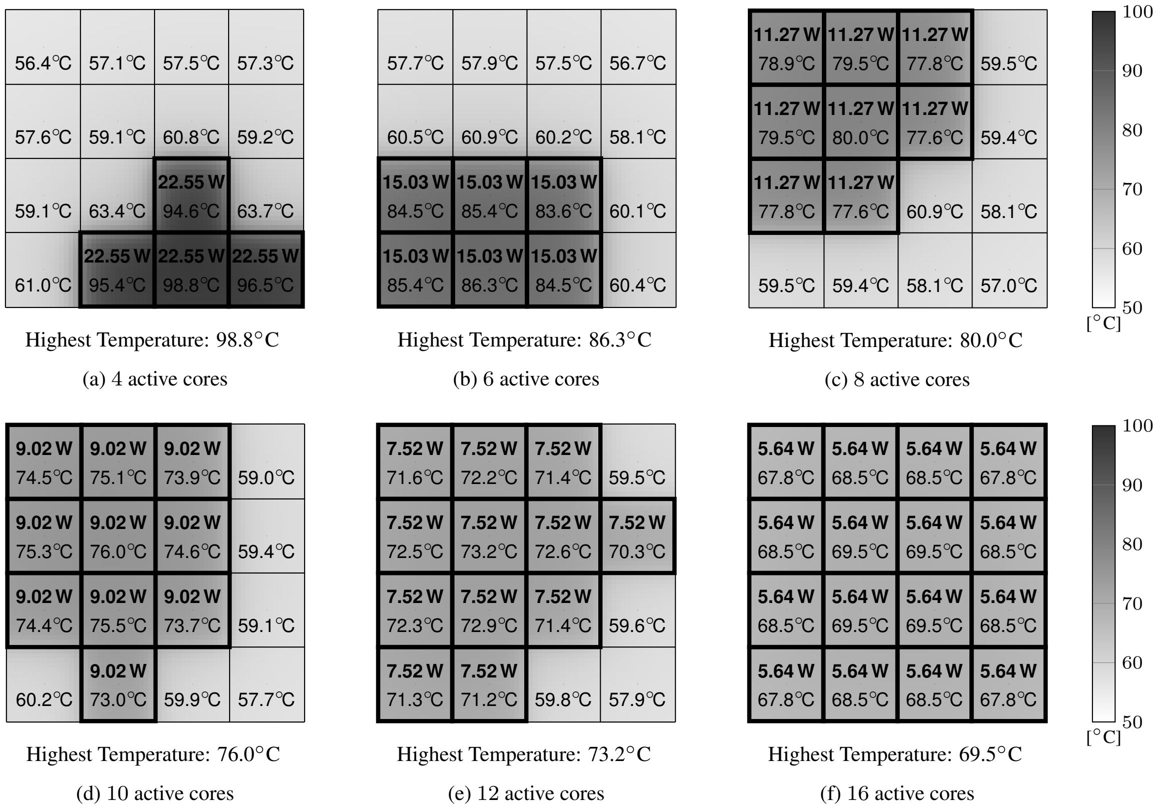
In this chapter, we present a novel power budget concept called Thermal Safe Power (TSP) [75, 76, 77]. TSP is an abstraction which provides *safe* and *efficient* per-core power budget values as a function of the number of simultaneously active cores. According to the number of cores that are active at a certain time, executing the cores in a way that their power consumptions reach TSP will result in a maximum temperature across the chip which is just below the critical threshold temperature that triggers DTM. In this way, TSP conceptually changes the typical design that uses a *single* and *constant* value as per-chip or per-core power budget, e.g., TDP. Furthermore, TSP can serve as a fundamental tool for guiding task partitioning, core mapping, DPM, and DVFS algorithms on their attempt to achieve high predictable performance under thermal constraints.

Based on the RC thermal network of a specific chip and cooling solution (as discussed in Chapter 3.5), this chapter presents polynomial-time algorithms to compute TSP for both *homogeneous* and *heterogeneous* manycore systems. For simplicity of presentation and introduction of the TSP concept, we start by presenting the special case for handling homogeneous manycore cores. We then present the algorithms for the general case, which are suitable for heterogeneous manycore systems. In Section 5.2.2 and Section 5.3.2, we present a polynomial-time method for computing the values of TSP for the worst-case mappings of active cores. That is, we derive an algorithm for computing a per-core power budget for a given number of simultaneously active cores, that will be thermally safe for any possible mapping with that number of active cores, thus allowing system designers to abstract from core mapping decisions, and that will also have little pessimism which could lead to underutilized resources and a large thermal headroom. Furthermore, in Section 5.2.1 and Section 5.3.1 we present another method that is able to compute the TSP values for a particular mapping of active cores and ambient temperature, which can be used at runtime, thus accounting for changes in the mapping decisions and further reducing any existing thermal headroom.

Open-Source Tools: The algorithms to compute TSP are implemented as an open-source tool available for download at ces.itec.kit.edu/download.

5.1.1 Motivational Example

In this section we present a motivational example providing some insight on the drawbacks of using *single* and *constant* per-chip or per-core power budgets as abstractions from thermal issues. For simplicity of presentation, consider the 16-core homogeneous system presented in Figure 4.3a (described in Chapter 4.2.1). Assume a critical temperature that should not be exceeded of 80°C, and consider that DTM is deactivated and will therefore not be triggered in this example. Following, consider a *constant* per-chip power budget of 90.2 W. Conducting simulations with HotSpot, Figure 5.1 presents the resulting steady-state temperatures on the chip when equally distributing the 90.2 W per-chip budget among all active cores, for different number of simultaneously active cores. From the figure, we can observe that the only case in which the maximum steady-state temperature among all cores reaches the critical temperature of 80°C (without exceed it and also without leaving thermal headroom) is when equally distributing the power budget among 8 cores. However, this power budget is either pessimistic or not thermally safe when equally distributing the budget among more than 8 cores or among less than 8 cores, respectively.



56.4°C 57.1°C 57.5°C 57.3°C
57.6°C 59.1°C 60.8°C 59.2°C
59.1°C 63.4°C 22.55 W 94.6°C 63.7°C
61.0°C 22.55 W 95.4°C 22.55 W 98.8°C 22.55 W 96.5°C
Highest Temperature: 98.8°C
(a) 4 active cores
57.7°C 57.9°C 57.5°C 56.7°C
60.5°C 60.9°C 60.2°C 58.1°C
15.03 W 84.5°C 15.03 W 85.4°C 15.03 W 83.6°C 60.1°C
15.03 W 85.4°C 15.03 W 86.3°C 15.03 W 84.5°C 60.4°C
Highest Temperature: 86.3°C
(b) 6 active cores
11.27 W 78.9°C 11.27 W 79.5°C 11.27 W 77.8°C 59.5°C
11.27 W 79.5°C 11.27 W 80.0°C 11.27 W 77.6°C 59.4°C
11.27 W 77.8°C 11.27 W 77.6°C 60.9°C 58.1°C
59.5°C 59.4°C 58.1°C 57.0°C
Highest Temperature: 80.0°C
(c) 8 active cores
100
90
80
70
60
50
[°C]
9.02 W 74.5°C 9.02 W 75.1°C 9.02 W 73.9°C 59.0°C
9.02 W 75.3°C 9.02 W 76.0°C 9.02 W 74.6°C 59.4°C
9.02 W 74.4°C 9.02 W 75.5°C 9.02 W 73.7°C 59.1°C
60.2°C 9.02 W 73.0°C 59.9°C 57.7°C
Highest Temperature: 76.0°C
(d) 10 active cores
7.52 W 71.6°C 7.52 W 72.2°C 7.52 W 71.4°C 59.5°C
7.52 W 72.5°C 7.52 W 73.2°C 7.52 W 72.6°C 7.52 W 70.3°C
7.52 W 72.3°C 7.52 W 72.9°C 7.52 W 71.4°C 59.6°C
7.52 W 71.3°C 7.52 W 71.2°C 59.8°C 57.9°C
Highest Temperature: 73.2°C
(e) 12 active cores
5.64 W 67.8°C 5.64 W 68.5°C 5.64 W 68.5°C 5.64 W 67.8°C
5.64 W 68.5°C 5.64 W 69.5°C 5.64 W 69.5°C 5.64 W 68.5°C
5.64 W 68.5°C 5.64 W 69.5°C 5.64 W 69.5°C 5.64 W 68.5°C
5.64 W 67.8°C 5.64 W 68.5°C 5.64 W 68.5°C 5.64 W 67.8°C
Highest Temperature: 69.5°C
(f) 16 active cores
100
90
80
70
60
50
[°C]

Figure 5.1: Temperature distribution example (with DTM deactivated) for using a 90.2 W per-chip power budget equally distributed among different number of simultaneously active cores. Top numbers are the power consumptions on each active core (boxed in black). Bottom numbers are the temperatures on the center of each core. Detailed temperatures are shown according to the color bar.

In order to show that similar effects also occur for other power budgets, Figure 5.2 presents simulations conducted in HotSpot that show the maximum steady-state temperature among all cores, as a function of the number of simultaneously active cores, for three per-chip power budgets and one per-core power budget, particularly, 58.7 W per-chip, 90.2 W per-chip, 129.0 W per-chip, and 8.06 W per-core. From Figure 5.2, it becomes clear that considering a *single* and *constant* per-chip or per-core power budget as an abstraction from thermal issues can be either a pessimistic approach (for small power budgets), or it can result in frequent thermal violations that would trigger DTM (for large power budgets).

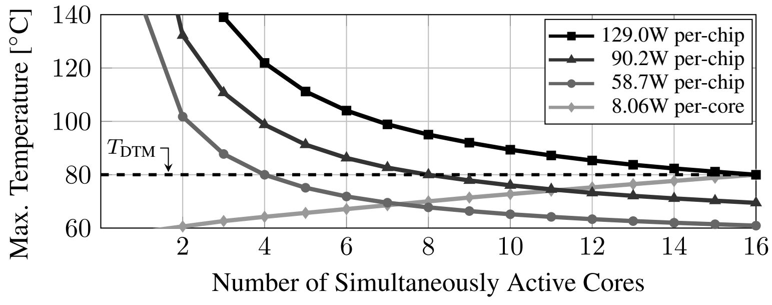
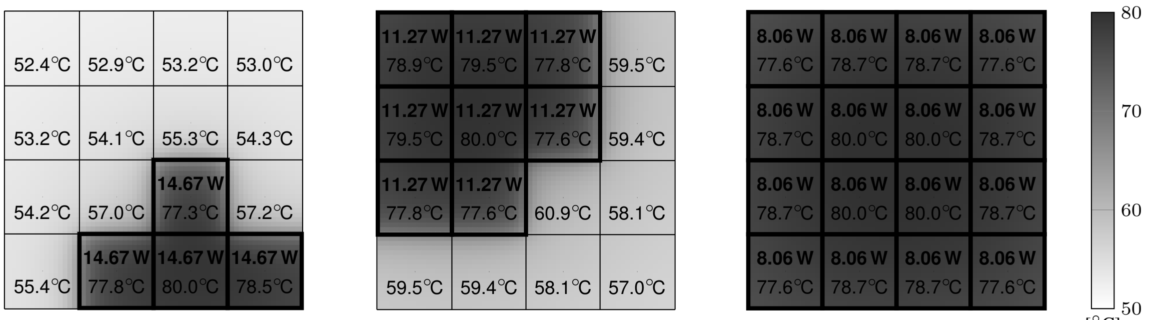


Figure 5.2: Maximum steady-state temperature (with DTM deactivated) among all cores as a function of the number of simultaneously active cores, when using different *single* and *constant* per-chip and per-core power budgets.

A more efficient power budgeting technique would consider different per-core power budgets according to the number of active cores, such that the maximum steady-state temperature among all cores is exactly 80°C for all cases, and *this is precisely the novel power budgeting approach which we call Thermal Safe Power (TSP).* For example, conducting simulations with HotSpot, Figure 5.3a illustrates the resulting steady-state temperatures on the chip when 4 cores consume 14.67 W each (58.7 W in total). Similarly, Figure 5.3b and Figure 5.3c illustrate the resulting steady-state temperatures on the chip when 8 cores consume 11.27 W each (90.2 W in total) and when 16 cores consume 8.06 W each (129.0 W in total), respectively. For all cases in Figure 5.3, although different per-core power budgets are considered, at least one core exactly reaches the critical temperature of 80°C in the steady-state.



(a) 4 active cores (b) 8 active cores (c) 16 active cores

Figure 5.3: Examples resulting in a maximum steady-state temperature of 80°C. Top numbers are the power consumptions on each active core (boxed in black). Bottom numbers are the temperatures on the center of each core. Detailed temperatures are shown according to the color bar.

5.1.2 Problem Definition

TSP is an abstraction that provides safe (but efficient) per-core power constraint values as a function of the number of simultaneously active cores. The values of TSP vary according to the floorplan, cooling solution, and according to which cores are simultaneously active. Some specific core mappings result in the lowest TSP values, and we define these core mappings as the *worst-case mappings*. For the corresponding mapping and number of active cores, executing cores at power consumption values that do not exceed TSP will result in maximum steady-state temperatures that do not exceed T_{DTM}.

For a specific chip and its corresponding RC thermal network, the *first objective* of this chapter is to derive a numerical method to compute TSP for a given mapping of cores. This given mapping of cores is typically determined by an operating/runtime system or by a design-time system software. This method should have polynomial-time complexity, such that TSP can be computed at runtime for a particular mapping of cores and ambient temperature. Formally, for the case with homogeneous cores, for a given core mapping $\mathbf{X}$, this means obtaining a uniform power constraint for the active cores, defined as $P_{\text{TSP}}(\mathbf{X})$, such that the steady-state temperature of all blocks in the floorplan does not exceed T_{DTM}, i.e., such that $T_{\text{steady}_i} \leq T_{\text{DTM}}$ for all $i \in \mathbf{L}$. Similarly, for the case with heterogeneous cores, also for a given core mapping $\mathbf{X}$ (independent of the type of core), this translates to obtaining a uniform power density constraint, defined as $P^{\rho}_{\text{TSP}}(\mathbf{X})$, and then multiplying $P^{\rho}_{\text{TSP}}(\mathbf{X})$ with the area of every active core.

The *second objective* of this chapter is to derive an algorithm to compute the most pessimistic TSP values for a given number of simultaneously active cores, i.e., for the worst-case mappings. Such TSP values can be used as safe power constraints for any possible mapping of cores, therefore allowing the system designers to abstract from core mapping decisions. Formally, for the case with homogeneous cores, this means obtaining the most pessimistic uniform power constraint for any m active cores, defined as $P^{\text{worst}}_{\text{TSP}}(m)$, such that $T_{\text{steady}_i} \leq T_{\text{DTM}}$ for all $i \in \mathbf{L}$. Furthermore, for the case with heterogeneous cores, given that different core types have different areas, the value of the power density constraint depends on the number of active cores for each type of core. Therefore, defining set $\mathbf{m} = \{m_1, m_2, \ldots, m_Q\}$ to represent the number of active cores for core types $\{1, 2, \ldots, Q\}$, respectively, this translates to obtaining the most pessimistic uniform power density constraint (independent of the type of core) for any $\mathbf{m} = \{m_1, m_2, \ldots, m_Q\}$ active cores, defined as $P_{\text{TSP}}^{\rho\text{ worst}}(\mathbf{m})$, and then multiplying $P_{\text{TSP}}^{\rho\text{ worst}}(\mathbf{m})$ with the area of every active core.

For simplicity of presentation and in order to consider safe margins, when computing TSP, we assume that all blocks in the floorplan that do not correspond to cores are always active and consuming their highest power consumption, such that column vector $\mathbf{P}^{\text{blocks}}$ is filled with the associated power consumption values.

The algorithms presented in Section 5.2 and Section 5.3 are derived considering the steady-state temperatures. Section 5.4 explains a method to further account for the transient thermal effects.

5.2 Thermal Safe Power for Homogeneous Systems

This section presents the algorithms necessary to compute TSP for homogeneous systems.

5.2.1 Given Core Mapping on Homogeneous Systems

This section presents a polynomial-time algorithm to compute TSP at runtime for a particular core mapping and ambient temperature, resulting in a uniform per-core value of TSP, for all active cores in mapping $\mathbf{X}$. That is, a per-core power budget value for each active core in the specified mapping, defined as $P_{\text{TSP}}(\mathbf{X})$, that results in a maximum steady-state temperature throughout the chip that does not exceed T_{DTM}.

Note that this does not imply that in practice all active cores are forced to consume the same power, as this would be an unrealistic assumption. Thus, every active core may consume any amount of power, which can be different for each core, as long as the power consumption on individual cores does not exceed $P_{\text{TSP}}(\mathbf{X})$.

Due to the heat transfer among cores, *which* cores are active and *which* cores are inactive (i.e., the specific core mapping), plays a major role in the computation of the maximum temperatures. This concept can be seen in the following example. Consider the 16-core homogeneous system presented in Figure 4.3a (described in Chapter 4.2.1). Figure 5.4 illustrates two possible mappings when simultaneously activating 6 cores: a concentrated mapping and a distributed mapping. For Figure 5.4a, the maximum steady-state temperature among all cores reaches 80°C when each active core consumes 12.74 W. Contrarily, for Figure 5.4b this happens when each active core consumes 14.64 W. Therefore, we have a total power difference of 11.4 W between these two core mappings; however, both cases result in the same maximum steady-state temperature.

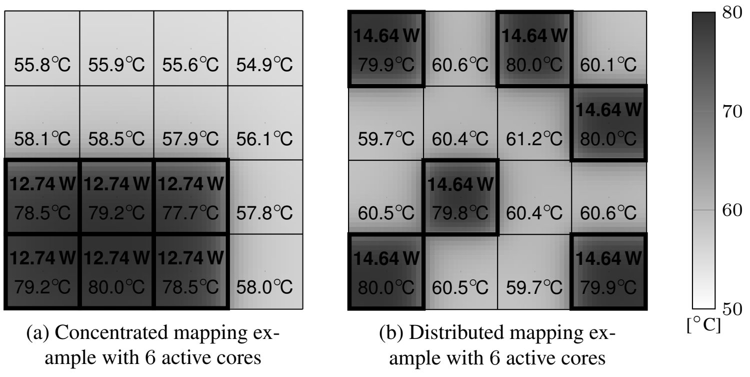


Figure 5.4: (from [75]) Example of TSP for two different mappings resulting in a maximum steady-state temperature of 80°C. Top numbers are the power consumptions on each active core (boxed in black). Bottom numbers are the temperatures on the center of each core. Detailed temperatures are shown according to the color bar.

For simplicity of presentation, we start by ignoring the maximum chip power constraint P_{max}. Therefore, by taking into account the thermal model of the chip, the power consumption on blocks of the floorplan that do not correspond to cores, the ambient temperature, and the power consumption of inactive cores, we derive a uniform per-core power budget for all active cores in mapping $\mathbf{X}$ that could potentially exceed P_{max}, such that the maximum steady-state temperature throughout the chip does not exceed T_{DTM}. This per-core power budget is defined as $P_{\text{TSP}}^{\star}(\mathbf{X})$.

For a given core mapping executing under per-core power budget $P_{\text{TSP}}^{\star}(\mathbf{X})$, the maximum temperatures would clearly occur if all active cores consume the entire power budget. Therefore, by considering vector

$\mathbf{X}$, set $\mathbf{K}$, and assuming that all inactive cores consume $P^{\text{core}}_{\text{inact}}$ and that all active cores consume equal power P_{equal} at a given point in time, we start by rewriting Equation (3.11) as shown in Equation (5.1).

$$T_{\text{steady}_i} = \sum_{j=1}^{N} \tilde{b}_{i,j} \cdot p^{\text{cores}}_j + \sum_{j=1}^{N} \tilde{b}_{i,j} \left(p^{\text{blocks}}_j + T_{\text{amb}} \cdot g_j \right) \tag{3.11 revisited}$$

$$T_{\text{steady}_i} = P_{\text{equal}} \cdot \sum_{j=1}^{M} \tilde{b}_{i,k_j} \cdot x_j + P^{\text{core}}_{\text{inact}} \cdot \sum_{j=1}^{M} \tilde{b}_{i,k_j} \left(1 - x_j\right) + \sum_{j=1}^{N} \tilde{b}_{i,j} \left(p^{\text{blocks}}_j + T_{\text{amb}} \cdot g_j \right) \tag{5.1}$$

In Equation (5.1), the value of P_{equal} is not yet $P^{\star}_{\text{TSP}}(\mathbf{X})$, but the expression represents the resulting steady-state temperature on thermal node i when all active cores in the given mapping consume equal power. Furthermore, for a given $\mathbf{X}$, $\mathbf{P}^{\text{blocks}}$, T_{amb}, and thermal model, the only variables in Equation (5.1) are T_{steady_i} and P_{equal}. As seen in Equation (5.1), that is,

$$T_{\text{steady}_i} = P_{\text{equal}} \cdot \underbrace{\sum_{j=1}^{M} \tilde{b}_{i,k_j} \cdot x_j + P^{\text{core}}_{\text{inact}} \cdot \sum_{j=1}^{M} \tilde{b}_{i,k_j} \left(1 - x_j\right) + \sum_{j=1}^{N} \tilde{b}_{i,j} \left(p^{\text{blocks}}_j + T_{\text{amb}} \cdot g_j \right)}_{\textbf{Constant for node } i} .$$

Given that the temperature which we do not wish to exceed is T_{DTM}, the goal is now to find the value of P_{equal} that would make T_{steady_i} reach the value of T_{DTM}. This can be easily done from Equation (5.1) by setting T_{steady_i} to T_{DTM} and then deriving the value of P_{equal} as shown in Equation (5.2).

$$P_{\text{equal}} = \frac{T_{\text{DTM}} - P^{\text{core}}_{\text{inact}} \cdot \sum_{j=1}^{M} \tilde{b}_{i,k_j} \left(1 - x_j\right) - \sum_{j=1}^{N} \tilde{b}_{i,j} \left(p^{\text{blocks}}_j + T_{\text{amb}} \cdot g_j \right)}{\sum_{j=1}^{M} \tilde{b}_{i,k_j} \cdot x_j} \tag{5.2}$$

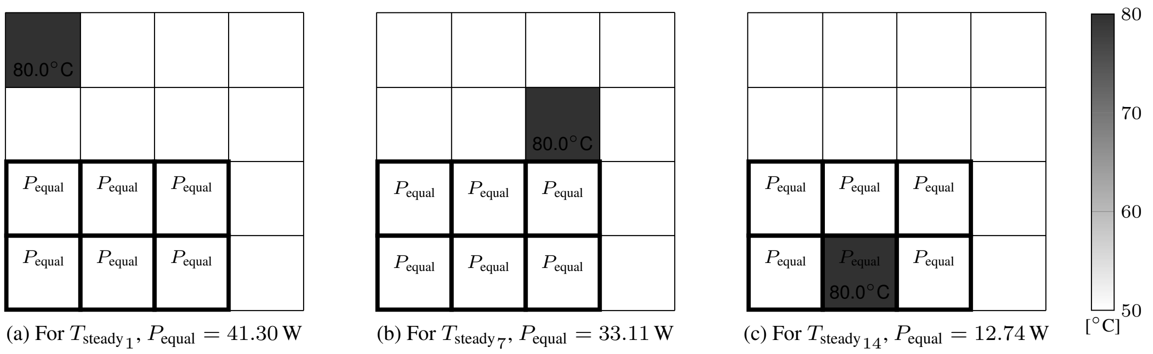
The most pessimistic value of P_{equal} is a safe per-core power budget for all cores in mapping $\mathbf{X}$. Therefore, the resulting $P^{\star}_{\text{TSP}}(\mathbf{X})$ for the given $\mathbf{X}$, $\mathbf{P}^{\text{blocks}}$, T_{amb}, T_{DTM}, $P^{\text{core}}_{\text{inact}}$, and thermal model, can be computed by finding the minimum value of P_{equal} for all blocks in the floorplan, i.e., $\forall i \in \mathbf{L}$. The computation of $P^{\star}_{\text{TSP}}(\mathbf{X})$ is presented in Equation (5.3). The value of i that results in $P^{\star}_{\text{TSP}}(\mathbf{X})$ corresponds to the block with the highest temperature for such a case.

$$P^{\star}_{\text{TSP}}(\mathbf{X}) = \min_{\forall i \in \mathbf{L}} \left\{ \frac{T_{\text{DTM}} - P^{\text{core}}_{\text{inact}} \cdot \sum_{j=1}^{M} \tilde{b}_{i,k_j} \left(1 - x_j\right) - \sum_{j=1}^{N} \tilde{b}_{i,j} \left(p^{\text{blocks}}_j + T_{\text{amb}} \cdot g_j \right)}{\sum_{j=1}^{M} \tilde{b}_{i,k_j} \cdot x_j} \right\} \tag{5.3}$$

Considering the 16-core homogeneous system presented in Figure 4.3a (described in Chapter 4.2.1), Figure 5.5 presents a brief example of how $P^{\star}_{\text{TSP}}(\mathbf{X})$ is computed using Equation (5.3) for Figure 5.4a. Namely, Figure 5.5a shows the power that should be consumed by the 6 active cores (i.e., P_{equal}), such that T_{steady_1} reaches 80°C, by momentarily ignoring the temperature on the other blocks of the floorplan. Figure 5.5b and Figure 5.5c present the same thing when focusing on T_{steady_7} and $T_{\text{steady}_{14}}$, respectively. The computation of P_{equal} for the other cores is not shown in the figure; however, the same principle applies. After computing P_{equal} for all T_{steady_i}, particularly $T_{\text{steady}_1}, T_{\text{steady}_2}, \ldots, T_{\text{steady}_{16}}$ for this example, $P^{\star}_{\text{TSP}}(\mathbf{X})$ is set to the smallest value of P_{equal}, which for this example was the case when computing $T_{\text{steady}_{14}}$. Furthermore, given that the final value of $P^{\star}_{\text{TSP}}(\mathbf{X})$ is found when computing $T_{\text{steady}_{14}}$, it holds that this block will have the highest steady-state temperature in the entire chip when all active cores consume equal power, as shown in Figure 5.4a.

Now that we have $P^{\star}_{\text{TSP}}(\mathbf{X})$, we can include P_{max} back into the formulation. Basically, if all active cores in mapping $\mathbf{X}$ consume the entire per-core power budget, considering the power consumption of the inactive cores and other blocks in the floorplan, the total power consumption in the chip should not exceed the value of P_{max}. Therefore, since the summation of the elements in vector $\mathbf{X}$ is equal to the number of active cores in the mapping, it should hold that

$$P_{\text{TSP}}(\mathbf{X}) \cdot \sum_{i=1}^{M} x_i + P^{\text{core}}_{\text{inact}} \left(M - \sum_{i=1}^{M} x_i \right) + \sum_{i=1}^{N} p^{\text{blocks}}_i \leq P_{\text{max}}.$$



(a) For T_{steady_1}, $P_{\text{equal}} = 41.30\,\text{W}$ (b) For T_{steady_7}, $P_{\text{equal}} = 33.11\,\text{W}$ (c) For $T_{\text{steady}_{14}}$, $P_{\text{equal}} = 12.74\,\text{W}$

Figure 5.5: (from [75]) TSP computation example for the given mapping from Figure 5.4a.

From this equation, it is simple to derive the amount of power that any m active cores can consume, such that the total power consumption precisely reaches the value of P_{max}. For such a purpose, we define auxiliary function $P^{\text{core}}_{\text{max}}(m)$, shown in Equation (5.4), where m represents the number of active cores.

$$P^{\text{core}}_{\text{max}}(m) = P^{\text{core}}_{\text{inact}} + \frac{P_{\text{max}} - \sum_{i=1}^{N} p_i^{\text{blocks}} - P^{\text{core}}_{\text{inact}} \cdot M}{m} \tag{5.4}$$

Therefore, if we have that $P^{\star}_{\text{TSP}}(\mathbf{X}) \le P^{\text{core}}_{\text{max}}\left(\sum_{i=1}^{M} x_i\right)$, it means that consuming $P^{\star}_{\text{TSP}}(\mathbf{X})$ in all active cores will not exceed P_{max}, and thus we can safely set $P_{\text{TSP}}(\mathbf{X})$ to $P^{\star}_{\text{TSP}}(\mathbf{X})$. Contrarily, in case that $P^{\star}_{\text{TSP}}(\mathbf{X}) > P^{\text{core}}_{\text{max}}\left(\sum_{i=1}^{M} x_i\right)$, then consuming $P^{\star}_{\text{TSP}}(\mathbf{X})$ in all active cores would exceed P_{max}. Hence, instead of setting the value of $P_{\text{TSP}}(\mathbf{X})$ to $P^{\star}_{\text{TSP}}(\mathbf{X})$, in this case we simply set it to a smaller value, specifically, to $P^{\text{core}}_{\text{max}}\left(\sum_{i=1}^{M} x_i\right)$, such that P_{max} is satisfied. Formally, the computation of power budget $P_{\text{TSP}}(\mathbf{X})$ for each active core in the specified mapping $\mathbf{X}$ is shown in Equation (5.5). The total time complexity for computing $P_{\text{TSP}}(\mathbf{X})$ for a given $\mathbf{P}^{\text{blocks}}$, T_{amb}, T_{DTM}, P_{max}, and thermal model of the chip, is $O(ZN)$.

$$P_{\text{TSP}}(\mathbf{X}) = \begin{cases} P^{\star}_{\text{TSP}}(\mathbf{X}) & \text{if } P^{\star}_{\text{TSP}}(\mathbf{X}) \le P^{\text{core}}_{\text{max}}\left(\sum_{i=1}^{M} x_i\right) \\ P^{\text{core}}_{\text{max}}\left(\sum_{i=1}^{M} x_i\right) & \text{otherwise} \end{cases} \tag{5.5}$$

5.2.2 Worst-Case Mappings on Homogeneous Systems

This section presents a polynomial-time algorithm to compute TSP for the worst-case mappings with m active cores, resulting in a uniform per-core value of TSP for all m active cores. That is, a per-core power budget value for each active core in *any* possible mapping with m simultaneously active cores, defined as $P^{\text{worst}}_{\text{TSP}}(m)$, that results in a maximum steady-state temperature throughout the chip that does not exceed T_{DTM}.

As mentioned in Section 5.2.1, note that this does not imply that in practice all active cores are forced to consume the same power, as this would be an unrealistic assumption. It means that every active core, *for any possible mapping with m active cores*, may consume any amount of power, which can be different for each core, as long as the power consumption on individual cores does not exceed $P^{\text{worst}}_{\text{TSP}}(m)$. The purpose behind this, is to allow system designers and resource management techniques to abstract themselves from mapping decisions when doing task partitioning and selecting the DVFS levels of cores, thus reducing the complexity of the management algorithms.

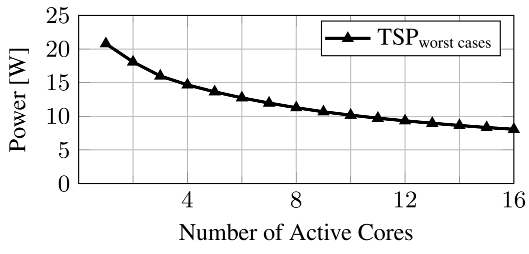
As already discussed in Section 5.2.1 and shown in Figure 5.4, due to the heat transfer among cores, the mapping of cores will play a major role in the resulting maximum temperatures. According to Equation (5.1), if all active cores in the system consume equal power P_{equal} at a given point in time, there will be one (or more) active core (or cores) that will heat up more than the other cores. Moreover, for the same value of P_{equal}, different $\mathbf{X}$ mappings will result in different maximum temperatures. Dually, for the same maximum temperature among all cores, different mappings will result in different power consumption values to achieve such a temperature. This duality is what allows us to compute $P_{\text{TSP}}(\mathbf{X})$ through Equation (5.5) for a given $\mathbf{X}$.

Generally, as already shown in Figure 5.4, for the same maximum steady-state temperatures with $p_i^{\text{blocks}} = 0$ for all i, having a group of cores active together in a corner of the chip results in lower P_{equal} values when compared to dispersing the active cores throughout the chip. *This happens because when active cores are dispersed, the* active *cores have a high chance of transferring heat to the* inactive *cores.* The worst-case mappings for TSP are those that result in the lowest power budgets, while no block in the floorplan exceeds (in the steady-state) the critical threshold temperature that triggers DTM, i.e., $T_{\text{steady}_i} \leq T_{\text{DTM}}$ for all $i \in \mathbf{L}$.

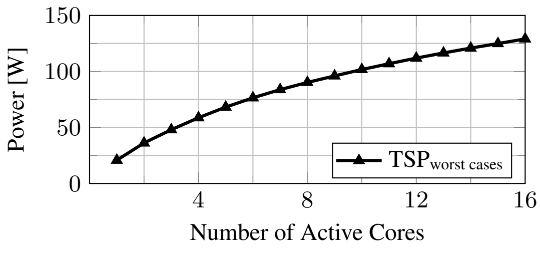
By computing TSP for the worst-case mappings, given that they are the most pessimistic cases, system designers and resource management techniques can abstract themselves from core mapping decisions. When having m active cores, executing cores at power consumption levels that do not exceed $P_{\text{TSP}}^{\text{worst}}(m)$ will result in maximum steady-state temperatures, among all blocks in the floorplan, that do not exceed the critical threshold temperature that triggers DTM, for any possible mapping with m active cores.

For example, considering the 16-core homogeneous system presented in Figure 4.3a (described in Chapter 4.2.1), Figure 5.6 illustrates the values of TSP for the worst-case core mappings when having and ambient temperature of $45°\text{C}$ and $m = 1, 2, \ldots, 16$ active cores. The figure shows that TSP is a per-core power budget that results in a decreasing function with respect to the number of simultaneously active cores. For presentation purposes, by multiplying the TSP values with the number of active cores for each case, Figure 5.6 also presents the resulting power consumptions at a chip level when all active cores are consuming their entire per-core power budgets, resulting in a non-decreasing function with respect to the number of active cores.

Active Cores	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
TSP worst cases (per-core) [W]	20.79	18.08	16.00	14.67	13.64	12.74	11.97	11.27	10.67	10.17	9.72	9.33	8.96	8.63	8.33	8.06



(a) Per-core budget



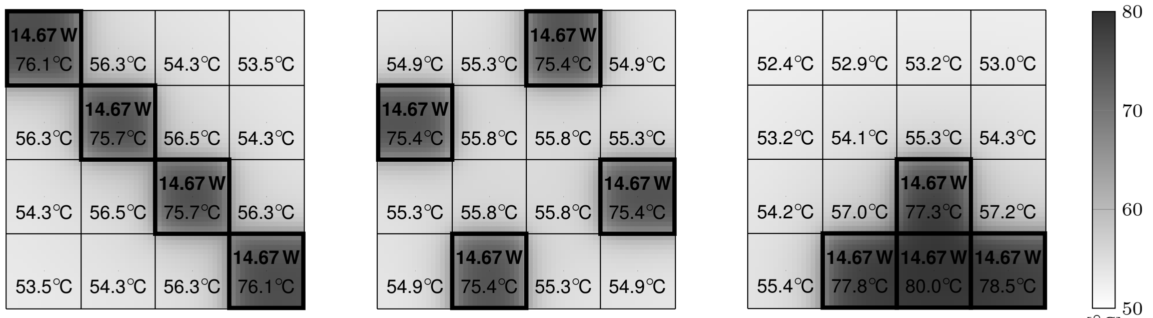
(b) Estimated per-chip budget

Figure 5.6: Example of TSP results for the worst-case core mappings.

To reinforce the concept that using TSP as a per-core power budget for the worst-case mappings with m active cores is safe for *any* possible core mapping with m simultaneously active cores, considering the 16-core homogeneous system presented in Figure 4.3a (described in Chapter 4.2.1), Figure 5.7 presents an example for three different core mappings with 4 active cores executing under the computed worst-case TSP of 14.67 W. The figure shows that distributed mappings result in a larger thermal headroom than concentrated mappings, but more importantly, it shows that in no case does the steady-state temperature anywhere on the chip exceeded the value of T_{DTM} while the worst-case TSP power budget is satisfied.

One possible approach for computing TSP for the worst-case mappings could consist on first finding one such worst-case mapping, and then computing the TSP value for the mapping by using Equation (5.5). However, it would be more efficient if we can derive a method to directly compute $P_{\text{TSP}}^{\text{worst}}(m)$ without troubling on finding a worst-case mapping first.

For a given thermal node i, we know from Equation (5.1) that there is one or more core mappings $\mathbf{X}$ that result in the maximum T_{steady_i} for a given value of P_{equal}. Particularly, we can distinguish which part of



(a) Highest Temperature: $76.1°$C (b) Highest Temperature: $75.4°$C (c) Highest Temperature: $80.0°$C

Figure 5.7: Example of different core mappings executing under TSP for the worst-case mappings with 4 active cores. Top numbers are the power consumptions on each active core (boxed in black). Bottom numbers are the temperatures on the center of each core. Detailed temperatures are shown according to the color bar.

Equation (5.1) is constant for a given thermal node i and which part depends on the mapping of cores, as

$$T_{\text{steady}_i} = P_{\text{equal}} \cdot \underbrace{\sum_{j=1}^{M} \tilde{b}_{i,k_j} \cdot x_j + P^{\text{core}}_{\text{inact}} \cdot \sum_{j=1}^{M} \tilde{b}_{i,k_j} \left(1 - x_j\right)}_{\textbf{Depends on the mapping}} + \underbrace{\sum_{j=1}^{N} \tilde{b}_{i,j} \left(p^{\text{blocks}}_j + T_{\text{amb}} \cdot g_j\right)}_{\textbf{Constant for node } i} .$$

Namely, from Equation (5.1), given that $\mathbf{B}^{-1}$, $\mathbf{G}$, $\mathbf{P}^{\text{blocks}}$, and T_{amb} are constant, for a given node i and value of P_{equal}, the value of T_{steady_i} is maximized when $P_{\text{equal}} \cdot \sum_{j=1}^{M} \tilde{b}_{i,k_j} \cdot x_j + P^{\text{core}}_{\text{inact}} \cdot \sum_{j=1}^{M} \tilde{b}_{i,k_j} \left(1 - x_j\right)$ is maximized. Moreover, assuming that an inactive core will always consume less power than an active core (even if the active core executes at its minimum DVFS levels), it holds that $P_{\text{equal}} \geq P^{\text{core}}_{\text{inact}}$. Hence, given that x_j and $(1 - x_j)$ are mutually exclusive, for a given node i and P_{equal}, then the value of T_{steady_i} is maximized when $\sum_{j=1}^{M} \tilde{b}_{i,k_j} \cdot x_j$ is maximized. For row i, this happens when mapping $\mathbf{X}$ activates the m cores with the highest $\tilde{b}_{i,k_j}$ values.

Thus, if all active cores consume equal power P_{equal}, the m highest values for $P_{\text{equal}} \cdot \tilde{b}_{i,j}$ such that $j \in \mathbf{K}$, correspond to the maximum amount of heat than any m cores can contribute to temperature T_{steady_i}.

In order to compute the maximum amount of heat that *any* m cores can contribute to the temperature on node i, we define auxiliary matrix $\mathbf{H} = [h_{i,j}]_{Z \times M}$. Matrix $\mathbf{H}$ is built by making a partial copy of matrix $\mathbf{B}^{-1}$, such that $h_{i,j} = \tilde{b}_{\ell_i,k_j}$ for all $i = 1, 2, \ldots, Z$ and for all $j = 1, 2, \ldots, M$, and then reordering every row of $\mathbf{H}$ decreasingly. For a given chip and cooling solution (i.e., for a given RC thermal network), matrix $\mathbf{H}$ only needs to be build and ordered one time, which has time complexity $O\left(ZM \log M\right)$. The pseudo-code to build matrix $\mathbf{H}$ is presented in Algorithm 1.

Algorithm 1 Computation of auxiliary matrix $\mathbf{H}$

```
Input: Matrix $\mathbf{B}^{-1} = [\tilde{b}_{i,j}]_{N \times N}$, set $\mathbf{L}$, and set $\mathbf{K}$;
Output: Auxiliary matrix $\mathbf{H} = [h_{i,j}]_{Z \times M}$;
1: for all $i = 1, 2, \ldots, Z$ (i.e., for all blocks in the floorplan) do
2:     for all $j = 1, 2, \ldots, M$ (i.e., for all cores in the chip) do
3:         $h_{i,j} \leftarrow \tilde{b}_{\ell_i,k_j}$; {Build partial copy of matrix $\mathbf{B}^{-1}$}
4:     end for
5:     Re-order row $i$ of matrix $\mathbf{H}$ decreasingly;
6: end for
7: return Auxiliary matrix $\mathbf{H}$;
```

Based on the definition of auxiliary matrix $\mathbf{H}$, for thermal node ℓ_i, the first m elements in row i of matrix $\mathbf{H}$ correspond to the m highest $\tilde{b}_{\ell_i,j}$ values, such that $j \in \mathbf{K}$. Therefore, if all active cores consume equal power P_{equal}, multiplying the power consumption on each core with the summation of the first m elements in row i of auxiliary matrix $\mathbf{H}$, i.e., computing $P_{\text{equal}} \cdot \sum_{j=1}^{m} h_{i,j}$, will result in the maximum amount of heat that any m cores can contribute to the steady-state temperature on node ℓ_i, i.e., $T_{\text{steady}_{\ell_i}}$.

For simplicity of presentation and similarly to Section 5.2.1, we first start by ignoring the maximum chip power constraint P_{max}. Therefore, by taking into account the thermal model of the chip, the power consumption on blocks of the floorplan that do not correspond to cores, the ambient temperature, and the power consumption of inactive cores, we derive the worst-case uniform per-core power budget for *any* possible mapping of cores with m simultaneously active cores that could potentially exceed P_{max}, such that the maximum steady-state temperature throughout the chip does not exceed T_{DTM}. This per-core power budget is defined as $P_{\text{TSP}}^{\star\text{worst}}(m)$. In order to derive $P_{\text{TSP}}^{\star\text{worst}}(m)$, we first rewrite Equation (5.1) considering matrix $\mathbf{H}$ as

$$T_{\text{steady}_{\ell_i}} \leq P_{\text{equal}} \cdot \sum_{j=1}^{m} h_{i,j} + P_{\text{inact}}^{\text{core}} \cdot \sum_{j=m+1}^{M} h_{i,j} + \sum_{j=1}^{N} \tilde{b}_{\ell_i,j} \left(p_j^{\text{blocks}} + T_{\text{amb}} \cdot g_j \right).$$

Similar to Section 5.2.1, by setting $T_{\text{steady}_{\ell_i}}$ to be equal to T_{DTM} for a given i, we can compute the value of P_{equal} that would make $T_{\text{steady}_{\ell_i}}$ reach the value T_{DTM} as

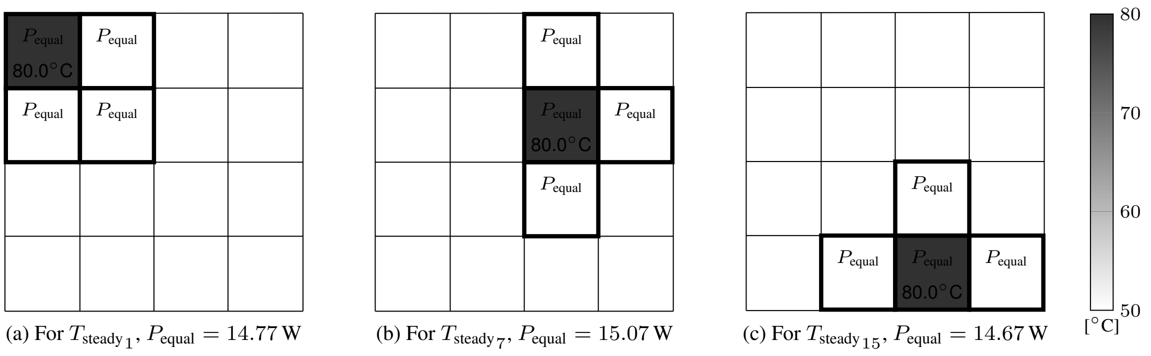
$$P_{\text{equal}} = \frac{T_{\text{DTM}} - P_{\text{inact}}^{\text{core}} \cdot \sum_{j=m+1}^{M} h_{i,j} - \sum_{j=1}^{N} \tilde{b}_{\ell_i,j} \left(p_j^{\text{blocks}} + T_{\text{amb}} \cdot g_j \right)}{\sum_{j=1}^{m} h_{i,j}}.$$

Given that the most pessimistic value of P_{equal} is a safe power budget for any m active cores, for the given $\mathbf{P}^{\text{blocks}}$, T_{amb}, T_{DTM}, $P_{\text{inact}}^{\text{core}}$, and thermal model, the value of $P_{\text{TSP}}^{\star\text{worst}}(m)$ can be computed by finding the minimum P_{equal} for all blocks in the floorplan, i.e., for every $i = 1, 2, \ldots, Z$. The computation of $P_{\text{TSP}}^{\star\text{worst}}(m)$ is presented in Equation (5.6), and the value of i that results in $P_{\text{TSP}}^{\star\text{worst}}(m)$ corresponds to the block with the highest temperature in the worst-case mapping with m active cores.

$$P_{\text{TSP}}^{\star\text{worst}}(m) = \min_{1 \leq i \leq Z} \left\{ \frac{T_{\text{DTM}} - P_{\text{inact}}^{\text{core}} \cdot \sum_{j=m+1}^{M} h_{i,j} - \sum_{j=1}^{N} \tilde{b}_{\ell_i,j} \left(p_j^{\text{blocks}} + T_{\text{amb}} \cdot g_j \right)}{\sum_{j=1}^{m} h_{i,j}} \right\} \tag{5.6}$$

Similar to Section 5.2.1, considering the 16-core homogeneous system presented in Figure 4.3a (described in Chapter 4.2.1), Figure 5.8 presents a brief example of how $P_{\text{TSP}}^{\star\text{worst}}(m)$ is computed using Equation (5.6) for the worst-case mappings with 4 active cores. Namely, Figure 5.8a shows the power that should be consumed by the 4 cores that contribute the most heat to T_{steady_1}, such that T_{steady_1} reaches 80°C, momentarily ignoring the temperature on the other blocks of the floorplan. Figure 5.8b and Figure 5.8c present the same thing when focusing on T_{steady_7} and $T_{\text{steady}_{15}}$, respectively. The computation of P_{equal} for the other cores is not shown in the figure; however, the same principle applies. It is important to note that for each T_{steady_i}, there is a different set of m active cores that contributes more heat into temperature T_{steady_i}. After computing P_{equal} for all T_{steady_i}, particularly $T_{\text{steady}_1}, T_{\text{steady}_2}, \ldots, T_{\text{steady}_{16}}$ for this example, $P_{\text{TSP}}^{\star\text{worst}}(m)$ is set to the smallest value of P_{equal}, which for this example was the case when computing $T_{\text{steady}_{15}}$. Furthermore, given that the final value of $P_{\text{TSP}}^{\star\text{worst}}(m)$ is found when computing $T_{\text{steady}_{15}}$, it holds that this block will have the highest steady-state temperature in the chip when all active cores consume equal power under the worst-case mapping, as shown in Figure 5.7c. Naturally, for symmetrical chips there exist more than one worst-case mapping for m active cores; however, computing TSP for one such mapping is sufficient to derive a safe per-core power budget for *any* possible mapping with m active cores.

Similar to Section 5.2.1, now that we have $P_{\text{TSP}}^{\star\text{worst}}(m)$, we can include P_{max} back into the formulation. The logic and procedure is equivalent as that in Section 5.2.1. Namely, if we have that $P_{\text{TSP}}^{\star\text{worst}}(m) \leq P_{\text{max}}^{\text{core}}(m)$, it means that simultaneously consuming $P_{\text{TSP}}^{\star\text{worst}}(m)$ in m active cores will not exceed P_{max}, and thus we can safely set $P_{\text{TSP}}^{\text{worst}}(m)$ to $P_{\text{TSP}}^{\star\text{worst}}(m)$. Contrarily, in case $P_{\text{TSP}}^{\star\text{worst}}(m) > P_{\text{max}}^{\text{core}}(m)$, then simultaneously consuming $P_{\text{TSP}}^{\star\text{worst}}(m)$ in m active cores would exceed P_{max}. Hence, instead of setting the value of $P_{\text{TSP}}^{\text{worst}}(m)$ to $P_{\text{TSP}}^{\star\text{worst}}(m)$, in this case we simply set it to a smaller value, specifically, to $P_{\text{max}}^{\text{core}}(m)$, such that P_{max} is satisfied. Formally, the computation of power constraint $P_{\text{TSP}}^{\text{worst}}(m)$ for each active core in *any* possible core



(a) For T_{steady_1}, $P_{\text{equal}} = 14.77\,\text{W}$ (b) For T_{steady_7}, $P_{\text{equal}} = 15.07\,\text{W}$ (c) For $T_{\text{steady}_{15}}$, $P_{\text{equal}} = 14.67\,\text{W}$

Figure 5.8: (from [75]) TSP computation example for the worst-case mappings with 4 active cores.

mapping with m simultaneously active cores is shown in Equation (5.7). Given that matrix $\mathbf{H}$ only needs to be build once for a given chip by using Algorithm 1, the total time complexity for computing $P_{\text{TSP}}^{\text{worst}}(m)$ for a given m is $O(ZN)$, and for computing $P_{\text{TSP}}^{\text{worst}}(m)$ for all $m = 1, 2, \ldots, M$ is $O(MZN)$.

$$P_{\text{TSP}}^{\text{worst}}(m) = \begin{cases} P_{\text{TSP}}^{\star\text{worst}}(m) & \text{if } P_{\text{TSP}}^{\star\text{worst}}(m) \leq P_{\max}^{\text{core}}(m) \\ P_{\max}^{\text{core}}(m) & \text{otherwise} \end{cases} \tag{5.7}$$

5.3 Thermal Safe Power for Heterogeneous Systems

Given that in heterogeneous systems cores have different areas and consume different amounts of power, a power budgeting technique should not use the same per-core power constraint for different types of cores. The two algorithms in Section 5.2 provide the foundations of TSP for homogeneous systems. In this section, we extend these concepts for the general case of heterogeneous systems.

5.3.1 Given Core Mapping on Heterogeneous Systems

In this section we extend Equation (5.5) from Section 5.2.1 in order to consider heterogeneous systems, like the one illustrated in Figure 4.4 (described in Chapter 4.2.2). To start, *it is important to note that the temperature on a chip is directly related to how* power density *is distributed throughout the chip, and only indirectly related to the power consumption.* For example, for two cores with different areas, if both cores have the same power consumption, the core with a small area will have a higher temperature than the core with a big area. Therefore, core heterogeneity should be handled by focusing on power density instead of power consumption, thus deriving a *power density budget*. This could be potentially done by dividing the cores of a heterogeneous floorplan into smaller homogeneous sub-blocks of the same size, and then computing TSP through Equation (5.5) for these smaller sub-blocks. However, such an approach would not be computationally efficient, especially when considering the required size of the sub-blocks (and corresponding RC thermal network) to perfectly fit all types of cores. A more efficient method is thus to directly compute the power density budgets. Therefore, in this section we derive a uniform *power density* budget for every active core in the specified mapping (independent of the type of core), defined as $P_{\text{TSP}}^{\rho}(\mathbf{X})$, that results in a maximum steady-state temperature throughout the chip that does not exceed T_{DTM}. To compute the corresponding TSP value for core j, we simply multiply the power density budget $P_{\text{TSP}}^{\rho}(\mathbf{X})$ with the area of core j.

As already done in Section 5.2 for simplicity of presentation, we first start by ignoring the maximum chip power constraint P_{max}. Therefore, by taking into account the thermal model of the chip, the power consumption on blocks of the floorplan that do not correspond to cores, the ambient temperature, and the power consumption of inactive cores, we derive a uniform power density budget (independent of the types of cores) for all active cores in mapping $\mathbf{X}$ that could potentially exceed P_{max}, such that the maximum steady-state temperature throughout the chip does not exceed T_{DTM}. This power density budget is defined as $P^{\rho\star}_{\text{TSP}}(\mathbf{X})$. Considering the above and assuming that all active cores have an equal power density P^{ρ}_{equal} at a given point in time, Equation (3.11) can be rewritten as shown in Equation (5.8).

$$T_{\text{steady}_i} = P^{\rho}_{\text{equal}} \cdot \sum_{j=1}^{M} \tilde{b}_{i,k_j} \cdot \text{area}^{\text{core}}_j \cdot x_j + \sum_{j=1}^{M} \tilde{b}_{i,k_j} \cdot P^{\text{core}}_{\text{inact}j} (1 - x_j) + \sum_{j=1}^{N} \tilde{b}_{i,j} \left(p^{\text{blocks}}_j + T_{\text{amb}} \cdot g_j \right) \tag{5.8}$$

From this expression and by following a similar procedure as in Section 5.2.1, we can compute $P^{\rho\star}_{\text{TSP}}(\mathbf{X})$ as presented in Equation (5.9), with total time complexity $O(ZM)$. The value of i that results in $P^{\rho\star}_{\text{TSP}}(\mathbf{X})$ corresponds to the block with the highest temperature for such a case.

$$P^{\rho\star}_{\text{TSP}}(\mathbf{X}) = \min_{\forall i \in \mathbf{L}} \left\{ \frac{T_{\text{DTM}} - \sum_{j=1}^{M} \tilde{b}_{i,k_j} \cdot P^{\text{core}}_{\text{inact}j} (1 - x_j) - \sum_{j=1}^{N} \tilde{b}_{i,j} \left(p^{\text{blocks}}_j + T_{\text{amb}} \cdot g_j \right)}{\sum_{j=1}^{M} \tilde{b}_{i,k_j} \cdot \text{area}^{\text{core}}_j \cdot x_j} \right\} \tag{5.9}$$

Once again, now that we have $P^{\rho\star}_{\text{TSP}}(\mathbf{X})$ we can include $P_{\max}$ back into the formulation. Basically, if the power density in all active cores in mapping $\mathbf{X}$ is equal to the entire power density budget, considering the power consumption of the inactive cores and other blocks in the floorplan, the total power consumption should not exceed the value of $P_{\max}$, i.e., it should hold that

$$P^{\rho}_{\text{TSP}}(\mathbf{X}) \cdot \sum_{j=1}^{M} \text{area}^{\text{core}}_j \cdot q_j + \sum_{j=1}^{M} P^{\text{core}}_{\text{inact}j} (1 - x_j) + \sum_{i=1}^{N} p^{\text{blocks}}_i \leq P_{\max}.$$

From this equation, it is simple to derive the amount of power density that any active cores in mapping $\mathbf{X}$ can have, such that the total power consumption precisely reaches the value of $P_{\max}$. For such a purpose, we define auxiliary function $P^{\text{core}\,\rho}_{\max}(\mathbf{X})$, shown in Equation (5.10), used to compute this power density value.

$$P^{\text{core}\,\rho}_{\max}(\mathbf{X}) = \frac{P_{\max} - \sum_{i=1}^{N} p^{\text{blocks}}_i - \sum_{j=1}^{M} P^{\text{core}}_{\text{inact}j} (1 - x_j)}{\sum_{j=1}^{M} \text{area}^{\text{core}}_j \cdot x_j} \tag{5.10}$$

Similar to Section 5.2, we then have that $P^{\rho}_{\text{TSP}}(\mathbf{X})$ can be computed as presented in Equation (5.11). The total time complexity for computing $P^{\rho}_{\text{TSP}}(\mathbf{X})$ for a given $\mathbf{P}^{\text{blocks}}$, T_{amb}, T_{DTM}, $P_{\max}$, and thermal model of the chip, is $O(ZM)$. To compute the corresponding TSP value for core j, we simply multiply $P^{\rho}_{\text{TSP}}(\mathbf{X})$ with the area of core j.

$$P^{\rho}_{\text{TSP}}(\mathbf{X}) = \begin{cases} P^{\rho\star}_{\text{TSP}}(\mathbf{X}) & \text{if } P^{\rho\star}_{\text{TSP}}(\mathbf{X}) \leq P^{\text{core}\,\rho}_{\max}(\mathbf{X}) \\ P^{\text{core}\,\rho}_{\max}(\mathbf{X}) & \text{otherwise} \end{cases} \tag{5.11}$$

5.3.2 Worst-Case Mappings on Heterogeneous Systems

This section extends Equation (5.7) presented in Section 5.2.2 in order to consider heterogeneous systems. Similar to Section 5.3.1, we focus on deriving a safe *power density* budget, rather than focus on power consumption. Furthermore, in Section 5.2.2 the value of the per-core power budget for the worst-case core mappings depends on the number of active cores. However, as mentioned in Section 5.1.2, for the case with heterogeneous systems, given that different core types have different areas, the value of the power density budget will depend on the *number of active cores for each type of core*. Therefore, in this section we derive a uniform *power density* budget for the worst-case core mappings with $\mathbf{m} = \{m_1, m_2, \ldots, m_Q\}$ active cores, and we define such a power density budget as $P^{\rho\,\text{worst}}_{\text{TSP}}(\mathbf{m})$. For example, if we have a system with three types of cores and we activate 4 cores of type 1 and 7 cores of type 3, the power density budget for such a case is denoted as $P^{\rho\,\text{worst}}_{\text{TSP}}(\{4, 0, 7\})$. We then simply multiply the power density budget with the area of each core, thus obtaining a per-core power budget for each type of core for *any* possible core mapping with $\mathbf{m} = \{m_1, m_2, \ldots, m_Q\}$ simultaneously active cores, that results in a maximum steady-state temperature throughout the chip which does not exceed T_{DTM}.

Similar to Section 5.2.2, we define auxiliary matrix $\mathbf{H}^\rho = \left[h^\rho_{q,i,j}\right]_{Q \times Z \times M^{\text{type}}_q}$ which is used to compute the maximum amount of heat that any m_q cores of type q can contribute to the temperature on node i, for all core types $q = 1, 2, \ldots, Q$. Particularly, matrix $\mathbf{H}^\rho$ is built by making a partial copy of matrix $\mathbf{B}^{-1}$, considering the type and the area of the cores, such that $h^\rho_{q,i,j} = \tilde{b}_{\ell_i, k^q_j} \cdot \text{area}^{\text{type}}_q$ for all $q = 1, 2, \ldots, Q$, for all $i = 1, 2, \ldots, Z$, and for all $j = 1, 2, \ldots, M^{\text{type}}_q$, and then, for every q and every i, reordering (with respect to j) each row of $\mathbf{H}^\rho$ decreasingly. For a given chip and cooling solution (i.e., for a given RC thermal network), matrix $\mathbf{H}^\rho$ only needs to be built and ordered one time, which has time complexity $O\left(Z M^{\text{type}}_q \log M^{\text{type}}_q\right)$ for every type of core q. The pseudo-code to build matrix $\mathbf{H}^\rho$ is presented in Algorithm 2.

Algorithm 2 Computation of auxiliary matrix $\mathbf{H}^\rho$

```
Input: Matrix $\mathbf{B}^{-1} = [\tilde{b}_{i,j}]_{N \times N}$, set $\mathbf{L}$, number of cores $M^{\text{type}}_q$, and sets $\mathbf{K}^q$ for $q = 1, 2, \ldots, Q$;
Output: Auxiliary matrix $\mathbf{H}^\rho = \left[h^\rho_{q,i,j}\right]_{Q \times Z \times M^{\text{type}}_q}$;
1: for all $q = 1, 2, \ldots, Q$ (i.e., for all types of cores) do
2:     for all $i = 1, 2, \ldots, Z$ (i.e., for all blocks in the floorplan) do
3:         for all $j = 1, 2, \ldots, M^{\text{type}}_q$ (i.e., for all cores of type $q$) do
4:             $h^\rho_{q,i,j} \leftarrow \tilde{b}_{\ell_i, k^q_j} \cdot \text{area}^{\text{type}}_q$; {Build partial copy of matrix $\mathbf{B}^{-1}$}
5:         end for
6:         For these $q$ and $i$ values, reorder $\mathbf{H}^\rho$ decreasingly with respect to $j$;
7:     end for
8: end for
9: return Auxiliary matrix $\mathbf{H}^\rho$;
```

As already done in Section 5.2 for simplicity of presentation, we first start by ignoring the maximum chip power constraint P_{max}. Therefore, by taking into account the thermal model of the chip, the power consumption on blocks of the floorplan that do not correspond to cores, the ambient temperature, and the power consumption of inactive cores, we derive the worst-case uniform power density budget (independent of the types of cores) for each active core in *any* possible mapping of cores with $\mathbf{m} = \{m_1, m_2, \ldots, m_Q\}$ simultaneously active cores that could potentially exceed P_{max}, such that the maximum temperature in the steady-state among all blocks does not exceed T_{DTM}. This power constraint is defined as $P^{\rho\star\text{worst}}_{\text{TSP}}(\mathbf{m})$.

In order to derive $P^{\rho\star\text{worst}}_{\text{TSP}}(\mathbf{m})$, we first rewrite Equation (5.8) by grouping together the mappings of different types of cores as

$$T_{\text{steady}_i} = P^\rho_{\text{equal}} \cdot \sum_{q=1}^{Q} \sum_{j=1}^{M^{\text{type}}_q} \tilde{b}_{i,k^q_j} \cdot \text{area}^{\text{type}}_q \cdot x^q_j + \sum_{q=1}^{Q} \sum_{j=1}^{M^{\text{type}}_q} \tilde{b}_{i,x^q_j} \cdot P^{\text{type}}_{\text{inact}_q}\left(1 - x^q_j\right) + \sum_{j=1}^{N} \tilde{b}_{i,j}\left(p^{\text{blocks}}_j + T_{\text{amb}} \cdot g_j\right),$$

after which we rewrite this expression now considering matrix $\mathbf{H}^\rho$ as

$$T_{\text{steady}_{\ell_i}} = P^\rho_{\text{equal}} \sum_{q=1}^{Q} \sum_{j=1}^{m_q} h^\rho_{q,i,j} + \sum_{q=1}^{Q} \frac{P^{\text{type}}_{\text{inact}_q}}{\text{area}^{\text{type}}_q} \cdot \sum_{j=m_q+1}^{M^{\text{type}}_q} h^\rho_{q,i,j} + \sum_{j=1}^{N} \tilde{b}_{\ell_i,j}\left(p^{\text{blocks}}_j + T_{\text{amb}} \cdot g_j\right).$$

From this last expression and by following a similar procedure as in Section 5.2.2, we can compute $P^{\rho\star\text{worst}}_{\text{TSP}}(\mathbf{m})$ as presented in Equation (5.12).

$$P^{\rho\star\text{worst}}_{\text{TSP}}(\mathbf{m}) = \min_{1 \le i \le Z} \left\{ \frac{T_{\text{DTM}} - \sum_{q=1}^{Q} \frac{P^{\text{type}}_{\text{inact}_q}}{\text{area}^{\text{type}}_q} \sum_{j=m_q+1}^{M^{\text{type}}_q} h^\rho_{q,i,j} - \sum_{j=1}^{N} \tilde{b}_{\ell_i,j}\left(p^{\text{blocks}}_j + T_{\text{amb}} \cdot g_j\right)}{\sum_{q=1}^{Q} \sum_{j=1}^{m_q} h^\rho_{q,i,j}} \right\} \tag{5.12}$$

Once again, now that we have $P_{\text{TSP}}^{\rho\star\text{worst}}(\mathbf{m})$ we can include P_{max} back into the formulation. Basically, if the power density in all active cores is equal to the entire per-core power density budget, considering the power consumption of the inactive cores and other blocks in the floorplan, the total power consumption should not exceed the value of P_{max}, i.e., it should hold that

$$P_{\text{TSP}}^{\rho\text{ worst}}(\mathbf{m}) \cdot \sum_{q=1}^{Q} \text{area}_q^{\text{type}} \cdot m_q + \sum_{q=1}^{Q} P_{\text{inact}q}^{\text{type}} \left(M_q^{\text{type}} - m_q\right) + \sum_{i=1}^{N} p_i^{\text{blocks}} \leq P_{\text{max}}.$$

From this equation, it is simple to derive the amount of power density that any $\mathbf{m} = \{m_1, m_2, \ldots, m_Q\}$ active cores can have, such that the total power consumption precisely reaches the value of P_{max}. For such a purpose, we define auxiliary function $P_{\text{max}}^{\text{core}\rho}(\mathbf{m})$, shown in Equation (5.13), used to compute this power density value.

$$P_{\text{max}}^{\text{core}\rho}(\mathbf{m}) = \frac{P_{\text{max}} - \sum_{i=1}^{N} p_i^{\text{blocks}} - \sum_{q=1}^{Q} P_{\text{inact}q}^{\text{type}} \left(M_q^{\text{type}} - m_q\right)}{\sum_{q=1}^{Q} \text{area}_q^{\text{type}} \cdot m_q} \tag{5.13}$$

Similar to Section 5.2, we then have that $P_{\text{TSP}}^{\rho\text{ worst}}(\mathbf{m})$ can be computed as presented in Equation (5.14). Given that matrix $\mathbf{H}^{\rho}$ only needs to be built once for a given chip using Algorithm 2, the total time complexity for computing $P_{\text{TSP}}^{\rho\text{ worst}}(\mathbf{m})$ for a given $\mathbf{m} = \{m_1, m_2, \ldots, m_Q\}$ is $O(ZN)$, and for computing $P_{\text{TSP}}^{\rho\text{ worst}}(\mathbf{m})$ for all possible combinations of active cores of different types is $O\left(ZN \prod_{q=1}^{Q} M_q^{\text{type}}\right)$.

$$P_{\text{TSP}}^{\rho\text{ worst}}(\mathbf{m}) = \begin{cases} P_{\text{TSP}}^{\rho\star\text{worst}}(\mathbf{m}) & \text{if } P_{\text{TSP}}^{\rho\star\text{worst}}(\mathbf{m}) \leq P_{\text{max}}^{\text{core}\rho}(\mathbf{m}) \\ P_{\text{max}}^{\text{core}\rho}(\mathbf{m}) & \text{otherwise.} \end{cases} \tag{5.14}$$

Note that the algorithms for homogeneous systems presented in Section 5.2 are a special case of the algorithms presented in this section. Particularly, when the area of all cores is constant, i.e., $\text{area}_j^{\text{core}} = \text{area}_{j+1}^{\text{core}}$ for $j = 1, 2, \ldots, M-1$, and when the power consumption of all inactive cores is also constant, i.e., $P_{\text{inact}j}^{\text{core}} = P_{\text{inact}j+1}^{\text{core}}$ for $j = 1, 2, \ldots, M-1$, then multiplying the derived power density budget with the area of the cores results in the same TSP values computed using the algorithms from Section 5.2.

5.4 Transient State Considerations

Depending on the executed applications, the task partitioning, the task-to-core mapping, and the DPM/DVFS policies implemented in the system, the power consumption and the number of active cores throughout the chip could change very frequently (in the order of milliseconds) or it could change rarely (in the order of several seconds). The former happens very often in normal systems, e.g., when there are context switches inside different cores due to task preemption, when some cores become idle waiting for data from memory or waiting for some other thread to finish its computation before being able to continue, etc. The latter may occur in scenarios with long running applications that have no more running threads than cores (such that preemption is not needed). In either case, when drastic power changes occur and depending on the adopted DVFS policy, the temperature of some cores might exceed the value of the critical temperature due to the effects of transient temperatures. When this happens, DTM is triggered in order to avoid damages to the chip, resulting in lower performance than originally expected. *Unless care is taken, these transient thermal effects can be observed in any power budgeting technique derived for the steady-state temperatures, e.g., for constant power budgets like TDP, and also for some cases with TSP.*

The following example briefly illustrates how this effect could possibly occur for systems constrained both by (1) TDP and (2) TSP. Consider the 16-core homogeneous system presented in Figure 4.3a (described in Chapter 4.2.1), with a $P_{\text{inact}}^{\text{core}}$ of 0 W, and (1) a TDP of 90.2 W used as a per-chip power budget, or (2) $P_{\text{TSP}}^{\text{worst}}(4) = 14.67\,\text{W}$ and $P_{\text{TSP}}^{\text{worst}}(8) = 11.27\,\text{W}$, i.e., a per-core TSP of 14.67 W and 11.27 W when activating 4 and 8 cores, respectively. Under these assumptions, Figure 5.9 presents simulations in which there are 8 active cores according to Figure 5.3b during $t = [0\,\text{s}, 0.5\,\text{s}]$, each core consuming 11.27 W (90.2 W in total).

During $t = [0.5\,\text{s}, 1\,\text{s}]$, these cores are shut-down and we activate other 4 cores according to Figure 5.3a, each core consuming $14.67\,\text{W}$ ($58.7\,\text{W}$ in total). Therefore, when using (1) TDP as a per-chip power budget, the system consumes the entire TDP budget during $t = [0\,\text{s}, 0.5\,\text{s}]$, and less than TDP during $t = [0.5\,\text{s}, 1\,\text{s}]$. Moreover, when constrained by (2) TSP, the system consumes the corresponding TSP values according to the number of active cores at all times. However, although the power budgets are being satisfied in both cases, Figure 5.9 shows that during $t = [0.5\,\text{s}, 1\,\text{s}]$ the temperature of at least one core exceeds the 80°C critical threshold temperature that triggers DTM. Such a transient effect, i.e., when we have transient temperature peaks that are higher than the corresponding steady-state temperatures, normally occurs when the power density of some cores is incremented during a change in power. For example, for the opposite case in which we transition from the mapping in Figure 5.3a to that in Figure 5.3b, given that the power density in all the active cores decreases, the transient temperatures remain below both steady-states, as shown in Figure 5.10. *Other examples about this issue and an analytical method to compute the transient peaks in temperature, called MatEx, are later presented in Chapter 6.*

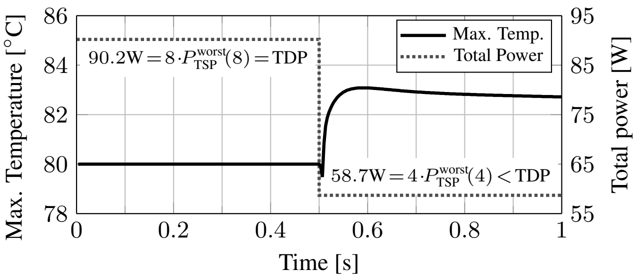


Figure 5.9: Transient example for both TSP and TDP (the bold line shows the maximum temperature among all elements in the chip). During $t = [0\,\text{s}, 0.5\,\text{s}]$ there are 8 active cores according to Figure 5.3b, each core consuming $11.27\,\text{W}$. During $t = [0.5\,\text{s}, 1\,\text{s}]$, these cores are shut-down and we activate 4 cores according to Figure 5.3a, each core consuming $14.67\,\text{W}$.

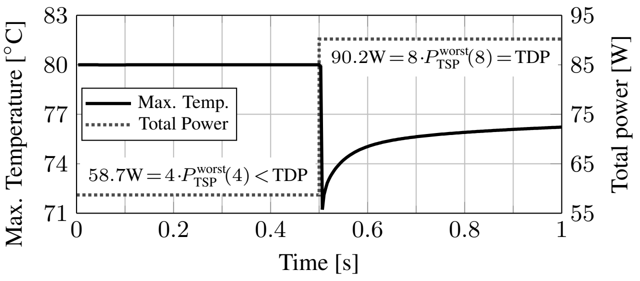


Figure 5.10: Transient example for both TSP and TDP (the bold line shows the maximum temperature among all elements in the chip). During $t = [0\,\text{s}, 0.5\,\text{s}]$ there are 4 active cores according to Figure 5.3a, each core consuming $14.67\,\text{W}$. During $t = [0.5\,\text{s}, 1\,\text{s}]$, these cores are shut-down and we activate 8 cores according to Figure 5.3b, each core consuming $11.27\,\text{W}$.

If the frequency of the power changes that produce this transient effect is very high, then DTM could potentially be triggered frequently, and the associated performance losses (compared to the expected performance) would be noticeable. There are several approaches that can be used to deal with this issue in regards to TSP, and in this section we detail two of them. Particularly, one approach (discussed in Section 5.4.1) is to adjust the temperature for which we compute TSP such that the transient peak temperatures are constrained below T_{DTM} even if we always adjust the DVFS levels to consume the entire TSP budget according to the number of active cores at any given time. The other approach (discussed in Section 5.4.2) is to maintain the DVFS levels at nominal operation for a given mapping by ignoring the partial number of active cores due to cores being idle while waiting for data from memory or waiting for other threads to finish.

5.4.1 Adjusting the Temperature for Computing TSP

For this approach, we need to quantify the maximum values that the transient peak temperatures can actually reach during the transient state. We denote the difference between such maximum transient temperatures and the value of T_{DTM} as $\Delta T_{\text{transient}}^{\text{max}}$, with value 3.08°C for the example in Figure 5.9. Therefore, if instead of computing TSP for temperature T_{DTM} we compute it for temperature $T_{\text{DTM}} - \Delta T_{\text{transient}}^{\text{max}}$, we can make sure that the transient temperatures never exceed T_{DTM}. Nevertheless, depending on the thermal model of a chip and its resulting thermal capacitances, it may occur that the transient temperatures for this new TSP value are too pessimistic compared to T_{DTM}. For such cases, with just a few iterations, a near optimal value for which to compute TSP can be derived. This method should be applied at design-time, due to the large required design-space exploration and the overheads for obtaining $\Delta T_{\text{transient}}^{\max}$ for each case. *A similar method should also be adopted for systems that use constant power budgets, e.g., TDP.* Furthermore, the MatEx technique later described in Chapter 6 is a lightweight tool for computing the transient peak temperatures, and can therefore be used to speed-up the computation of $\Delta T_{\text{transient}}^{\max}$.

Procedure Example

Consider the 16-core homogeneous system presented in Figure 4.3a (described in Chapter 4.2.1) and an ambient temperature of 45°C. For the power consumption values, consider a hypothetical scenario in which the power of the cores changes every 0.1 seconds, and these changes in power are for a random number of active cores. The mapping and power values adopted in all cases are those of TSP for the worst cases, computed using Equation (5.6) and Equation (5.7), according to the number of active cores at each point in time. Under these assumptions, we run simulations for such a case and present the resulting temperatures in Figure 5.11a. In Figure 5.11a, we can observe that for our experimental settings (detailed in Chapter 4) the thermal capacitances of the resulting thermal model are not negligible, which results in long transient-state periods. Therefore, for such a case, the TSP values should be recomputed for some temperature below T_{DTM}.

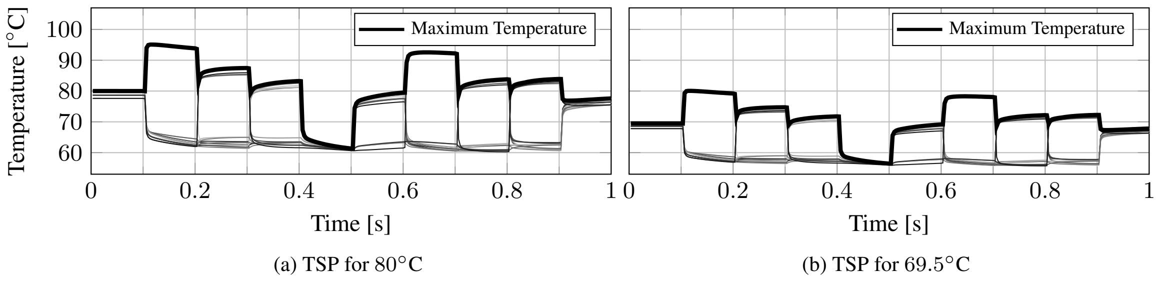


Figure 5.11: Transient example for 16 cores. The number of active cores and their power consumption changes every 0.1 s. The adopted mapping and power values are those of TSP for the worst-case mappings, computed for (a) 80°C and (b) 69.5°C. The temperature on each core is illustrated using a different color, and the maximum temperature among all cores at any given time is highlighted by the bold curve.

Looking at Figure 5.11a, we can quantify the value of $\Delta T_{\text{transient}}^{\max}$, which we set to 15°C, and then we recompute TSP for 65°C. After re-conducting the experiments, we observe that this results in a maximum transient temperature of 74°C, which is too pessimistic and thus we need a higher value. Therefore, we iterate computing TSP and running transient temperature simulations. After just 5 iterations, particularly, by computing TSP for 80°C, 65°C, 71°C, 69°C, and 69.5°C, we reach a near optimal value for which to compute TSP, which for this thermal model is 69.5°C. Naturally, the new TSP values are smaller than those for a TSP computed for 80°C. Finally, Figure 5.11b presents simulation results for a similar experiment than that in Figure 5.11a, but with power states according to the new TSP values computed for 69.5°C. From the figure, we can observe that the transient temperatures are always below T_{DTM}. When computing TSP at runtime for particular mapping scenarios, the target temperature should also be 69.5°C, and not the original 80°C.

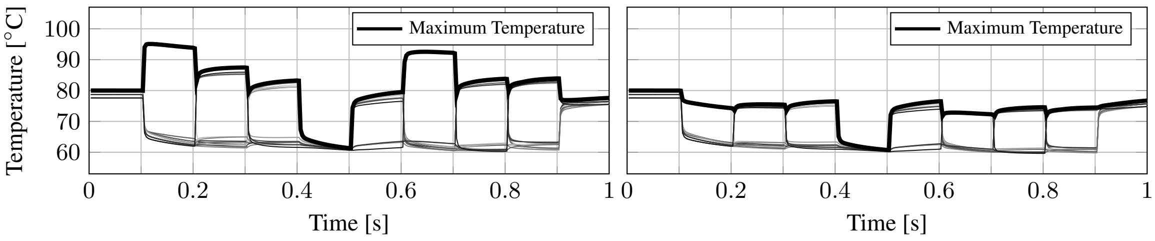
If, *unlike* Figure 5.11, the changes in power do *not* occur very frequently, such that DTM is triggered with low frequency and during short time intervals, then computing TSP for the original 80°C could still prove to be a better approach that results in higher total performance.

5.4.2 Nominal DVFS Operation for a Given Mapping

Another approach for dealing with the transient peak temperatures is to keep the DVFS levels at nominal operation for a given mapping by ignoring the partial number of active cores due to cores being idle while waiting for data from memory or waiting for other threads to finish. For example, considering a 64-core system like the one shown in Figure 3.2, assume a situation in which the operating system partitions the tasks and maps them to cores such that we have 32 cores with threads assigned to them while the other 32 cores remain power gated. For such a case, the DVFS levels of the 32 active cores can be set such that the power consumption in every active core is below the TSP values for the worst-case mappings with 32 active cores, i.e., $P_{\text{TSP}}^{\text{worst}}(32)$. Normally, there will be time intervals (which could last some milliseconds or entire seconds) during which some of these 32 active cores will remain idle in practice, e.g., when a thread is locked waiting for data from another thread to continue. When this occurs, there are two possible alternatives of how the system could operate: (1) change the DVFS levels in order to satisfy the TSP values for the partial number of active cores, or (2) maintain the same DVFS levels that satisfy TSP when there are 32 active cores.

It is important to remember at this point that the TSP values result in a decreasing function with respect to the number of active cores, as illustrated in Figure 5.6. Therefore, given that the TSP values for less than 32 active cores are larger than the TSP values for 32 active cores, this means that operating under alternative (1) we can potentially achieve higher system performance by dynamically changing the DVFS levels, since cores can execute at faster frequencies while satisfying TSP. However, this is precisely the case in which we can have many transient thermal violations, as explained above and shown in Figure 5.11a.

Contrarily, operating under alternative (2) is a conservative approach which needs little additional considerations. Namely, since the TSP values for less than 32 active cores are larger than the TSP values for 32 active cores, if the DVFS levels are not changed, then individual cores will not consume more than $P_{\text{TSP}}^{\text{worst}}(32)$ and this is safe for any scenario with less than 32 active cores. Moreover, given that the power density of the active cores remains constant under alternative (2), in this scenario the transient temperatures will generally remain under T_{DTM} and hence DTM will not be frequently triggered. This statement is supported by Figure 5.12, showing further simulations for the 16-core system from Figure 4.3a. Figure 5.12a presents the same simulations as in Figure 5.11a, in which the DVFS levels are constantly changing to match the TSP values for the partial number of active cores for every time interval. On the other hand, Figure 5.12b illustrates the resulting temperatures for the same assumptions and scenario, but keeping the DVFS levels constant at nominal operation when there are 16 active cores, where it can be observed that all transient temperatures remain below the value of T_{DTM}. *Finally, note that under alternative (2) the DVFS levels are not maintained always constant, but are rather set to nominal values. In other words, when an application finishes its execution or when a new application arrives, the runtime management system will change the mapping of threads and compute new nominal DVFS levels to satisfy the TSP values for the new mapping. Alternative (2) simply maintains the DVFS levels constant when the* partial *number of active cores changes due to core idling or similar situations.*



(a) DVFS for partial active cores

(b) Constant DVFS

Figure 5.12: Transient example for 16 active cores when using different DVFS policies for idling. The partial number of active cores changes every 0.1 seconds, e.g., when threads become idle waiting for other threads to finish. In (a), DVFS is used to match the worst-case TSP values for the partial number of active cores in every time interval. In (b), DVFS levels are maintained constant to match the worst-case TSP values when no core is idle, i.e., for 16 active cores. The temperature on each core is illustrated using a different color, and the maximum temperature among all cores at any given time is highlighted by the bold curve.

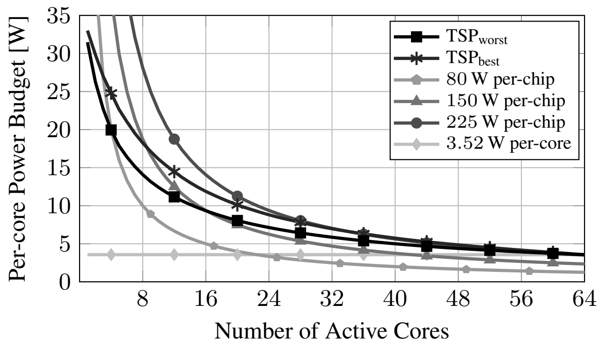
5.5 Experimental Evaluations for Homogeneous Systems

This section presents experimental evaluations for homogeneous systems, that compare the total system performance of seven different power budget techniques: TSP for given mappings, TSP for the worst-case mappings, three constant per-chip power budgets, a constant per-core power budget, and a runtime boosting technique, specifically, Intel's Turbo Boost [9, 10, 35, 91] (described in Chapter 2.1.2). We also show how TSP can be used to estimate the amount of dark silicon [6, 19, 95], and how such estimations are much more realistic than those considering constant power budgets.

For the evaluations, we use the simulation framework described in Chapter 4 in detailed mode, considering the homogeneous architecture illustrated in Figure 3.2 and described in Chapter 4.2.1. From the PARSEC benchmarks described in Chapter 4.3, we use four representative applications, specifically, *x264*, *bodytrack*, *blackscholes*, and *swaptions*.

5.5.1 Power Constraints

In this section we compute the power constraints used in our experiments. By using Equation (5.6) and Equation (5.7), we compute TSP for the worst-case mappings, i.e., $P_{\text{TSP}}^{\text{worst}}(m)$ for all $m = 1, 2, \ldots, M$. Figure 5.13 presents the computed per-core TSP values as TSP$_{\text{worst}}$, which results in a decreasing function with respect to the number of simultaneously active cores. For presentation purposes, by multiplying the TSP$_{\text{worst}}$ values from Figure 5.13 with the number of active cores for each case, Figure 5.14 also presents TSP estimations at a chip level, resulting in a non-decreasing function with respect to the number of active cores. Furthermore, for all possible number of active cores $m = 1, 2, \ldots, M$, we obtain the best-case core mappings for a uniform power budget through integer linear programming, and compute the TSP values for such given mappings by using Equation (5.5), presented in Figure 5.13 and Figure 5.14 as TSP$_{\text{best}}$. This allows us to estimate how much pessimism we can have when considering TSP values for the worst-case mappings compared to the best-case mappings, such that we quantify the highest possible performance losses incurred when simplifying the resource management by abstracting from mapping decisions.

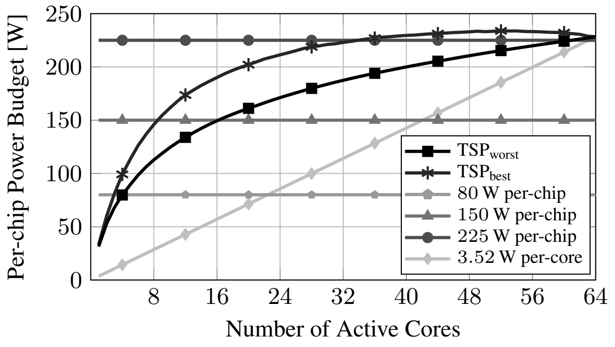


Active Cores	Per-core Power Budget [W]					
	TSP$_{\text{worst}}$	TSP$_{\text{best}}$	3.52 W per-core	80 W per-chip	150 W per-chip	225 W per-chip
8	14.09	18.24	3.52	10.00	18.75	28.13
16	9.34	11.88	3.52	5.00	9.38	14.06
24	7.13	8.81	3.52	3.33	6.25	9.38
32	5.86	6.97	3.52	2.50	4.69	7.03
40	5.00	5.74	3.52	2.00	3.75	5.63
48	4.39	4.85	3.52	1.67	3.13	4.69
56	3.93	4.16	3.52	1.43	2.68	4.02
64	3.52	3.52	3.52	1.25	2.34	3.52

Figure 5.13: Worst-case and best-case TSP for the 64-core homogeneous architecture illustrated in Figure 3.2 and described in Chapter 4.2.1, compared to a constant per-core power budget, and estimations of constant per-chip power budgets equally distributed among the active cores.

For the constant per-chip power budgets, we cannot simply consider TDP, as this is a simulated platform for which we do not have a datasheet with that information. Therefore, we consider three different per-chip power budgets, which coincide with $m \cdot P_{\text{TSP}}^{\text{worst}}(m)$ for $m = 4$, $m = 16$, and $m = 64$. Particularly, these power budgets are 80 W per-chip, 150 W per-chip, and 225 W per-chip. These constant per-chip power budgets are representative TDP values for current technologies [34], and are shown as horizontal lines in Figure 5.14. In Figure 5.13, we estimate the maximum power consumed by individual cores when these per-chip power budgets are equally distributed among the active cores. For the constant per-core power budget, we consider it to be equal to TSP when simultaneously activating all cores, i.e., $P_{\text{TSP}}^{\text{worst}}(m)$ for $m = 64$. This results in a constant per-core power budget of 3.52 W, which is represented by a horizontal line in Figure 5.13 and by an increasing linear function in Figure 5.14. *Note that representing TSP and the constant per-core power budget in Figure 5.14 at a chip level does not imply that either constraint should be considered as a per-chip power budget. Both budgets should be strictly considered at a per-core level, and we include them in Figure 5.14 only as a mean to compare their resulting total chip power consumption with the other per-chip power budgets. Similarly, the opposite applies when representing the constant per-chip power budgets in Figure 5.13.*

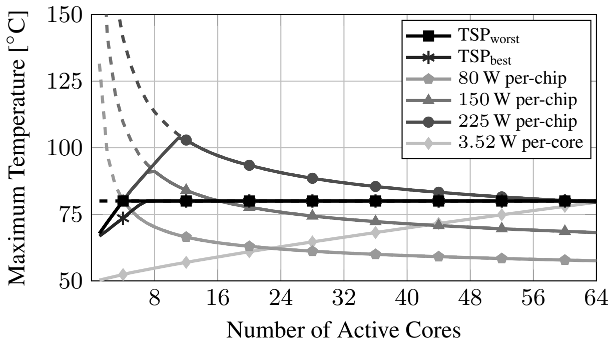
With regards to temperature, Figure 5.15 presents simulation results that show the maximum steady-state



Active Cores	Per-chip Power Budget [W]					
	TSP$_{\text{worst}}$	TSP$_{\text{best}}$	3.52 W per-core	80 W per-chip	150 W per-chip	225 W per-chip
8	112.76	145.94	28.16	80.00	150.00	225.00
16	149.45	190.06	56.32	80.00	150.00	225.00
24	171.18	211.35	84.48	80.00	150.00	225.00
32	187.43	223.00	112.64	80.00	150.00	225.00
40	199.96	229.57	140.80	80.00	150.00	225.00
48	210.51	232.91	168.96	80.00	150.00	225.00
56	219.82	233.13	197.12	80.00	150.00	225.00
64	225.28	225.28	225.28	80.00	150.00	225.00

Figure 5.14: Constant per-chip power budgets, compared to estimations by multiplying the number of active cores with a constant per-core power budget, and the worst-case and best-case TSP values, for the 64-core homogeneous architecture illustrated in Figure 3.2 and described in Chapter 4.2.1.

temperature throughout the chip as a function of the number of simultaneously active cores, for the discussed power budgets, considering that DTM is deactivated. As expected, when consuming TSP in all active cores, the maximum steady-state temperature on the chip is 80°C. Moreover, from Figure 5.15 we can conclude that whenever the value of TSP for a given number of active cores is greater than any constant power budget, in case such a power budget is used as the power constraint, the system could actually consume more power without exceeding T_{DTM}, which accounts for performance losses. On the other hand, whenever a constant power budget exceeds the value of TSP for a given number of active cores, this implies that if the cores consume more power than TSP, most likely DTM would be very frequently triggered due to thermal violations. Note that for some cases in Figure 5.15, especially when having just a few active cores, the cores never consume the entire power budgets even when executing at the maximum DVFS levels (e.g., in our experiments no individual core ever consumes more than 20 W). Hence, the dashed lines in the figure illustrate the *potential* maximum steady-state temperature in case every power budget would have been entirely consumed, while the solid lines show the maximum steady-state temperatures that can be practically achieved for this type of core.



Active Cores	Maximum Temperature [°C]					
	TSP$_{\text{worst}}$	TSP$_{\text{best}}$	3.52 W per-core	80 W per-chip	150 W per-chip	225 W per-chip
8	80.00	80.00	54.81	70.25	91.10	94.08
16	80.00	80.00	58.98	64.34	80.13	97.05
24	80.00	80.00	62.83	61.96	75.82	90.66
32	80.00	80.00	66.41	60.51	73.21	86.83
40	80.00	80.00	69.87	59.52	71.47	84.28
48	80.00	80.00	73.20	58.75	70.15	82.36
56	80.00	80.00	76.41	58.11	69.07	80.82
64	80.00	80.00	79.51	57.57	68.16	79.51

Figure 5.15: Maximum steady-state temperatures throughout the chip for the 64-core homogeneous architecture illustrated in Figure 3.2 and described in Chapter 4.2.1, with DTM deactivated, when using TSP, a constant per-core power budget, and three equally distributed constant per-chip power budgets.

5.5.2 Execution Time of Online TSP Computation

In order to verify the applicability of TSP computations at runtime, we measure the execution time required by an application implementing Equation (5.3) and Equation (5.5) (implemented in C++ as a single-threaded application) on a desktop computer with a 64-bit quad-core Intel Sandybridge i5-2400 CPU running at 3.10 GHz. We consider several floorplans with different number of cores for every floorplan. For every floorplan, we consider 25000 random mappings and present the maximum measured execution time for each case in Figure 5.16, from which we can conclude that TSP is *suitable for runtime usage*. Moreover, given that in order to compute TSP, Equation (5.3) needs to find the minimum value of P_{equal} among all blocks in the floorplan, TSP could be easily implemented as a multi-threaded application (e.g., one thread computing P_{equal} for each block), and thus parallelized into multiple cores, further reducing the execution time.

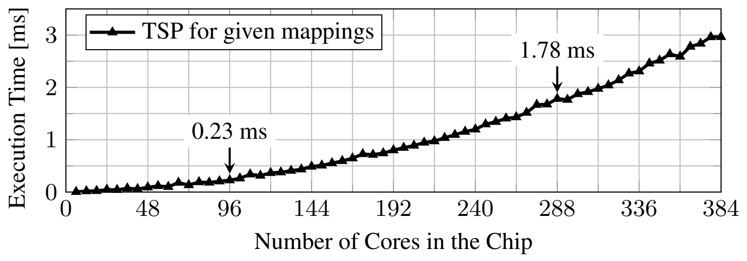


Figure 5.16: Execution time for computing TSP for a given mapping, considering several floorplans with different number of cores in each floorplan.

5.5.3 Dark Silicon Estimations

From Figure 5.17 (a summarized version of Figure 5.13), we can easily estimate the amount of dark silicon for a given power consumption on the cores. For example, consider that we would like to execute the active cores at certain DVFS levels, such that the resulting power consumption in each core is 4 W. Then, considering the values of TSP for this chip and cooling solution, it would not be possible to simultaneously activate more than 54 cores for such a case (as doing so would violate TSP), resulting in 15.63% of the chip being dark at all times for these desired DVFS levels. Contrarily, if a constant per-chip power budget equally distributed among all active cores is used for the same example estimations, e.g., 150.0 W per-chip, then it would not be possible to active more than 37 cores simultaneously, resulting in 42.19% of the chip being dark, which is much higher than the estimations for TSP. This happens because, as explained in Section 5.5.1, whenever the value of TSP for a given number of active cores is greater than a constant power budget, then using the constant power budget as a constraint would keep the system underutilized.

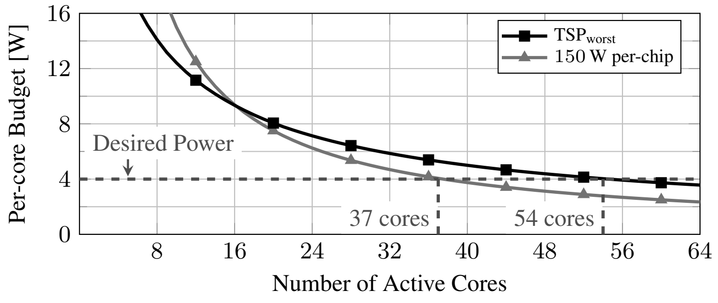


Figure 5.17: Example of dark silicon estimations when using TSP for the worst-case mappings and a constant per-chip power budget, for the 64-core homogeneous architecture illustrated in Figure 3.2 and described in Chapter 4.2.1

Moreover, when estimating the amount of dark silicon for another desired power consumption value on the active cores for which the constant power budget exceeds the value of TSP for a given number of active cores (e.g., 14 W in this example), the TSP estimations would be more pessimistic than the estimations for the constant power budget. However, as explained in Section 5.5.1, when this occurs, if the constant power budget is used as a constraint, the most likely scenario is that DTM would be very frequently triggered due to thermal violations, meaning that the associated dark silicon estimations would not be valid.

5.5.4 Performance Simulations

Figure 5.18 presents the resulting average total performance (using IPS as metric), for considering different numbers of active cores and the different power budgets described in Section 5.5.1. We select the nominal DVFS levels for operation of every active core for each case, such that the performance of each specific application is maximized without exceeding the power budget under consideration.

In Figure 5.18, we can observe that the per-core budget and the 225 W per-chip budget achieve the same performance as $\text{TSP}_{\text{worst}}$ when all 64 cores are simultaneously active. This is an expected result, as in both

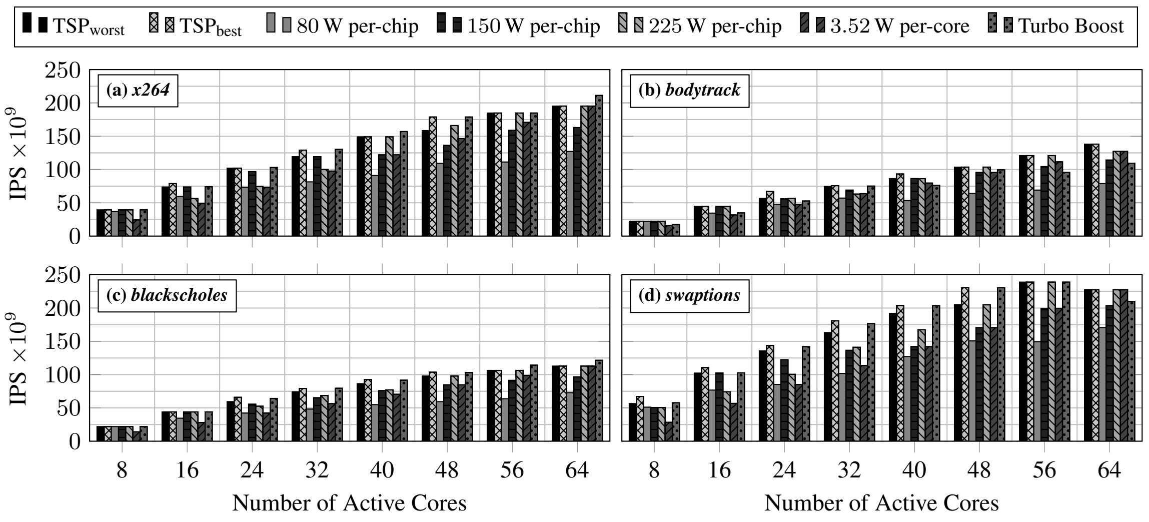


Figure 5.18: Experimental evaluation results for homogeneous systems showing the average total system performance when using different power budgets.

cases these power budgets coincide with the TSP values. However, there are many cases in which the fixed per-chip and per-core power budgets are pessimistic and therefore the chip remains underutilized, as explained in Section 5.5.1. Particularly, this happens with all other cases for the per-core power budget, and when activating more than 4 and 16 cores with the $80\,$W and $150\,$W per-chip power budgets, respectively. On the other hand, there are many other cases in which the fixed per-chip power budgets constantly trigger DTM, specifically, when activating less than 16 cores under the $150\,$W per-chip budget, and when activating less than 64 cores under the $225\,$W per-chip budget. Here, given that DTM is constantly triggered, thermal violations are avoided at the cost of reducing the DVFS levels of the cores during long periods of time. Therefore, the resulting performance of the per-chip power budgets is generally much worse than originally expected by the task partitioning and mapping algorithms. Furthermore, there is no simple way of predicting the performance losses for such cases, making it almost impossible for the system to provide performance and timing guarantees. Contrarily, TSP never triggers DTM and can thus achieve the performance expected by the task partitioning and mapping decisions. The *average percentage increase in performance* (among all number of active cores and all applications) for using TSP$_{worst}$ as a per-core power budget results in a 12% higher average total IPS when compared to all the evaluated constant per-chip and per-core power budgets. There are just a few cases in which the resulting performance of the per-chip power budgets is higher than that of TSP. This happens mostly because a per-chip power budget can potentially execute different cores at different DVFS levels, thus reducing the thermal headroom for these few cases. Contrarily, under TSP all cores share the same per-core power budget, and sometimes a certain DVFS level would slightly exceed this budget, while a lower DVFS level would result in a large thermal headroom.

Intel's Turbo Boost is a simple but very efficient runtime technique, achieving higher performance than TSP$_{worst}$ in some cases. Namely, while under TSP the DVFS levels of the cores are maintained constant (as does the performance), under Turbo Boost the DVFS levels are constantly changing, thus exploiting the thermal capacitances of the thermal model by knowing that the associated temperature changes require some time to follow the changes in power. In this way, the instantaneous performance of Turbo Boost can sometimes be much higher than TSP during some time intervals (when there is thermal headroom and the DVFS levels are increased), and much lower during other time intervals (when the critical temperature is reached and the DVFS levels are reduced). After computing the average performance of Turbo Boost among all number of active cores and all applications, we observe that Turbo Boost and TSP$_{worst}$ result in the same average total IPS. However, similar to having frequent triggers of DTM, there are no simple performance predictions which are suitable for estimating the behavior of such boosting techniques a priori, and thus no timing guarantees can be easily provided in advance. Hence, Turbo Boost cannot be used to guide the task partitioning and mapping algorithms to make intelligent decisions. Contrarily, this can be done with TSP, helping to simplify task partitioning and mapping algorithms to achieve high *predictable* performance without thermal violations.

Finally, there are no restrictions that prohibit the combination of Turbo Boost and TSP, and this can potentially result in higher performance than applying each technique separately. Namely, TSP may be used to guide the task partitioning, mapping, DPM, and DVFS selection algorithms to make intelligent decisions that optimize the performance without incurring in thermal violations at nominal operation. Then, Turbo Boost may be applied on top of such a solution in order to exploit any thermal headroom available at runtime, by boosting the cores to execute at power levels higher than the TSP values. Opposed to standard Turbo Boost, in which once the critical threshold temperature is reached the DVFS levels must be decreased until the temperature is again below the T_{DTM}, in this case Turbo Boost can stop decreasing the DVFS levels of the cores at the nominal operation levels that satisfy TSP, given that we know that these are thermally safe operation levels. *Therefore, by combining TSP and Turbo Boost, the system can potentially achieve high predictable performance while also exploiting the thermal headroom available at runtime.*

5.6 Experimental Evaluations for Heterogeneous Systems

This section presents evaluations for heterogeneous systems. Similarly as described in Section 5.5, we use the simulation framework described in Chapter 4 in detailed mode, considering the heterogeneous architecture illustrated in Figure 4.4 and described in Chapter 4.2.2. Given that the Odroid-XU3 platform does not provide performance counters to measure the total number of executed instructions, we use throughput as our performance metric, where throughput is defined as the total number of application instances finished (or partially finished) every second.

From the PARSEC benchmarks described in Chapter 4.3, we again consider four representative applications, specifically, *x264*, *bodytrack*, *blackscholes*, and *swaptions*. Furthermore, given that different applications have different power consumptions depending on the type of cores and number of threads in which they are executed, we run the applications under different scenarios. Specifically, we focus on different applications individually, considering multiple instances of the same application, with different number of threads per instance and also different thread-to-core mappings, as detailed in Table 5.1.

Scenario	**Alpha OOO**	**Alpha *simple***	**A15**	**A7**
S1	a: 8 threads b: 8 threads c: 2 threads 6 threads	a: - b: - c: 3 threads d: 2 threads	a: 4 threads b: 2 threads c: 2 threads d: -	a: - b: 2 threads c: 2 threads d: 4 threads
S2	a: 5 threads 3 threads b: - c: 7 threads 1 thread	a: 4 threads b: 1 thread c: 2 threads d: -	a: 2 threads b: - c: 1 thread d: 4 threads	a: 4 threads b: - c: - d: 2 threads
S3	a: 4 threads b: 4 threads 4 threads c: -	a: 2 threads b: - c: 3 threads d: -	a: 4 threads b: 1 thread c: - d: -	a: 2 threads b: 2 threads c: - d: -
S4	a: 4 threads 4 threads b: 4 threads 4 threads c: 4 threads 4 threads	a: 4 threads b: 4 threads c: 4 threads d: 4 threads	a: 4 threads b: 4 threads c: 4 threads d: 4 threads	a: 4 threads b: 4 threads c: 4 threads d: 4 threads

Table 5.1: Details of the application mapping scenarios for our experiments. Indexes a, b, c, and d represent the cluster ID as explained in Figure 4.4. Every line corresponds to an application instance executed in the corresponding cluster with the indicated number of threads, where "-" means that a cluster is not executing any application.

5.6.1 Power Constraints

For our heterogeneous evaluations, we use similar power budgets as those described in Section 5.5.1, but extended for heterogeneous systems. Using Equation (5.11), we compute TSP for the given mappings detailed in Table 5.1. For the per-chip power budgets, we choose 205 W per-chip (i.e., the total active power of using TSP when all cores are activated), as well as 140 W and 70 W per-chip (to have a similar relation with the per-chip budgets from Section 5.5.1). For the evaluations in Section 5.5, the total per-chip power budgets were equally divided among the number of active cores for each experiment. Similarly, for the heterogeneous case, the per-chip power budgets are *proportionally* divided according to the *area of the active cores*. For example, when activating two OOO Alpha cores and three Cortex-A7 cores under the $140\,\mathrm{W}$ per-chip power budget, the total active area for these cores is $2 \cdot 9.6\,\mathrm{mm}^2 + 3 \cdot 0.8\,\mathrm{mm}^2 = 21.6\,\mathrm{mm}^2$, such that each OOO Alpha core can consume up to $\frac{9.6\,\mathrm{mm}^2}{21.6\,\mathrm{mm}^2} \cdot 140\,\mathrm{W} = 62.2\,\mathrm{W}$ and each Cortex-A7 core can consume up to $5.2\,\mathrm{W}$. Finally, we use $0.586\,\mathrm{W/mm}^2$ as the per-core power budget, simply by dividing $205\,\mathrm{W}$ by the total core area in the chip, i.e., $350.36\,\mathrm{mm}^2$.

5.6.2 Performance Simulations

Figure 5.19 presents the average total throughput for considering the different mappings from Table 5.1 and the different power budgets described in Section 5.6.1. Similar to Section 5.5.4, we also compare with Intel's Turbo Boost [34, 91]. The observations of the results are very similar to those in Section 5.5.4. Additionally, developing intelligent and efficient task partitioning and mapping algorithms is much more challenging for heterogeneous systems than it is for the special case of homogeneous cores. Hence, TSP can dearly help to reduce the complexity of such decisions, achieving high *predictable* performance without thermal violations.

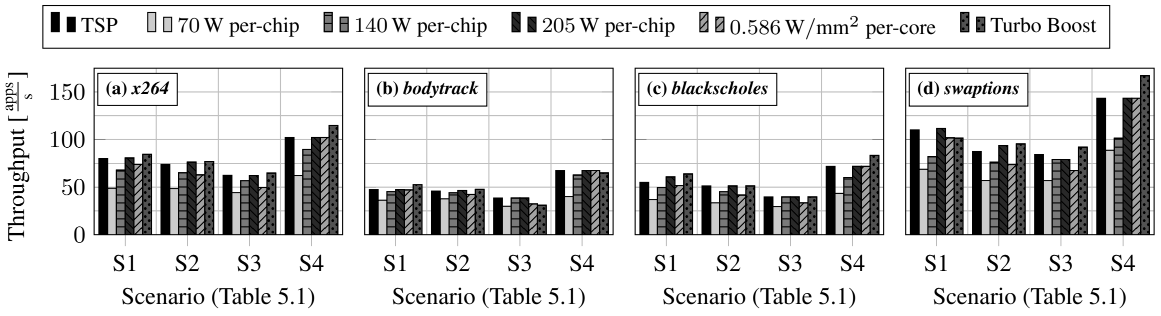


Figure 5.19: Experimental evaluation results for heterogeneous systems showing the average total system performance when using different power budgets.

5.7 Summary

Using a *single* and *constant* per-chip or per-core power budget as an abstraction from thermal problems (e.g., TDP), is a pessimistic approach for homogeneous and heterogeneous manycore systems. Therefore, this chapter presents a novel thermal-aware power budget concept called Thermal Safe Power (TSP), which is an abstraction that provides *safe* and *efficient* per-core power budget values as a function of the number of simultaneously active cores. Using TSP results in high total system performance, while the maximum temperature in the chip remains below the critical level that triggers DTM. TSP is a new step and advancement towards dealing with the dark silicon problem as it alleviates the unrealistic bounds of TDP and enables system designers and architects to explore new avenues for performance improvements in the dark silicon era. Furthermore, TSP can also serve as a fundamental tool for guiding task partitioning, core mapping, DPM, and DVFS algorithms on their attempt to achieve high predictable performance under thermal constraints.

For a specific chip, cooling solution, and ambient temperature, TSP can be used to obtain safe power and power density budgets for the worst cases, allowing the system designers to initially abstract the mapping decisions. Moreover, TSP can also be computed at runtime for a particular mapping of active cores and ambient temperature. The experimental evaluations show the validity of our arguments, comparing the total performance of using TSP, several constant per-chip/per-core power budgets, and a runtime boosting technique. TSP can also be used to estimate the amount of dark silicon, which results in much more realistic estimations than those using constant power budgets.

Finally, Section 5.4 raised an interesting problem by showing that power changes on the chip can result in much higher transient temperatures than any steady-state scenarios. This motivates for the development of a fast and accurate method to compute such transient temperature peaks, which is the focus of Chapter 6. Furthermore, the technique described in Chapter 6 can be used to assist the approach presented in Section 5.4.1 to quantify the maximum values that the transient peak temperatures can reach during the transient state.

Chapter 6

Transient and Peak Temperature Computation based on Matrix Exponentials (MatEx)

6.1 Overview

Runtime/design-time management decisions, such as mapping new application tasks/threads to cores, migrating tasks/threads among cores, scheduling tasks in individual cores, activating/deactivating cores (i.e., DPM decisions), changing the DVFS levels, etc., are typically used by resource management techniques to optimize the usage of the available resources. Among the existing techniques in the literature, there are several power budgeting and thermal management techniques that are derived/formulated for the steady-state temperatures (e.g., [31, 48, 64, 96]). However, management decisions change the power consumption throughout the chip, and this can in turn result in transient temperatures which are much higher than any expected steady-state scenarios (as shown in the examples in Chapter 5.4 and Section 6.1.1). If this occurs and the transient temperatures are higher than the critical threshold temperature, some DTM technique would be activated on the chip to guarantee that it is not damaged. Nevertheless, very frequent triggers of aggressive DTM techniques may degrade the overall performance of the system in an unpredictable manner (from the perspective of the resource management techniques). More importantly, chips could also be seriously damaged if in some case the transient temperatures grow at a faster rate than the speed in which DTM can react to them. In order for the system to operate in thermally safe ranges and have a predictable behavior, resource management techniques could thus benefit from evaluating (i.e., estimating or predicting) such transient temperature peaks when making management decisions.

In this chapter, we present a *lightweight* and *accurate* method for computing the peaks in transient temperatures at runtime. Our technique, called MatEx [73, 74], is suitable for any compact thermal model that consist in a system of first-order differential equations, e.g., a thermal model based on RC thermal networks (like the one used by HotSpot [33]). Most existing state-of-the-art techniques/tools for temperature computation/estimation/prediction use standard numerical methods to solve such a system of first-order differential equations (e.g., [2, 32, 33, 51, 89, 103, 104, 114]). Although some of these techniques are reasonably efficient, they are not suitable to *only* compute the peaks in temperature during the transient state, and thus these peaks must be extracted from extensive simulations for many time steps, taking sometimes several seconds to compute. Contrarily, MatEx is based on an *analytical* solution using matrix exponentials and linear algebra, that results in a mathematical expression which can be easily analyzed and differentiated in order to only compute the peaks in transient temperatures. Furthermore, given that MatEx is based on an exact solution which is a function of time, it can also be used to efficiently compute any future transient temperatures without accuracy losses, making it able to potentially replace existing temperature estimation tools.

Open-Source Tools: We implement our algorithms as an open-source tool called MatEx (from matrix exponentials), which is particularly useful for making thermally-safe resource management decisions at runtime. MatEx is available for download at ces.itec.kit.edu/download.

6.1.1 Motivational Example

For simplicity of presentation, consider the 16-core homogeneous system presented in Figure 4.3b (described in Chapter 4.2.1), and a critical threshold temperature for triggering DTM of 80°C. After running simulations with HotSpot, Figure 6.1 illustrates the steady-state temperature distribution of two mappings with different numbers of active cores and different power consumptions. For both cases in Figure 6.1, the maximum steady-state temperature throughout the chip is 80°C, such that DTM would not be triggered in either steady-state scenario.

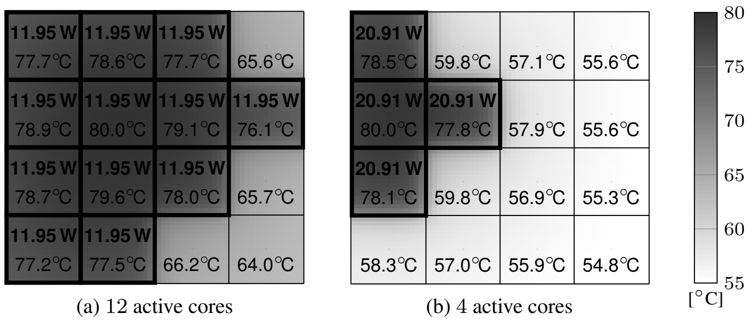


Figure 6.1: (from [73]) Example of two steady-states for the 16-core chip presented in Figure 4.3b (described in Chapter 4.2.1). Active cores are boxed in black. Top numbers represent the power consumptions on active cores. Bottom numbers represent the temperatures on the center of each core. Detailed temperatures are shown according to the color bar.

However, as presented in the transient simulations in Figure 6.2, the temperature on at least one core exceeds 80°C for several seconds when transitioning from the mapping in Figure 6.1a to that in Figure 6.1b, reaching up to $87.51°\mathrm{C}$ after 60 ms of the change in power consumption. Furthermore, according to the numbering of cores in Figure 4.3b, the temperature on core C_5 *raises to* $86.60°C$ *almost instantly*, potentially damaging the chip because DTM may take some time to react and become active. This transient thermal behavior occurs regardless of the fact that the steady-state temperatures for both mappings do not exceed the critical threshold temperature of 80°C. *The reason behind this effect is that the 8 cores that are shut-down at time* $t = 1\,\mathrm{s}$ *require some time to cool down, and thus they transfer heat to the other cores during this time period, while at the same time the 4 cores that remain active are almost duplicating their power density.*

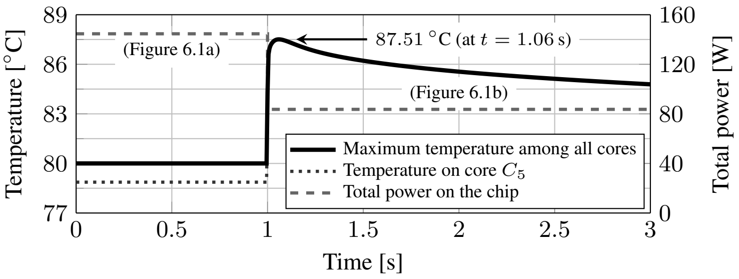


Figure 6.2: Transient temperatures example. During $t = [0\,\mathrm{s}, 1\,\mathrm{s}]$ there are 12 active cores according to Figure 6.1a. During $t = [1\,\mathrm{s}, 3\,\mathrm{s}]$, there are 4 cores active according to Figure 6.1b. The highest transient temperature occurs on core C_5 at time $t = 1.06\,\mathrm{s}$.

6.1.2 Problem Definition

For any compact thermal model that is composed by a system of first-order differential equations relating the temperatures of different areas of the chip with their power consumptions and the ambient temperature, as shown in Equation (3.9), the objective of this chapter is to derive a lightweight and accurate method to efficiently compute the peak in the transient temperature on thermal node k, i.e., $T_k(t)$, after a change in the power consumption of one or more nodes, such that vector $\mathbf{P}$ is the new power vector after the change in power.

$$\mathbf{T}' = \mathbf{C}\mathbf{T} + \mathbf{A}^{-1}\mathbf{P} + T_{\text{amb}}\mathbf{A}^{-1}\mathbf{G} \qquad \text{with } \mathbf{C} = -\mathbf{A}^{-1}\mathbf{B} \tag{3.9 revisited}$$

For such a purpose, we first require an analytical solution for the system of first-order differential equations from Equation (3.9). In other words, we require a mathematical expression from which we can efficiently compute $T_k(t)$ for any thermal node k and time $t \geq 0$. This is presented in Section 6.2.

Following, from the results in Section 6.2, we then need to find the time instant at which $T_k(t)$ is maximized, and we define this time instant as $t^{\uparrow k}$. Finally, we simply compute $T_k\left(t^{\uparrow k}\right)$ in order to obtain the value of the peak in the transient temperature of thermal node k for the given power change. This is presented in Section 6.3.

6.2 Computing All Transient Temperatures

In this section, we find a mathematical expression from which we can efficiently compute $T_k(t)$ for any thermal node k and time $t \geq 0$. Figure 6.3 shows an overview of the different steps required to achieve such a goal.

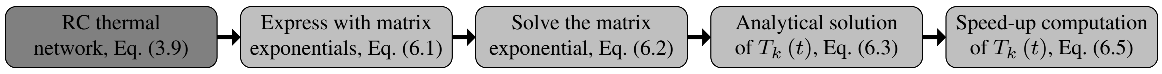

RC thermal network, Eq. (3.9)
Express with matrix exponentials, Eq. (6.1)
Solve the matrix exponential, Eq. (6.2)
Analytical solution of $T_k(t)$, Eq. (6.3)
Speed-up computation of $T_k(t)$, Eq. (6.5)

Figure 6.3: Overview of the steps involved in deriving MatEx to compute all transient temperatures.

It has being well studied in the literature (e.g., [61]), that the system of first-order differential equations from Equation (3.9) can be solved analytically by using matrix exponentials. Therefore, by assuming that when there is a change in power we *reset* the time such that the change in power occurs at time $t = 0$ and the new power values are those of vector $\mathbf{P}$, we can apply the results from [61] in order to rewrite Equation (3.9) as a function of $\mathbf{T}_{\text{init}}$, $\mathbf{T}_{\text{steady}}$, and $\mathbf{C}$, as shown in Equation (6.1).

$$\mathbf{T} = \mathbf{T}_{\text{steady}} + e^{\mathbf{C}t}\left(\mathbf{T}_{\text{init}} - \mathbf{T}_{\text{steady}}\right) \tag{6.1}$$

In Equation (6.1), $e^{\mathbf{C}t} = \left[{e^{\mathbf{C}t}}_{i,j}\right]_{N\times N}$ is defined as a matrix exponential. As explained in Chapter 3.5, matrix $\mathbf{B}$ (used to compute the steady-state temperatures) and matrix $\mathbf{C}$ are related to the hardware and cooling solution, such that they are constant for a given thermal model. Therefore, the variables in Equation (6.1) are time t, the initial temperatures $\mathbf{T}_{\text{init}}$, and the steady-state temperatures $\mathbf{T}_{\text{steady}}$ (which in turn depend on the new power consumptions $\mathbf{P}$ and the ambient temperature T_{amb}). *Note that Equation (6.1) does not imply that the temperature on a thermal node approaches its steady-state temperature exponentially, given that $e^{\mathbf{C}t}$ is a matrix exponential (i.e., its solution will also be a matrix) and not a regular exponential function.*

There are several methods that can be used to solve the matrix exponential $e^{\mathbf{C}t}$, including numerical methods that solve it for a certain time t, and analytical methods that are based on linear algebra in which t remains a variable. Focusing on the latter, the work in [61] presents a solution to the matrix exponential, particularly,

$$e^{\mathbf{C}t} = \mathbf{\Gamma}\mathbf{\Lambda}\mathbf{\Gamma}^{-1},$$

where matrix $\mathbf{\Gamma} = [\Gamma_{i,j}]_{N\times N}$ represents a matrix containing the eigenvectors of matrix $\mathbf{C}$, matrix $\mathbf{\Gamma}^{-1} = [\tilde{\Gamma}_{i,j}]_{N\times N}$ is the inverse of matrix $\mathbf{\Gamma}$, and matrix $\mathbf{\Lambda}$ is a diagonal matrix such that

$$\mathbf{\Lambda} = \begin{bmatrix} e^{\lambda_1\cdot t} & 0 & \cdots & 0 \\ 0 & e^{\lambda_2\cdot t} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & e^{\lambda_N\cdot t} \end{bmatrix},$$

where $\lambda_1, \lambda_2, \ldots, \lambda_N$ are the eigenvalues of matrix $\mathbf{C}$. The eigenvalues and eigenvectors of matrix $\mathbf{C}$ can be computed by applying linear algebra (e.g., using the Eigen C++ template library [100]), with total time complexity $O\left(N^3\right)$. Given that this a physical and stable system in which all temperatures eventually reach their steady-state values, it holds that $\lambda_i \leq 0$ for all $i = 1, 2, \ldots, N$. *Note that for a given matrix $\mathbf{C}$ (i.e., for a given thermal model associated to a given chip and cooling solution), the eigenvalues, matrix $\mathbf{\Gamma}$ and its inverse $\mathbf{\Gamma}^{-1}$, and matrix $\mathbf{\Lambda}$, only need to be computed once and can be stored in memory for later use.* Hence, the matrix exponential can be solved by computing every ${e^{\mathbf{C}t}}_{k,j}$ inside $e^{\mathbf{C}t}$ as shown in Equation (6.2).

$${e^{\mathbf{C}t}}_{k,j} = \sum_{i=1}^{N} \Gamma_{k,i} \cdot \tilde{\Gamma}_{i,j} \cdot e^{\lambda_i\cdot t} \tag{6.2}$$

Therefore, we can apply the results for matrix exponentials from [61] that are shown in Equation (6.2) to the expression in Equation (6.1), such that we can compute the transient temperature on thermal node k as a function of time, i.e., $T_k(t)$, as shown in Equation (6.3).

$$T_k(t) = T_{\text{steady}_k} + \sum_{i=1}^{N} e^{\lambda_i \cdot t} \cdot \Gamma_{k,i} \cdot \sum_{j=1}^{N} \tilde{\Gamma}_{i,j} \left(T_{\text{init}_j} - T_{\text{steady}_j} \right) \tag{6.3}$$

Note that in Equation (6.3), we consider that time $t = 0$ corresponds to the time point at which a change in power takes place, such that $\mathbf{P}$ is the new power vector. The steady-state temperatures $\mathbf{T}_{\text{steady}}$, which can be computed using Equation (3.10) for each node k, are the temperatures to which every thermal node will eventually converge considering this new power vector.

$$T_{\text{steady}_k} = \sum_{j=1}^{N} \tilde{b}_{k,j} \cdot p_j + T_{\text{amb}} \cdot \sum_{j=1}^{N} \tilde{b}_{k,j} \cdot g_j \tag{3.10 revisited}$$

The initial temperatures on all thermal nodes right before the change in power are represented by vector $\mathbf{T}_{\text{init}}$. Therefore, for a given change in power and power vector $\mathbf{P}$, the only variable in Equation (6.3) is t. Figure 6.4 illustrates the physical interpretation of the symbols in Equation (6.3) through an abstract example of the transient temperature on thermal node k.

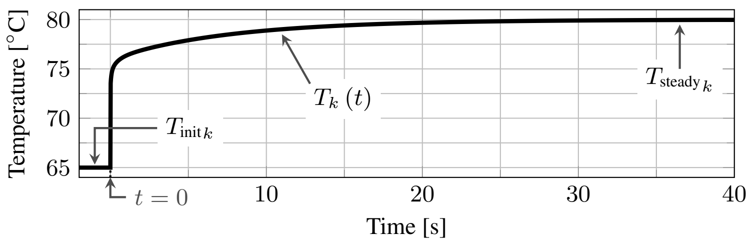


Figure 6.4: Physical interpretation of the symbols in Equation (6.3) for computing the transient temperature of thermal node k. In Equation (6.3), time $t = 0$ corresponds to the moment at which a change in power occurs.

Furthermore, considering that t is the only variable in Equation (6.3), the computation of $T_k(t)$ could be achieved more efficiently by first building auxiliary matrix $\mathbf{\Omega} = [\Omega_{k,i}]_{N \times N}$, such that

$$\Omega_{k,i} = \Gamma_{k,i} \cdot \sum_{j=1}^{N} \tilde{\Gamma}_{i,j} \left(T_{\text{init}_j} - T_{\text{steady}_j} \right), \tag{6.4}$$

for all $k = 1, 2, \ldots, N$ and for all $i = 1, 2, \ldots, N$. Finally, for a given ambient temperature, initial temperatures, and power change which would eventually converge to new steady-state temperatures, the temperature on node k at time t can be computed as shown in Equation (6.5).

$$T_k(t) = T_{\text{steady}_k} + \sum_{i=1}^{N} \Omega_{k,i} \cdot e^{\lambda_i \cdot t} \tag{6.5}$$

Note that, unlike the eigenvalues and eigenvectors which only need to be computed once for a given thermal model (i.e., for a given chip and cooling solution), *vector $\mathbf{T}_{steady}$ and matrix $\mathbf{\Omega}$ are computed/build once for every change in power*, with total time complexity $O(N^2)$. Therefore, for a given time t and considering that matrix $\mathbf{\Omega}$ is already built, the total time complexity for computing $T_k(t)$ for thermal node k is $O(N)$, and $O(N^2)$ when computing $T_k(t)$ for all thermal nodes $k = 1, 2, \ldots, N$.

Finally, there are additional *implementation improvements* that may reduce the execution time (not the complexity) for computing $T_k(t)$. For example, when building matrix $\mathbf{\Omega}$ for every change in power, we can compute an auxiliary vector $\mathbf{T}_{\text{diff}} = \mathbf{T}_{\text{init}} - \mathbf{T}_{\text{steady}}$, therefore allowing subtraction $T_{\text{init}_j} - T_{\text{steady}_j}$ to be computed once for every node (N times in total for all nodes), instead of N^3 times in total for all nodes as suggested by Equation (6.4) when computing $\Omega_{k,i}$ for all $k = 1, 2, \ldots, N$ and for all $i = 1, 2, \ldots, N$.

Similarly, for a given time t, we can compute an auxiliary vector $\left\{e^{\lambda_1 \cdot t}, e^{\lambda_2 \cdot t}, \ldots, e^{\lambda_N \cdot t}\right\}$, which would allow this exponentiation to be computed only once for every node (N times in total for all nodes), instead of N times for each node k (N^2 times in total for all nodes) as suggested by Equation (6.5). A pseudo-code for building auxiliary matrix $\mathbf{\Omega}$ is presented in Algorithm 3, and the corresponding pseudo-code to compute $T_k(t)$ for every node k is presented in Algorithm 4.

Algorithm 3 Build auxiliary matrix $\mathbf{\Omega}$

```
Input: Matrix $\mathbf{\Gamma}$ and its inverse $\mathbf{\Gamma}^{-1}$, vector $\mathbf{T}_{\text{init}}$, and vector $\mathbf{T}_{\text{steady}}$;
Output: Auxiliary matrix $\mathbf{\Omega}$;
1:  for all $j = 1, 2, \ldots, N$ (i.e., for all thermal nodes) do
2:      $T_{\text{diff}_j} \leftarrow T_{\text{init}_j} - T_{\text{steady}_j}$;
3:  end for
4:  for all $i = 1, 2, \ldots, N$ (i.e., for all thermal nodes) do
5:      $\text{aux}\Omega_i \leftarrow \sum_{j=1}^{N} \tilde{\Gamma}_{i,j} \cdot T_{\text{diff}_j}$;
6:  end for
7:  for all $k = 1, 2, \ldots, N$ (i.e., for all thermal nodes) do
8:      for all $i = 1, 2, \ldots, N$ (i.e., for all thermal nodes) do
9:          $\Omega_{k,i} \leftarrow \Gamma_{k,i} \cdot \text{aux}\Omega_i$;
10:     end for
11: end for
12: return Auxiliary matrix $\mathbf{\Omega}$;
```

Algorithm 4 Compute the transient temperatures of all thermal nodes at time t

```
Input: Auxiliary matrix $\mathbf{\Omega}$ for the current change in power, the eigenvalues, and time of interest $t$;
Output: Temperatures $T_k(t)$ for all thermal nodes $k = 1, 2, \ldots, N$;
1:  for all $i = 1, 2, \ldots, N$ (i.e., for all thermal nodes) do
2:      $\text{auxExp}_i \leftarrow e^{\lambda_i \cdot t}$;
3:  end for
4:  for all $k = 1, 2, \ldots, N$ (i.e., for all thermal nodes) do
5:      $T_k(t) \leftarrow T_{\text{steady}_k} + \sum_{i=1}^{N} \text{auxExp}_i \cdot \Omega_{k,i}$;
6:  end for
7:  return $T_k(t)$ for all $k = 1, 2, \ldots, N$;
```

6.3 Computing Peaks in Transient Temperatures

In this section we find the time instant at which $T_k(t)$ is maximized, defined as $t^{\uparrow k}$, such that we can compute $T_k\left(t^{\uparrow k}\right)$ to obtain the value of the peak in the transient temperature of thermal node k for the given power change. Given that the value of $T_k(t)$ depends on power vector $\mathbf{P}$, time instant $t^{\uparrow k}$ for every thermal node k needs to be computed once for every change in power. Figure 6.5 shows an overview of the different steps required to achieve such a goal.

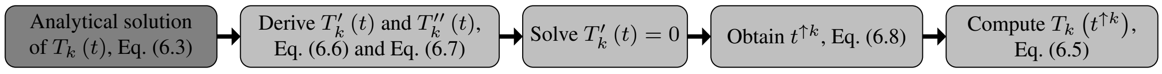

Analytical solution of $T_k(t)$, Eq. (6.3)
Derive $T'_k(t)$ and $T''_k(t)$, Eq. (6.6) and Eq. (6.7)
Solve $T'_k(t) = 0$
Obtain $t^{\uparrow k}$, Eq. (6.8)
Compute $T_k\left(t^{\uparrow k}\right)$, Eq. (6.5)

Figure 6.5: Overview of the steps involved in deriving MatEx to compute the peaks in transient temperatures.

It should be noted that since $T_k(t)$ results in a different expression for every thermal node k, the value of $t^{\uparrow k}$ will be different for every node. For example, Figure 6.6 shows the transient temperatures on cores C_2 and C_5 for the transient simulations already presented in Figure 6.2, where we can clearly observe the difference in the transient temperatures on each core.

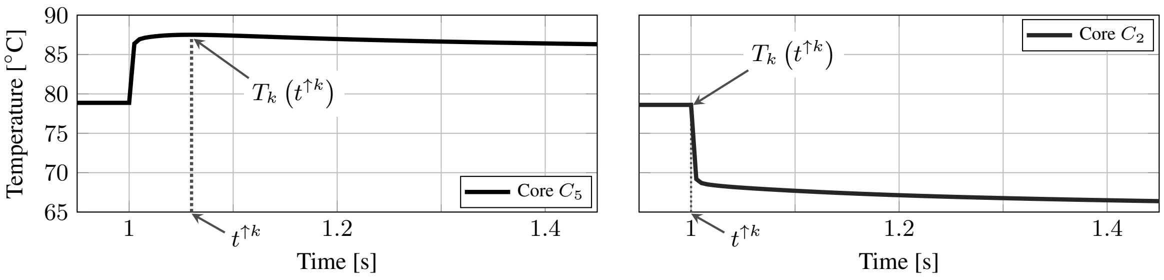


Figure 6.6: Example showing that, since $T_k\left(t\right)$ results in a different expression for every thermal node, the value of $t^{\uparrow k}$ is also different for every node k.

From Equation (6.3) and Equation (6.5), we know that $T_k\left(t\right)$ can be computed as a summation of decaying exponential functions, and therefore $T_k\left(t\right)$ is continuous and differentiable with respect to t for the case of $t \geq 0$. Therefore, we can derive the first-order derivative of $T_k\left(t\right)$ with respect to time, defined as $T'_k\left(t\right)$, which starting from Equation (6.5) can be expressed as shown in Equation (6.6).

$$T'_k\left(t\right) = \sum_{i=1}^{N} \lambda_i \cdot \Omega_{k,i} \cdot e^{\lambda_i \cdot t} \tag{6.6}$$

For example, Figure 6.7 illustrates the first-order derivative of the temperature on core C_5 with respect to time, for the simulations already presented in Figure 6.2.

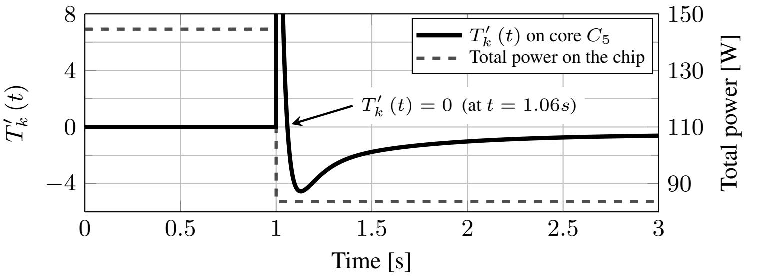


Figure 6.7: First-order derivative of the temperature on core C_5 for the transient simulations in Figure 6.2. After the change in power, $T'_k\left(t\right)$ is zero at $t = 1.06$ s and when $t \to \infty$ (i.e., when the temperature reaches its steady-state value).

As seen in Figure 6.7, when there is a peak in the temperature of thermal node k, the slope of $T_k\left(t\right)$ changes sign and the value of $T'_k\left(t\right)$ is equal to zero. Therefore, one possible method to find a transient peak on thermal node k is to solve $T'_k\left(t\right) = 0$ for $t \geq 0$. Given that $T_k\left(t\right)$ would eventually reach its steady-state temperature T_{steady_k} (if the power remains constant for a long enough time), there is always at least one solution for $T'_k\left(t\right) = 0$ when $t \to \infty$. Furthermore, there can also exist several other solutions for $T'_k\left(t\right) = 0$, and every solution in which $t \geq 0$ is a potential peak in the temperature on thermal node k. A naive approach would attempt to find all the solutions for $T'_k\left(t\right) = 0$, and then test the one that results in the highest temperature. However, given that $T_k\left(t\right)$ is a summation of decaying exponential functions, from control theory (e.g., [55]) we know that, for any stable system in which all poles are less than or equal to zero, the maximum temperature (or overshoot in control theory [55]) always occurs at the first peak. In order to be certain that this is the case, we apply the Laplace transform to Equation (6.5), such that

$$T_k\left(S\right) = \frac{T_{\text{steady}_k}}{S} + \sum_{i=1}^{N} \frac{\Omega_{k,i}}{S - \lambda_i}$$

where $T_k\left(S\right)$ is the Laplace transform of $T_k\left(t\right)$. Given that, as mentioned in Section 6.2, this a physical and stable system in which all temperatures eventually reach their steady-state values, we have that $\lambda_i \leq 0$ for all $i = 1, 2, \ldots, N$, such that all poles in $T_k\left(S\right)$ are less than or equal to zero, and therefore the above conclusion holds. In this way, the only solution of interest for $T'_k\left(t\right) = 0$ is the closest solution to zero such that $t \geq 0$, and this solution is time instant $t^{\uparrow k}$.

Unfortunately, there is no closed analytical form to directly solve $T'_k(t) = 0$, and therefore we need to apply a numerical method for such a purpose. One approach for solving such an expression would be to apply the Newton-Raphson method [7], for which we need the second-order derivative of $T_k(t)$, defined as $T''_k(t)$, expressed in Equation (6.7).

$$T''_k(t) = \sum_{i=1}^{N} {\lambda_i}^2 \cdot \Omega_{k,i} \cdot e^{\lambda_i \cdot t} \tag{6.7}$$

Then, starting from an initial guess for $t^{\uparrow k}$, which we define as $t_0^{\uparrow k}$, the value of $t^{\uparrow k}$ can be iteratively approximated. The value of $t^{\uparrow k}$ for the n-th iteration, defined as $t_n^{\uparrow k}$, is computed from the value of the previous iteration $t_{n-1}^{\uparrow k}$. Specifically, the value of $t_n^{\uparrow k}$ is computed as shown in Equation (6.8), with time complexity $O(N)$ for every iteration of a thermal node k.

$$t_n^{\uparrow k} = t_{n-1}^{\uparrow k} - \frac{T'_k\left(t_{n-1}^{\uparrow k}\right)}{T''_k\left(t_{n-1}^{\uparrow k}\right)} = t_{n-1}^{\uparrow k} - \frac{\sum_{i=1}^{N} \lambda_i \cdot \Omega_{k,i} \cdot e^{\lambda_i \cdot t_{n-1}^{\uparrow k}}}{\sum_{i=1}^{N} {\lambda_i}^2 \cdot \Omega_{k,i} \cdot e^{\lambda_i \cdot t_{n-1}^{\uparrow k}}} \tag{6.8}$$

In order for $t_n^{\uparrow k}$ to converge to the *first* transient peak with $t \geq 0$, we use $t_0^{\uparrow k} = 0$ as initial guess. Finally, time $t^{\uparrow k}$ is set to $t_W^{\uparrow k}$, where W is a constant indicating the total number of iterations used when applying the Newton-Raphson method, and then $T_k\left(t^{\uparrow k}\right)$ can be computed through Algorithm 4. A pseudo-code to compute the peak temperatures on all nodes is presented in Algorithm 5. Figure 6.8 shows an example of the Newton-Raphson method for the first-order derivative of the temperature on core C_5 shown in Figure 6.7.

Algorithm 5 Peak temperature computation for all thermal nodes

```
Input: Auxiliary matrix $\mathbf{\Omega}$, and total number of iterations $W$ for the Newton-Raphson method;
Output: Peak temperatures $T_k\left(t^{\uparrow k}\right)$ for all $k = 1, 2, \ldots, N$ (i.e., for all thermal nodes);
1: for all $k = 1, 2, \ldots, N$ (i.e., for all thermal nodes) do
2:     $t^{\uparrow k} \leftarrow 0$; {Set the value of the initial guess of $t^{\uparrow k}$}
3:     for all $n = 1, 2, \ldots, W$ (i.e., for all iterations of the Newton-Raphson method) do
4:         $t^{\uparrow k} \leftarrow t^{\uparrow k} - \frac{\sum_{i=1}^{N} \lambda_i \cdot \Omega_{k,i} \cdot e^{\lambda_i \cdot t^{\uparrow k}}}{\sum_{i=1}^{N} {\lambda_i}^2 \cdot \Omega_{k,i} \cdot e^{\lambda_i \cdot t^{\uparrow k}}}$; {Compute $t^{\uparrow k}$ for the $n$-th iteration from the previous iteration}
5:     end for
6: end for
7: return $T_k\left(t^{\uparrow k}\right)$ for all $k = 1, 2, \ldots, N$ from Algorithm 4;
```

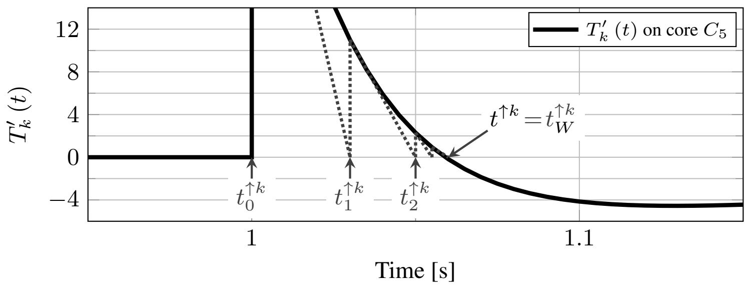


Figure 6.8: Conceptual example of the Newton-Raphson method for solving $T'_k(t) = 0$ on core C_5 in W iterations (only a few shown for simplicity of presentation), for the transient simulations in Figure 6.2. After the change in power, the first solution to $T'_k(t) = 0$ is found at time $t = 1.06$ s.

There is the possibility that $T_k(t)$ is a strictly decreasing or strictly increasing function, such that there is no peak in $T_k(t)$ and its maximum value occurs at $t = 0$ or when $t \to \infty$, respectively, with value $T_k(0)$ or T_{steady_k}, respectively. For example, this is the case for core C_2 in Figure 6.6. Therefore, the actual maximum temperature to consider is

$$\max\left\{T_k\left(t^{\uparrow k}\right), T_k(0), T_{\text{steady}_k}\right\}.$$

Furthermore, if there is a new change in power at time t_{change}, the temperature on thermal node k might never reach the value of T_{steady_k}. For such cases, the maximum temperature to consider is

$$\max\left\{T_k\left(t^{\uparrow k}\right), T_k\left(0\right), T_k\left(t_{\text{change}}\right)\right\},$$

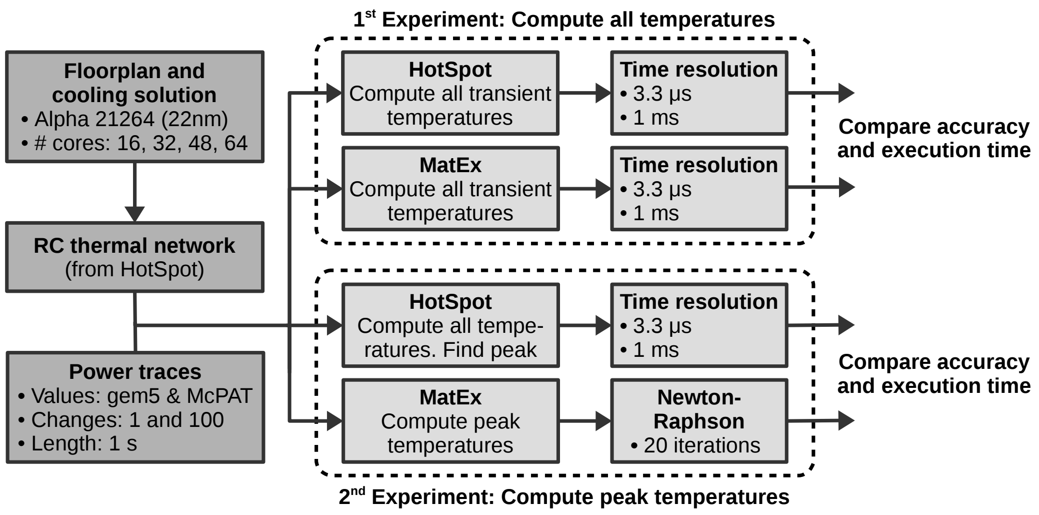
and $T_{\text{init}k}$ for the next computation can be set to $T_k\left(t_{\text{change}}\right)$.

6.4 Experimental Evaluations

In this section, we evaluate the accuracy and the efficiency (in terms of computation time) of MatEx, compared to the widely-adopted HotSpot [33].

6.4.1 Setup

For this particular experimental evaluations, we only use parts of the simulation framework described in Chapter 4. Figure 6.9 presents an overview of the simulation flow used for evaluating the accuracy and execution time of both HotSpot and MatEx. Specifically, we again use HotSpot to derive the compact thermal model (i.e., the values of matrix $\mathbf{A}$, matrix $\mathbf{B}$, and vector $\mathbf{G}$), and this thermal model is used both by HotSpot and MatEx as an input. Note that since MatEx is not tied to HotSpot, *any other* compact modeling tool could have been used to derive the thermal model, and we merely use HotSpot due to its popularity. With respect to the floorplan and the number of cores in the chip, since the size of the thermal model affects the complexity and execution time of the algorithms under evaluation, we consider four different cases in which the chip has 16 cores, 32 cores, 48 cores, and 64 cores. For these numbers of cores, HotSpot generates RC thermal networks with 76 nodes, 140 nodes, 204 nodes, and 268 nodes, respectively.



1st Experiment: Compute all temperatures
Floorplan and cooling solution
• Alpha 21264 (22nm)
• # cores: 16, 32, 48, 64
RC thermal network (from HotSpot)
Power traces
• Values: gem5 & McPAT
• Changes: 1 and 100
• Length: 1 s
HotSpot Compute all transient temperatures
Time resolution • 3.3 µs • 1 ms
MatEx Compute all transient temperatures
Time resolution • 3.3 µs • 1 ms
Compare accuracy and execution time
HotSpot Compute all temperatures. Find peak
Time resolution • 3.3 µs • 1 ms
MatEx Compute peak temperatures
Newton-Raphson • 20 iterations
Compare accuracy and execution time
2nd Experiment: Compute peak temperatures

Figure 6.9: Overview of the simulation flow used for evaluating the accuracy and execution time of HotSpot and MatEx.

In MatEx, Algorithm 4 is used to compute all transient temperatures at a given time t, while Algorithm 5 is used to compute the peaks in the transient temperatures generated by the changes of power inside the chip. Therefore, we conduct two separate experiments, one experiment to compare each algorithm to HotSpot. In the first experiment, when computing the transient temperatures using HotSpot, it is required to set a time resolution (or time step) for the transient simulations, i.e., the period of time elapsed between each computed temperature. Hence, when we evaluate Algorithm 4 of MatEx to compute specific transient temperatures, we use equivalent time resolutions as used in HotSpot to have comparable results. Specifically, we consider two resolutions: $3.333\,\mu$s and $1\,$ms. In the second experiment, given that HotSpot cannot *only* compute the peaks in the transient temperatures, we assume that it considers the same two time resolutions as in the first experiments, and extracts the maximum values after the complete simulation finished. Contrarily, when evaluating Algorithm 5 of MatEx, we are able to only compute the transient peaks, for which we consider 20 iterations for the Newton-Rhapson method.

Given that MatEx needs to build auxiliary matrix $\mathbf{\Omega}$ for every change in power, a higher number of power changes in the input power traces would result in longer execution times for MatEx. Therefore, for the power traces required as inputs by both HotSpot and MatEx, we intentionally generate synthetic traces using randomly selected power values from simulations using gem5 and McPAT for applications from the PARSEC benchmark suite, but do not directly considering the detailed power values from specific application executions. The reason behind this consideration is to have a controlled number of power changes in each power trace, such that we can evaluate how the frequency of the power changes affects the execution time of MatEx. Were we to directly use power traces from real applications, these effects would be harder to observe since the number of power changes in the input trace would be imposed by the application and architecture. In order to account for two different cases, the power traces are generated considering 1 power change and 100 power changes, and in all cases the traces have a length of 1 s.

6.4.2 Results

Figure 6.10 presents the experimental results comparing the accuracy of both HotSpot and MatEx, for the case of 64 cores and a single change in power (other results are omitted because equivalent observations apply to all other cases). In the figure, we can see that aside from some minor jitter in HotSpot's transient output when the time resolution is 1 ms, both HotSpot and MatEx produce the same results (within a margin of 0.01°C). Furthermore, *the accuracy of MatEx is entirely independent on the time resolution used for computing the transient temperatures*, and this comes from the fact that MatEx can compute a future transient temperature through Equation (6.5) for any given time t, and the time of the previous temperature computation is irrelevant. Therefore, when using MatEx to run full transient simulations, *the user could adapt the time resolution for any specific needs, without additional considerations about accuracy losses*.

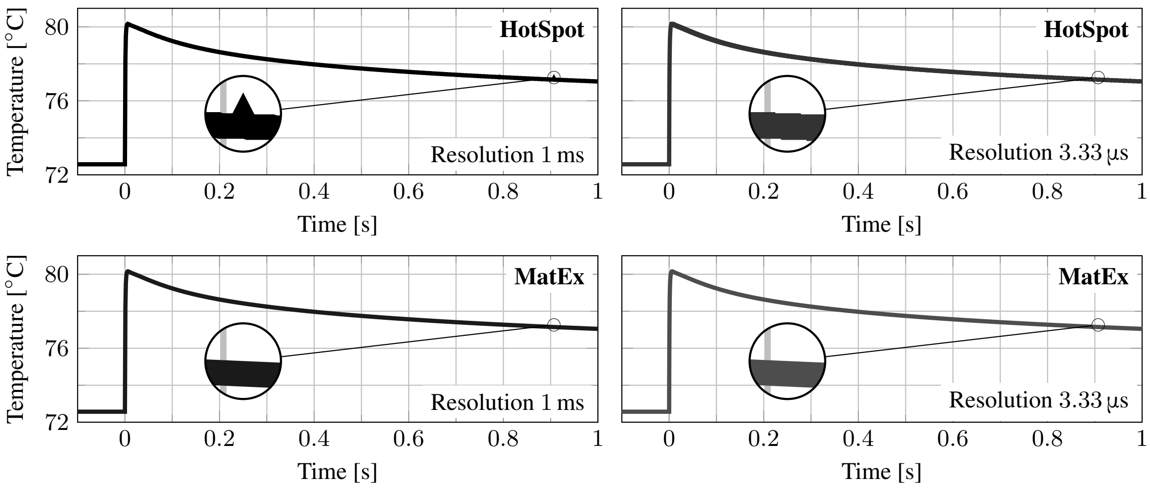


Figure 6.10: Simulation results for all transient temperatures used to compare the accuracy of HotSpot and MatEx, for the case of 64 cores and a single change in power.

Figure 6.11 and Figure 6.12 present the execution time results of HotSpot and MatEx (Algorithm 3 and Algorithm 4), respectively, for the experiment that computes all transient temperatures, for all the cases described in Section 6.4.1. For both HotSpot and MatEx, a higher time resolution results in longer execution times. Moreover, in HotSpot, the execution time is only affected by the time resolution, and not by the number of power changes, as this is not relevant for a numerical method. In MatEx, aside from the effect of the time resolution, the number of changes in power could also increase the execution time. In our experiment, for a resolution of 1 ms, Algorithm 4 is executed 1000 times (since the power trace lasts 1 s and we compute all transient temperatures every 1 ms), while Algorithm 3 is executed 100 times (for the traces with 100 power changes) or only once (for the traces with 1 power change). Hence, given that Algorithm 3 is executed just 10 times less or 1000 times less than Algorithm 4, the effect of the number of power changes is noticeable in the total execution time of MatEx for a time resolution of 1 ms. However, for a resolution of 3.333 µs, Algorithm 4 is executed $3 \cdot 10^5$ times, while Algorithm 3 is executed the same number of times as before (given that we have the same number of power changes in the trace), thus masking the execution time of Algorithm 3. The most important fact to highlight is that *MatEx is always faster than HotSpot for the same time resolution*. Particularly, for all the evaluated cases, *MatEx is on average 40 times faster than HotSpot, being up to 100 times faster for the case with* 64 *cores, one power change, and a time resolution of* 1 *ms*.

Therefore, due to its high efficiency and accuracy, aside for using MatEx to conduct peak temperature estimations, these results enable MatEx also to replace HotSpot for general transient temperature computations.

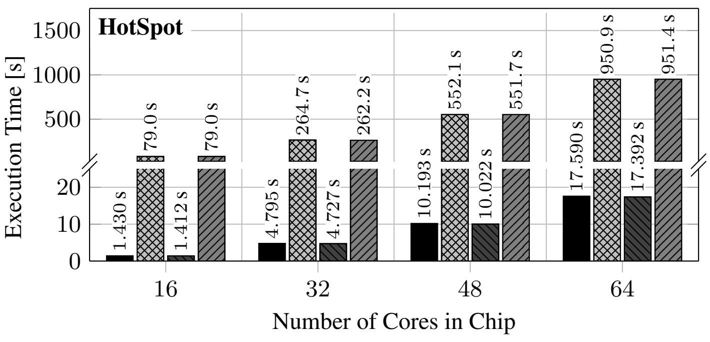


Figure 6.11: Execution time for computing *all* transient temperatures and the *peaks* in the transient temperatures when *using HotSpot*.

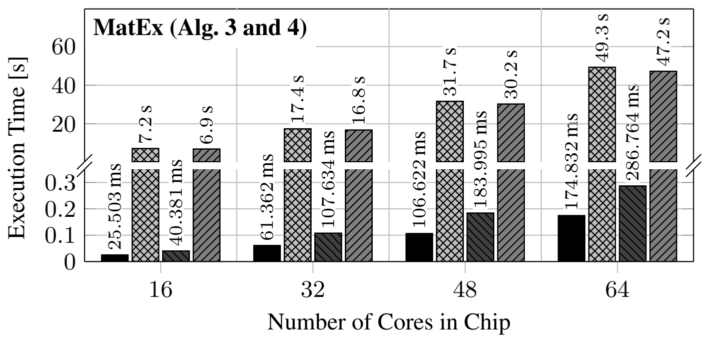


Figure 6.12: Execution time for computing *all* transient temperatures when *using MatEx* (Algorithm 3 and Algorithm 4).

Figure 6.13 presents the execution time results of MatEx (Algorithm 3 and Algorithm 5) for the second experiment that only focuses on computing the peaks in the transient temperatures, for all the cases described in Section 6.4.1. With regards to HotSpot (or any other numerical method), since this type of temperature estimation tool is unable to only analyze the peaks in temperature without simulating the entire trace, hence the execution time of HotSpot for this experiment is the same as when computing all transient temperatures and these values are already presented in Figure 6.11 (for this experiment HotSpot was trivially modified in order to keep track of the highest transient temperature, incurring in negligible overheads). Hence, from Figure 6.11, we can see that for these experimental settings HotSpot needs at least 1.4 s to compute the peaks in transient temperatures, which is not suitable for runtime management decisions. On the other hand, MatEx (Algorithm 5) uses the Newton-Raphson method to compute the time instants of the peaks in temperature, and therefore, for every change in power, it only needs to compute the temperature on a few points, significantly reducing the execution time. Particularly, for a system with 16 cores, Figure 6.13 shows that the execution time of MatEx when having *one power change is only up to* $2.5\,ms$, which might be suited for *runtime* management decisions. Furthermore, given that Algorithm 3 and Algorithm 5 have independent computations for every thermal node, MatEx could be easily implemented as a multi-threaded application (e.g., one thread for each node), and thus parallelized into multiple cores, further reducing the execution time. Finally, the results also show that the number of power changes has a linear impact in the execution time of MatEx for peak temperature computation. Therefore, the execution time of MatEx takes around 100 times longer when we have 100 power changes than when having a single power change.

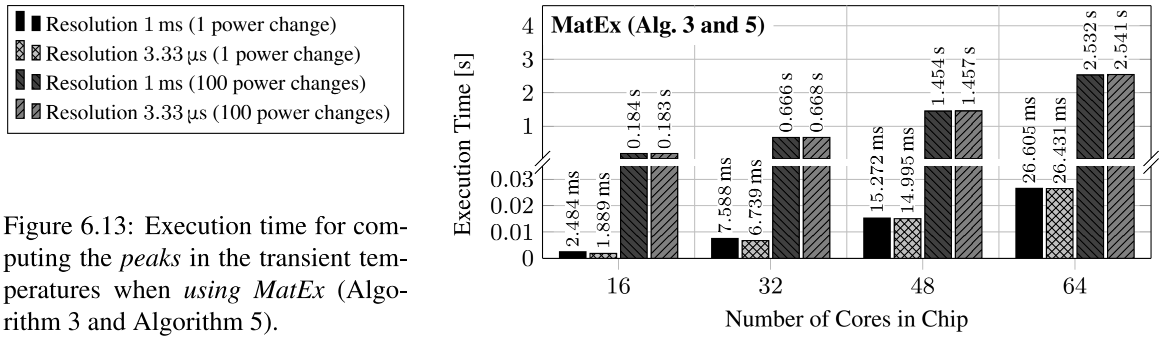


Figure 6.13: Execution time for computing the *peaks* in the transient temperatures when *using MatEx* (Algorithm 3 and Algorithm 5).

6.5 Summary

In this chapter we have shown that management decisions typically used to optimize resource usages (e.g., task migration, DPM, DVFS, etc.) can result in transient temperatures that are much higher than their associated steady-state temperatures. Therefore, given that the system could benefit from evaluating such transient peaks while making resource management decisions in order to operate in thermally safe ranges and have a predictable behavior, in this chapter we presented a lightweight and accurate method for estimating/predicting transient thermal peaks, as well as computing any future transient temperatures without requiring incremental computations. In contrast to traditional numerical methods for model-based temperature estimation, this new method, called MatEx (implemented as an open-source tool), is based on matrix exponentials and linear algebra, which allows us to derive an analytical expression that can be used for computing the transient temperatures, as well as analyzing and differentiating such an expression to obtain its maximum values.

Our experimental evaluations compared the efficiency and accuracy of MatEx with that of the widely-used temperature simulator HotSpot. With respect to the computation of future transient temperatures, our results show that the execution time of MatEx is in average 40 times faster than that of HotSpot, and up to 100 times faster in a few cases, which makes MatEx a promising candidate to replace existing temperature estimation tools. More importantly, for the computation of the peaks in temperature, the execution time of MatEx under the given experimental settings is just a few milliseconds for every power change. These results suggest that MatEx can be used for runtime estimation of transient thermal peaks when making resource management decisions, thus helping the system to prevent undesired triggers of DTM or damages to the chip when DTM would be unable to react fast enough.

Finally, MatEx can be used to assist the TSP power budgeting technique (presented in Chapter 5) in order to quantify the maximum values that the transient peak temperatures can reach during the transient state, as discussed in Chapter 5.4.1. Furthermore, MatEx is also the foundation of the runtime boosting technique based on transient temperature estimation presented in Chapter 7.

Chapter 7

Selective Boosting for Multicore Systems (seBoost)

7.1 Overview

Whenever a set of applications is executed on a multicore or manycore system, it is common that at some moment one or more application threads require to increase their performance during some time, mostly due to performance/timing requirements surges/peaks at runtime. For example, this could occur in a video face recognition application when suddenly a crowd of people enter the frame [113]. Figure 7.1 shows an abstract example of such a case with three applications running at nominal performance, where one application needs to increase its performance at runtime during some time interval. Moreover, several applications could require to increase their performance concurrently, but such runtime requirements, although overlapping, might arrive at different times or might have different durations.

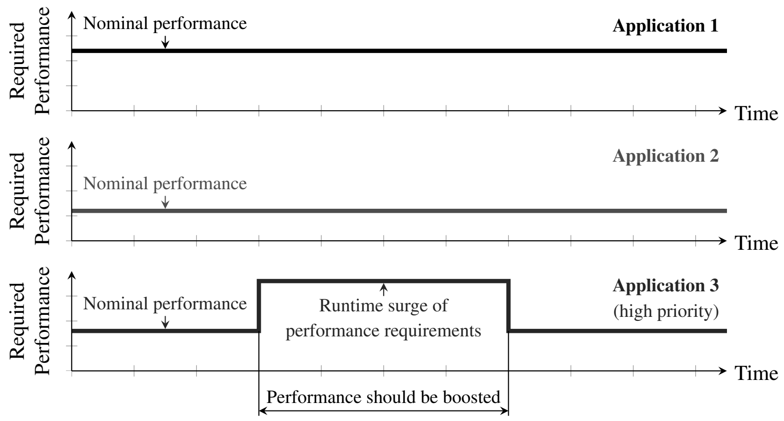


Figure 7.1: Abstract example of a performance requirement surge/peak at runtime. At the beginning, all three applications are executing at the required nominal performance. Then, application 3 (which is a high priority application) needs to increment its performance during some time in order to meet its timing requirements due to a drastic increase in its workload.

Boosting techniques provide the system with the means to satisfy such runtime performance/timing requirements surges, and have therefore been widely adopted in commercial multicore and manycore systems. For example, through DVFS, boosting techniques (e.g., Intel's Turbo Boost [9,10,35,91], described in Chapter 2.1.2, and AMD's Turbo CORE [66]) allow the system to execute some cores in the chip at high DVFS levels during short time intervals, even if this implies exceeding standard operating power budgets (e.g., TDP). Given that running cores at high DVFS levels increases their power consumption, boosting will normally result in an increment of the chip's temperature through time. Due to this temperature increase, if the temperature somewhere in the chip reaches a predefined critical threshold value, the system must either return to nominal operation (requiring some cool-down time before another boosting interval is allowed), or use some closed-loop control-based strategy in order to oscillate around the threshold temperature (prolonging the boosting time).

In order to have high overall performance, careful decisions must be taken when selecting the boosting levels. Particularly, when one or more applications identify a runtime boosting requirement, normally the applications that do not require boosting at this time can be momentarily considered to have a lower priority. This implies that the cores of these low-priority applications could be momentarily throttled-down, i.e., their DVFS levels could be reduced, such that their power consumption decreases and therefore the duration of the boosting intervals might be prolonged at the required performance levels. Failing to appropriately throttle-down the cores of such non-boosted applications can potentially result in failing to satisfy the runtime performance requirements surges of the high priority applications during the required time, mainly due to a rapid raise in the temperature. Contrarily, excessive throttling-down will result in unnecessary overall performance penalties to the system, especially for the non-boosted applications.

Therefore, in this chapter we present an efficient and lightweight runtime boosting technique based on transient temperature estimation (specifically, based on MatEx, presented in Chapter 6), called seBoost [80] (from Selective Boosting). This technique guarantees that the runtime performance requirements surges are met, by executing the cores requiring boosting at the desired DVFS levels for the entire boosting intervals, while throttling-down the non-boosted cores with minimum performance losses for the lower priority applications/threads. In order to minimize such performance losses, the throttling-down levels of the non-boosted cores are chosen such that the maximum temperature throughout the chip reaches the critical threshold value at the precise time in which the boosting requirements are expected to expire. Furthermore, seBoost is also capable of refining the boosting decisions when, e.g., new applications/threads receive additional (concurrent) boosting requirements in the middle of a boosting interval, or when some application/thread finishes its boosted execution but other applications/threads need to continue operating in boosting mode.

7.1.1 Motivational Example

This section presents a motivational example that provides insight into the impact of boosting on both temperature and performance. For simplicity in presentation, consider the 16-core homogeneous system presented in Figure 4.3b (described in Chapter 4.2.1), and a critical threshold temperature for triggering DTM of 80°C. Assume that the system is executing two applications from the PARSEC benchmark suite [4], specifically, an *x264* and a *bodytrack* application, each one running 8 parallel dependent threads (one thread mapped to each core), such that the nominal performance requirements of both applications are satisfied when all cores run at 3 GHz. For such a case, we conduct simulations using our simulation framework in detailed mode (described in Chapter 4), in order to obtain results for performance, power, and temperature.

Following, assume that after 0.1 s of starting its execution, the *bodytrack* application needs to boost its performance, specifically, all 8 cores running this application need to be boosted to 3.7 GHz. The precise duration of the required boosting time is not exactly known; however, based on historical data, it is expected to last no more than 0.25 s. Figure 7.2 presents the simulation results for four different boosting methods, where we can see the maximum temperature throughout the chip and the performance of each application as a function of time. Among these boosting methods, method A simply decides to fully throttle-down the lower-priority *x264* application to the lowest available DVFS levels. Therefore, the cores executing the *bodytrack* application are able to be boosted during 0.25 s at the required frequency, while the *x264* application is only able to achieve 3% of its nominal performance. Boosting method B on the other hand, simply decides to keep running the cores executing the *x264* application at its nominal values. Hence, although now the *x264* application does not suffer performance losses, in this case the temperature raises faster due to a higher power consumption, and the cores executing the *bodytrack* application can only be boosted to 3.7 GHz during 0.1 s, reducing the boosting time in comparison to method A. Boosting method C applies a closed-loop control-based strategy, specifically, Intel's Turbo Boost. For such a case, although the *x264* application again does not suffer from performance losses, and also while the *bodytrack* application runs faster than its nominal frequency, the latter fails to meet its runtime requirements during the entire 0.25 s, particularly, only running at 3.7 GHz for merely 0.01 s out of the required 0.25 s. Contrarily, boosting method D intelligently selects to throttle-down the cores executing the *x264* application to 2.5 GHz. In this way, the cores executing the *bodytrack* application are able to be boosted to the required 3.7 GHz during the full 0.25 s, precisely reaching the critical threshold temperature at the end of the maximum expected boosting time, while the *x264* application is able to achieve 84% of its nominal performance.

Therefore, this motivational example shows the benefits of having an efficient boosting method. Nevertheless, until now, existing boosting techniques have neglected to make such careful decisions when selecting the boosting levels and the duration of the boosting interval, such that this remains an open problem.

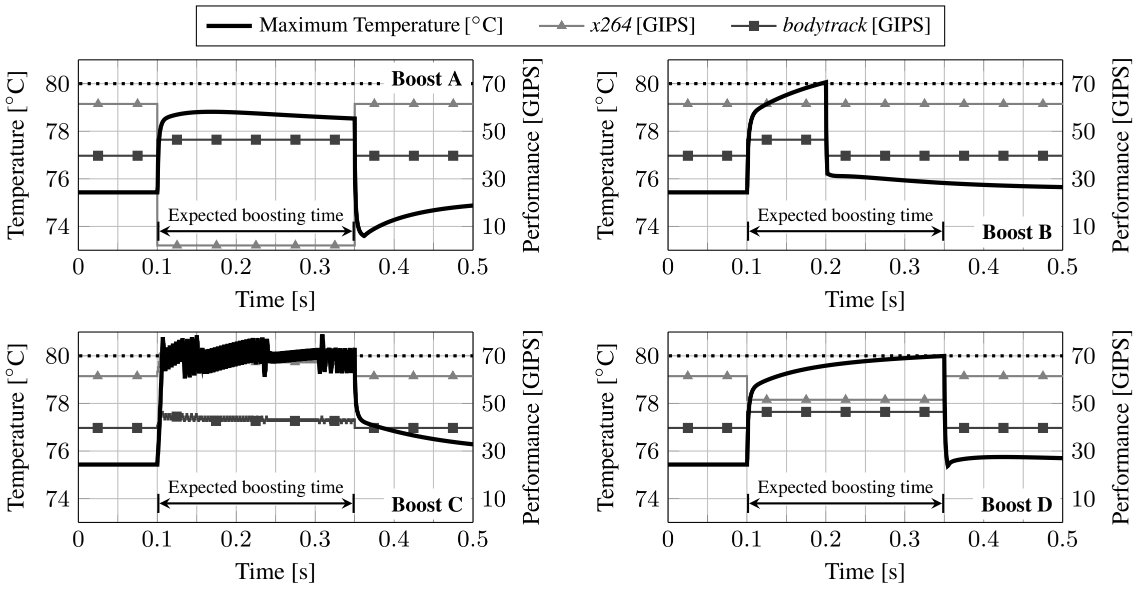


Figure 7.2: Motivational example with four different boosting techniques when the *bodytrack* application requires to increase its performance at runtime, for a critical temperature of 80°C (represented by the dotted line). The bold line shows the maximum temperature among all elements in the chip (left axis). The performance of the applications is measured in Giga Instructions per Second (GIPS) (right axis).

7.1.2 Problem Definition

We consider that the thread-to-core mapping of all the applications is known and given as an input. Through experimental evaluations, we assume to have given input tables with current and power profiles for every thread of every application, for the different execution phases of each application/thread, when executed on all different types of cores and for all available DVFS levels for each core type. We assume that the nominal DVFS levels of operation for every thread at a given time or execution phase are also known, and running the system at such nominal values for the given thread-to-core mapping meets the nominal performance requirements of all applications, while the critical threshold temperature (i.e., T_{DTM}) is not exceeded. That is,

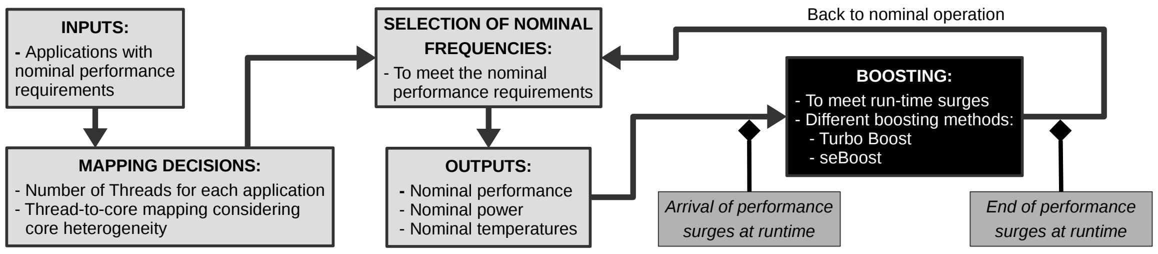
$$T_{\text{steady}_k} \leq T_{\text{DTM}} \qquad \text{for all } k \in \mathbf{L}$$

where T_{steady_k} can be computed through Equation (3.10). The given thread-to-core mapping and nominal DVFS levels can be derived with any of the existing solutions in the literature, e.g., [25,48].

$$T_{\text{steady}_k} = \sum_{j=1}^{N} \tilde{b}_{k,j} \cdot p_j + T_{\text{amb}} \cdot \sum_{j=1}^{N} \tilde{b}_{k,j} \cdot g_j \qquad \text{(3.10 revisited)}$$

At a given time t_{init}, for which we know the temperatures on all cores (i.e., column vector $\mathbf{T}_{\text{init}} = [T_{\text{init}k}]_{N\times1}$ is known), one or more application threads require a *runtime* increase in their performance, achieved by increasing the DVFS levels of their corresponding cores, i.e., boosting. The required boosting time is expected to last until no more than time instant t_{end}. During this time (i.e., the time between t_{init} and t_{end}), the cores that do not require boosting are considered to have a lower priority than the cores that do require boosting, and therefore, the non-boosted cores can be momentarily throttled-down by reducing their DVFS levels. All cores return to nominal operation after the runtime performance requirements expire. An overview of this description is presented in Figure 7.3.

When an application requires a runtime boost of its performance to meet its requirements, we assume that there are certain scenarios in which the maximum expected duration of the boosting interval is known or that it can be estimated. Given that this information can prove to be very useful when deciding the DVFS boosting levels, we assume it to be a system-level abstraction, and we consider it as an input to our algorithms. To



INPUTS:
- Applications with nominal performance requirements
MAPPING DECISIONS:
- Number of Threads for each application
- Thread-to-core mapping considering core heterogeneity
SELECTION OF NOMINAL FREQUENCIES:
- To meet the nominal performance requirements
OUTPUTS:
- Nominal performance
- Nominal power
- Nominal temperatures
Back to nominal operation
BOOSTING:
- To meet run-time surges
- Different boosting methods:
- Turbo Boost
- seBoost
Arrival of performance surges at runtime
End of performance surges at runtime

Figure 7.3: seBoost's system and problem overview.

specify such maximum expected boosting time requirements, the system could use historic profiling under different workloads. That is, although it might not be possible to predict when a certain kind of workload will arrive; however, the system can potentially estimate the duration of a runtime performance surge requirement based on the type of workload once the workload has arrived. For example, for a signal processing application, the application may not be able to predict when a new signal will arrive; however, the application could possibly estimate how much time it will require to process a new signal based on the size of the input data. Another potential method to estimate the maximum expected duration of the boosting intervals is through application phase-classification [97]. For example, we might have an application for which the critical workload phases require to be executed at the maximum DVFS levels during the entire critical phase, and the duration of each phase could also be profiled at design-time. Nevertheless, although the duration of a critical phase can be estimated, it might not be possible to predict when each critical phase will be executed.

With all these considerations in mind, we first assume that the DVFS levels that meet the runtime performance requirements surges are known and given as inputs. The problem then focuses on selecting the throttle-down levels for the non-boosted cores, such that the maximum temperature throughout the chip reaches T_{DTM} precisely at time t_{end}, also without exceeding the maximum chip power constraint (i.e., P_{max}) or the maximum chip current constraint (i.e., I_{max}). In this way, the non-boosted application threads do not suffer from unnecessary performance losses, like for example, adopting a trivial solution that throttles-down the non-boosted cores to their minimum DVFS levels. The solution to this problem is presented in Section 7.2.

Secondly, we assume that the runtime performance requirements and the associated DVFS levels that meet such requirements are unknown. Therefore, the new problem focuses on selecting the DVFS levels of the boosted cores in order to maximize the performance of their application threads, and also selecting the throttle-down levels for the non-boosted cores, such that the maximum temperature throughout the chip reaches T_{DTM} precisely at time t_{end}, also without exceeding P_{max} and I_{max}. In this case, maximizing the performance of the boosted cores has a higher priority than minimizing the performance losses of the non-boosted cores. The solution to this problem is presented in Section 7.3.

Finally, we consider cases in which the maximum expected boosting time is unknown and therefore cannot be specified in advance. Thus, this last problem focuses on finding the DVFS levels for the previous two cases, such that these DVFS levels can be sustained indefinitely (not only until time t_{end}) without exceeding T_{DTM}, P_{max}, and I_{max}. This is presented in Section 7.4.

7.2 Given Required Boosting Levels

For this case, the required DVFS levels for the cores that need boosting are given, and the duration of the maximum expected boosting time is also known, as conceptually shown in Figure 7.4.

According to the applications and threads mapped to every core, we can estimate the currents and power consumptions of every core for all possible DVFS levels from the input current and power profile tables. Therefore, knowing the initial temperatures and the power consumption throughout the chip, we can also estimate the temperature behavior on all cores after selecting/changing their DVFS levels by using our MatEx temperature estimation tool (described in Chapter 6). Particularly, we can compute/predict the temperature on every thermal node k at future time t_{end} after selecting the DVFS levels of all cores as already shown in Equation (6.3).

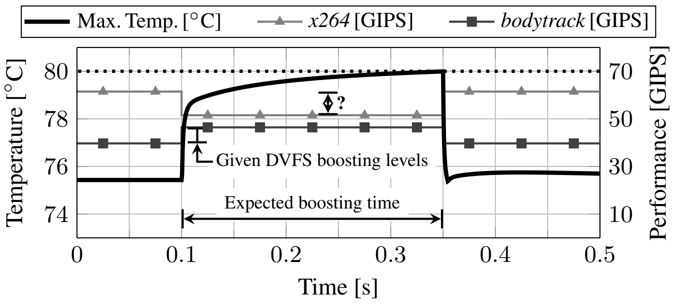


Figure 7.4: Conceptual problem example for the case with *given required boosting levels*, when the *bodytrack* application requires to increase its performance at runtime and the *x264* application can be throttled-down. The unknown parameter is shown with symbol **?**, in this case, the throttle-down levels of the non-boosted cores.

$$T_k\left(t\right) = T_{\text{steady}_k} + \sum_{i=1}^{N} e^{\lambda_i \cdot t} \cdot \Gamma_{k,i} \cdot \sum_{j=1}^{N} \tilde{\Gamma}_{i,j} \left(T_{\text{init}j} - T_{\text{steady}_j}\right) \qquad \text{(6.3 revisited)}$$

Nevertheless, although MatEx is very efficient in terms of execution time and can thus be used at runtime for future temperature estimation, evaluating all possible combinations of throttle-down levels for all non-boosted cores would still require an excessively long computation time. Because of this reason, we are limited in the number of combinations that can be evaluated in order to select such throttle-down levels. Particularly, our proposed seBoost technique, presented in Figure 7.5 for this case, is based on a binary search like approach. Basically, seBoost sets the DVFS levels of the boosted cores such that the runtime performance requirements surges can be satisfied. Then, by applying binary search proportional to the nominal DVFS values on each core, seBoost tests a limited number of combinations of DVFS levels for the non-boosted cores. Estimating the temperatures at time t_{end} using MatEx, i.e., through Equation (3.10) and Equation (6.3), seBoost selects the combination that resulted in the highest DVFS levels for the non-boosted cores such that the maximum temperature among all blocks in the floorplan at time t_{end} does not exceed T_{DTM}, i.e., such that $\max_{\forall k \in \mathbf{L}} \left\{T_k\left(t_{\text{end}}\right)\right\} \leq T_{\text{DTM}}$, and both $P_{\max}$ and $I_{\max}$ are also satisfied.

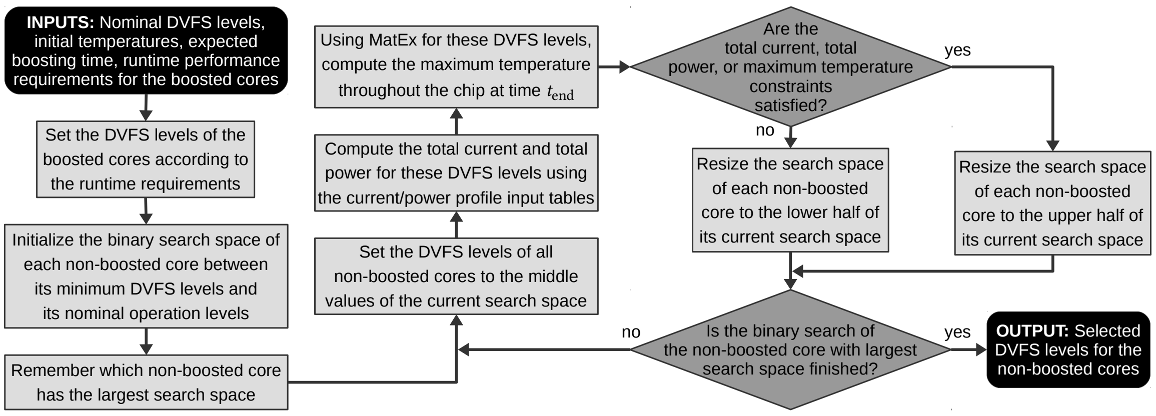

INPUTS: Nominal DVFS levels, initial temperatures, expected boosting time, runtime performance requirements for the boosted cores
Set the DVFS levels of the boosted cores according to the runtime requirements
Initialize the binary search space of each non-boosted core between its minimum DVFS levels and its nominal operation levels
Remember which non-boosted core has the largest search space
Set the DVFS levels of all non-boosted cores to the middle values of the current search space
Compute the total current and total power for these DVFS levels using the current/power profile input tables
Using MatEx for these DVFS levels, compute the maximum temperature throughout the chip at time t_{end}
Are the total current, total power, or maximum temperature constraints satisfied?
yes
no
Resize the search space of each non-boosted core to the lower half of its current search space
Resize the search space of each non-boosted core to the upper half of its current search space
Is the binary search of the non-boosted core with largest search space finished?
no
yes
OUTPUT: Selected DVFS levels for the non-boosted cores

Figure 7.5: Flow chart of our seBoost technique for *given boosting requirements*.

The algorithm presented in Figure 7.5 is described in detail as follows. First, the DVFS levels of the boosted cores are set to meet the runtime performance requirements. Moreover, the binary search space of each non-boosted core is initialized between its minimum DVFS levels and its nominal operation levels, such that the size of the search space for each core depends on the number of available DVFS levels between these two points. Hence, we keep track of the non-boosted core with the largest search space, which will determine the exit condition of the binary search loop. Specifically, since the number of iterations of the binary search depends on the search space of each core, which may vary from core to core, the search for the throttle-down levels of cores with small search spaces will finish before the search for cores with large search spaces is completed. After the initialization is completed, we enter the proportional binary search loop. The first step inside the search loop is to set the DVFS levels of the non-boosted cores to the middle value of the search space of each core. Then, considering the already selected DVFS levels for the boosted cores, and by assuming that the non-boosted cores would run at the DVFS levels in the middle of their search space, we compute the total current and power consumption for these settings by using the input current and power profile tables, and we use MatEx to estimate the temperature at future time t_{end}. At this point, we can verify whether selecting these throttle-down levels for the non-boosted cores would satisfy the temperature, power, and current constraints, T_{DTM}, P_{max}, and I_{max}, respectively. Depending on whether all constraints are satisfied or not, the search space is resized such that the next iteration considers the upper or lower half of the current search space. This proportional binary search loop is repeated until the search in the non-boosted core with the largest search space ends. Figure 7.6 shows a graphical example of the procedure of our seBoost technique for the case with *given required boosting levels*.

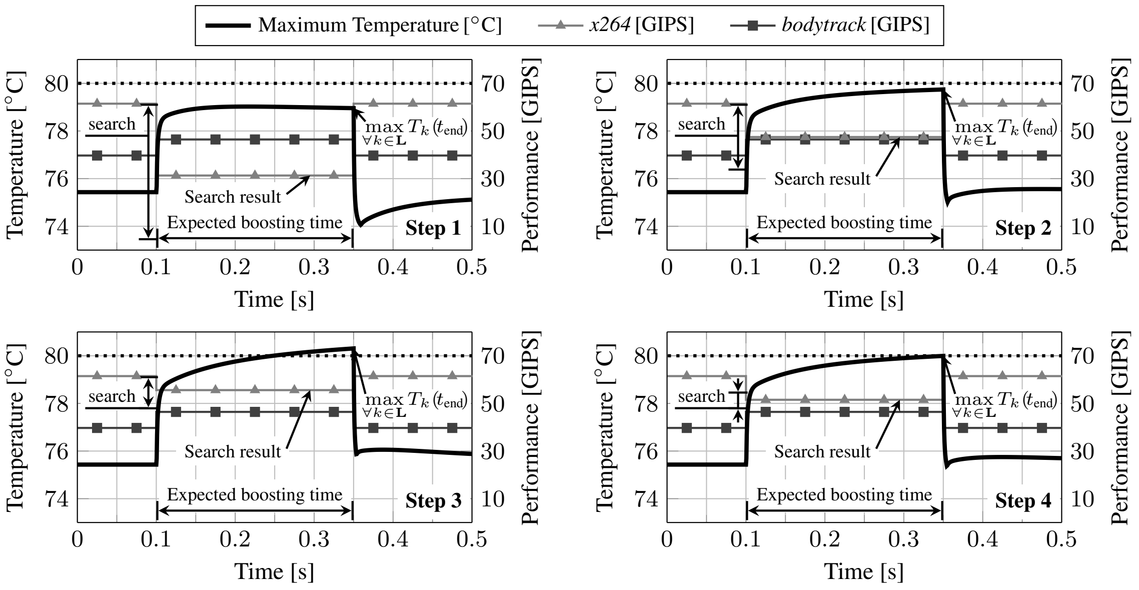


Figure 7.6: Conceptual example of our seBoost technique for *given boosting requirements* (algorithm illustrated in Figure 7.5), when the *bodytrack* application requires to increase its performance at runtime and the *x264* application can be throttled-down. The bold line shows the maximum temperature among all elements in the chip (left axis). The performance of the applications is measured in Giga Instructions per Second (GIPS) (right axis).

According to our hardware model in Chapter 3.2, every core i has $\hat{F}_i^{\text{core}}$ available active frequencies, denoted as $\left\{F_{i,1}^{\text{core}}, F_{i,2}^{\text{core}}, \ldots, F_{i,\hat{F}_i^{\text{core}}}^{\text{core}}\right\}$, where $F_{i,\hat{F}_i^{\text{core}}}^{\text{core}}$ is the maximum frequency for core i. Therefore, assuming that the nominal operating frequency of core i is denoted as F_{nom_i}, such that $1 \leq \text{nom}_i \leq \hat{F}_i^{\text{core}}$, and assuming that set $\boldsymbol{\Upsilon}$ holds the indexes of all non-boosted cores at a specific time instant, then the number of combinations evaluated by seBoost is merely $\log\left(\max_{\forall i \in \boldsymbol{\Upsilon}} \{\text{nom}_i\}\right)$. Contrarily, a brute-force algorithm would have evaluated $\prod_{\forall i \in \boldsymbol{\Upsilon}} \text{nom}_i$ combinations. In case it occurs that running all non-boosted cores at their minimum DVFS levels still exceeds either P_{max}, I_{max}, or T_{DTM} at time t_{end}, then satisfying the given boosting requirements for this case is not feasible, and therefore the boosting time interval will be shorter than the maximum expected time for the given DVFS level requirements and initial temperatures.

7.3 Unknown Required Boosting Levels

In this section, we consider that the required DVFS levels of the cores that need boosting are unknown, but the duration of the maximum expected boosting time is known, as conceptually shown in Figure 7.7.

Our seBoost algorithm for this case is presented in Figure 7.8, whose core computation is based on the algorithm from Section 7.2 illustrated in Figure 7.5 (called as a function inside the algorithm in Figure 7.8), but

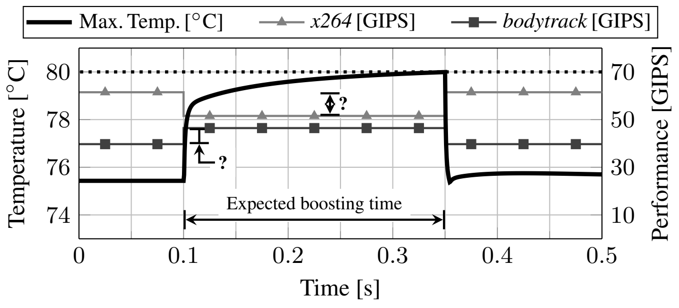
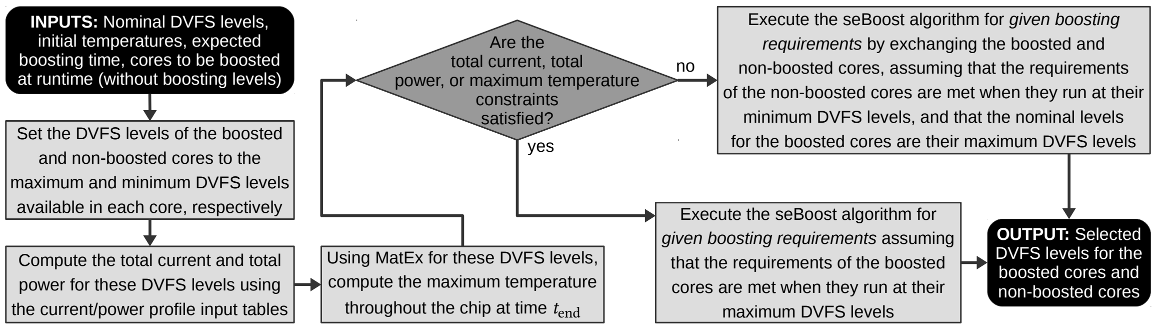


Figure 7.7: Conceptual problem example for the case with *unknown required boosting levels*, when the *bodytrack* application requires to increase its performance at runtime and the *x264* application can be throttled-down. The unknown parameters are shown with symbol **?**, in this case, the boosting levels of the boosted cores and the throttle-down levels of the non-boosted cores.



INPUTS: Nominal DVFS levels, initial temperatures, expected boosting time, cores to be boosted at runtime (without boosting levels)
Set the DVFS levels of the boosted and non-boosted cores to the maximum and minimum DVFS levels available in each core, respectively
Compute the total current and total power for these DVFS levels using the current/power profile input tables
Using MatEx for these DVFS levels, compute the maximum temperature throughout the chip at time t_{end}
Are the total current, total power, or maximum temperature constraints satisfied?
no
yes
Execute the seBoost algorithm for *given boosting requirements* by exchanging the boosted and non-boosted cores, assuming that the requirements of the non-boosted cores are met when they run at their minimum DVFS levels, and that the nominal levels for the boosted cores are their maximum DVFS levels
Execute the seBoost algorithm for *given boosting requirements* assuming that the requirements of the boosted cores are met when they run at their maximum DVFS levels
OUTPUT: Selected DVFS levels for the boosted cores and non-boosted cores

Figure 7.8: Flow chart of our seBoost technique for *unknown required boosting levels*.

which requires some additional logic. Namely, by assuming that the boosted cores are set to their maximum DVFS levels and that the non-boosted cores are set to their minimum DVFS levels, seBoost first verifies if the temperature at future time t_{end} would remain below T_{DTM}, and whether the current and power constraints are also satisfied. If this verification is successful, it implies that there exists at least one solution in which all boosted cores can be executed at their maximum DVFS levels. Therefore, since maximizing the performance of the boosted cores is the priority, we execute the algorithm illustrated in Figure 7.5 by assuming that the runtime requirements of the boosted cores are satisfied when they run at their maximum DVFS levels, such that we find the throttle-down levels for the non-boosted cores. Contrarily, if all boosted cores *cannot* be executed at their maximum DVFS levels without violating one of the constraints, then we need to find different boosting levels. For this purpose, we execute the algorithm illustrated in Figure 7.5 by exchanging the boosted and non-boosted cores, assuming that the runtime requirements of the non-boosted cores are met when they run at their minimum DVFS levels, and that the nominal operation levels for the boosted cores are their maximum DVFS levels. In this way, the non-boosted cores are forced to run as slow as possible, and the algorithm illustrated in Figure 7.5 will now select the DVFS levels of the cores that *do* require boosting, rather than the throttle-down levels of the non-boosted cores.

Figure 7.9, Figure 7.10, and Figure 7.11 present a graphical example of the procedure of our seBoost technique for this case. Figure 7.9 first shows two cases with opposite results for evaluating if the boosted cores can safely be executed at their maximum DVFS levels while the non-boosted cores are executed at their minimum DVFS levels. Figure 7.10 then shows the procedure example when the verification was successful, and Figure 7.11 shows the procedure example for the opposite scenario.

7.4 Unknown Maximum Expected Boosting Time

In this section, we extend our two previous algorithms to consider cases in which the maximum expected boosting time is unknown and therefore cannot be specified in advance. Here, we assume that the runtime performance requirements surges can last for a very long time, and therefore the goal is to find DVFS levels that can be sustained indefinitely. Specifically, instead of obtaining the DVFS levels that result in transient future temperatures for which the maximum temperature throughout the chip reaches T_{DTM} precisely at time

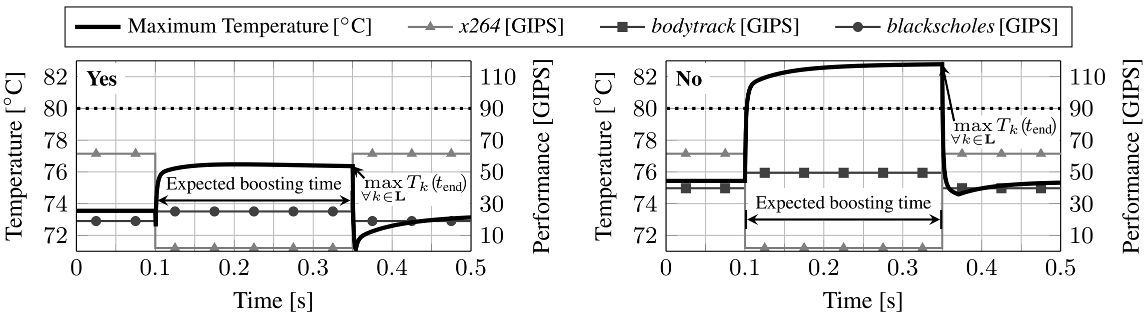


Figure 7.9: Conceptual example of the evaluation of our seBoost technique for *unknown required boosting levels* (illustrated in Figure 7.8), when testing if the boosted cores *can* (left figure) or *cannot* (right figure) be safely executed at their maximum DVFS levels. The cores requiring boosting are those mapped with the *blackscholes* or the *bodytrack* application. The bold line shows the maximum temperature among all elements in the chip at any given time (left axis). The performance of the applications is measured in Giga Instructions per Second (GIPS) (right axis).

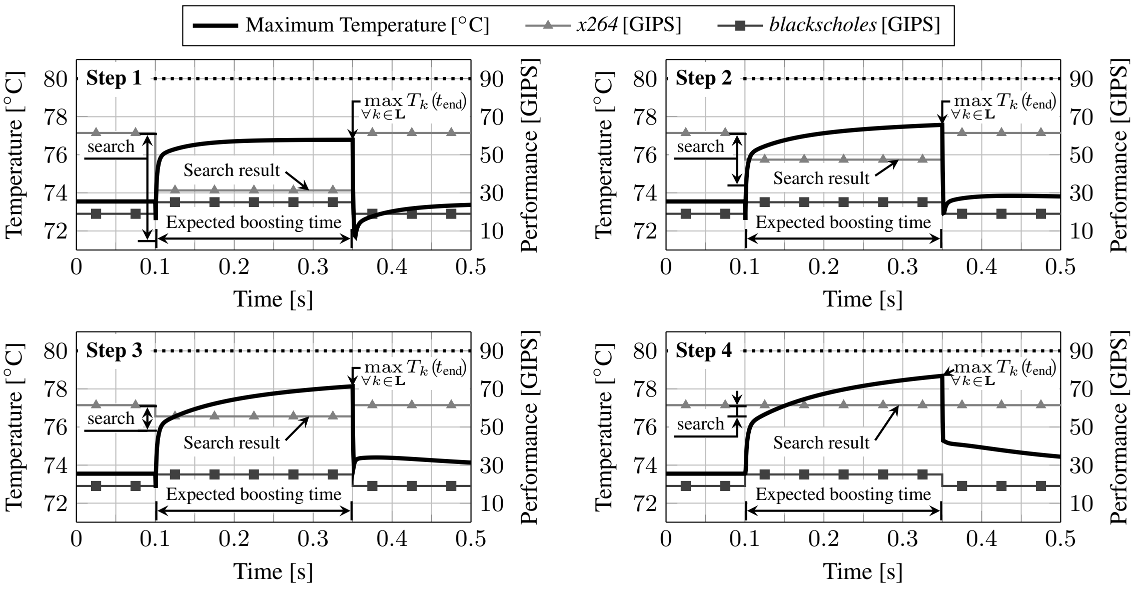


Figure 7.10: Conceptual example of our seBoost technique for *unknown required boosting levels* (illustrated in Figure 7.8), when all boosted cores of the *blackscholes* application *can* be safely executed at their maximum DVFS levels. Therefore, the DVFS levels of the boosted cores are set to their maximum values, and the throttle-down levels of the non-boosted cores are selected through the algorithm for *given boosting requirements* (illustrated in Figure 7.5). The bold line shows the maximum temperature among all elements in the chip at any given time (left axis). The performance of the applications is measured in Giga Instructions per Second (GIPS) (right axis).

t_{end}, we now focus on staying just below the value of T_{DTM} when $t \to \infty$, i.e., in the steady state, without also exceeding $P_{\max}$ and $I_{\max}$. Therefore, we only need to make some minor changes in order to extend the algorithms presented in Section 7.2 and Section 7.3. Particularly, for both algorithms illustrated in Figure 7.5 and Figure 7.8, instead using MatEx to compute $T_k(t_{\text{end}})$ through Equation (6.3) in order to select the highest DVFS levels for which $\max_{\forall k \in \mathbf{L}} \{T_k(t_{\text{end}})\} \leq T_{\text{DTM}}$, we now simply compute T_{steady_k} through Equation (3.10), in order to select the highest DVFS levels for which $\max_{\forall k \in \mathbf{L}} \left\{T_{\text{steady}_k}\right\} < T_{\text{DTM}}$.

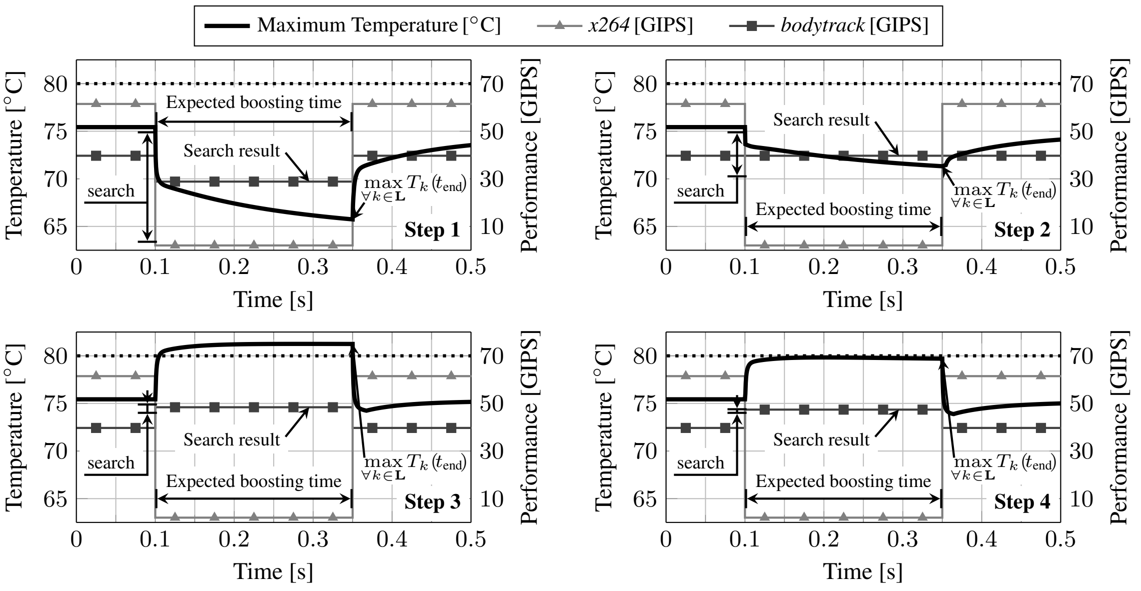


Figure 7.11: Conceptual example of our seBoost technique for *unknown required boosting levels* (illustrated in Figure 7.8), when all boosted cores of the *bodytrack* application *cannot* be safely executed at their maximum DVFS levels. Therefore, the boosted and non-boosted cores are exchanged, the DVFS levels of the non-boosted cores are set to their minimum values, and the boosting levels of the boosted cores are selected through the algorithm for *given boosting requirements* (illustrated in Figure 7.5). The bold line shows the maximum temperature among all elements in the chip at any given time (left axis). The performance of the applications is measured in Giga Instructions per Second (GIPS) (right axis).

7.5 Concurrency and Closed-Loop Control-Based Boosting

The algorithms illustrated in Figure 7.5 and Figure 7.8 implicitly assume that all the boosted cores have the same maximum expected boosting time. However, in practice there might be cases in which the runtime performance requirements surges of some applications finish and go back to nominal levels, or there might be cases in which new applications/threads require boosting when we are already in a boosting interval. When this occurs, time t_{end} should be set to the maximum value among all boosting times of all boosted cores. Furthermore, both algorithms (depending on which one is being used) should be re-executed every time the boosting requirements change.

For example, once one or more cores finish their boosted execution, the algorithm illustrated in Figure 7.5 (or Figure 7.8, depending on the case) should be re-executed (now with less cores requiring boosting), in order to improve the overall performance of the system, given that now more cores can probably be safely executed at higher DVFS levels due to the decrement in the power consumption of the cores that just finished their boosted execution. Similarly, if during a boosting period one or more additional cores require boosting, time t_{end} is set to the maximum expected absolute boosting time among all boosted cores, and the algorithm illustrated in Figure 7.5 (or Figure 7.8, depending on the case) is re-executed in order to adjust the DVFS levels such that we can feasibly boost all the required cores, specifically, the cores that were already being boosted plus the new cores that also require boosting. Figure 7.12 shows an example with such concurrent boosting requirements.

Furthermore, although the simple closed-loop control-based boosting technique used by Turbo Boost does not guarantee that the runtime performance requirements surges are satisfied, it does provide a simple method to prolong the boosting intervals *after* the temperature, power, or current constraints are exceeded. For example, even when running the non-boosted cores at their minimum DVFS levels it is not always possible to satisfy the runtime surges during the entire expected boosting times, particularly for high initial temperatures. For such cases, it would be pessimistic to directly return to nominal operation after the temperature, power, or current constraints are exceeded. Therefore, seBoost can easily incorporate a simple closed-loop control-

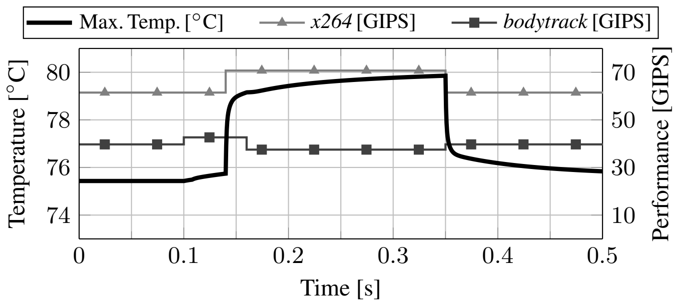


Figure 7.12: Concurrent boosting example. The *bodytrack* application is boosted to 3.3 GHz from 0.10 s to 0.16 s, and the *x264* application is boosted to 3.4 GHz from 0.14 s to 0.35 s. The bold line shows the maximum temperature among all elements in the chip (left axis). The performance of the applications is measured in Giga Instructions per Second (GIPS).

based boosting technique, like the one used by Turbo Boost, which is triggered *after* any of the constraints are exceeded. This simple control-based technique can then reduce the DVFS levels of the boosted cores, not longer meeting the performance requirements surges, but achieving higher performance than at nominal operation.

7.6 Experimental Evaluations

This section presents experimental evaluations that compare seBoost, Turbo Boost [9], and a simple boosting method that always throttles down the non-boosted cores to the slowest frequencies. For the evaluations, we use the simulation framework described in Chapter 4 in detailed mode, considering the heterogeneous architecture illustrated in Figure 4.4 and described in Chapter 4.2.2. For benchmarks, we consider applications from the PARSEC benchmark suite [4], described in Chapter 4.3. The ambient temperature is set to 45°C, and we consider a critical temperature of 80°C.

7.6.1 Results

We run the applications from the PARSEC benchmark suite under different scenarios. First, we focus on different applications individually, considering multiple instances of the same application, with different number of threads per instance and also different thread-to-core mappings. For each scenario we also consider different arrival periods for the runtime performance requirements surges and different maximum expected boosting times. Secondly, we focus on mixed application scenarios, considering several cases with different applications, number of threads and thread-to-core mappings. In all cases, we consider that the nominal frequency for the OOO and *simple* Alpha cores is 2.0 GHz, and the nominal frequencies for the Cortex-A7 and Cortex-A15 cores are 0.8 GHz and 1.5 GHz, respectively. Furthermore, for every application scenario, we consider that the system was running for a long enough time such that the temperatures throughout the chip correspond to the steady-state temperatures at nominal frequencies, and we assume these temperatures to be the initial temperatures in each case. As also done in Chapter 5.6, given that the Odroid-XU3 platform does not provide performance counters to measure the total number of executed instructions, we use throughput as our performance metric, where throughput is defined as the total number of application instances finished (or partially finished) every second. We consider that every time an application instance finishes, another instance is immediately executed under the same mapping and DVFS settings.

Figure 7.13 shows the timing behavior of each policy for the mixed application scenario M6, as detailed in Figure 7.15. The computational overheads incurred by seBoost to decide the boosting levels at runtime are considered in the experiments, resulting in 7.5 ms for this specific case as shown in the figure. With respect to performance, we can see that both the simple throttling-down method and seBoost satisfy the runtime performance requirements surges during the entire boosting time interval, but seBoost manages this with higher performance for the non-boosted applications. In regards to Turbo Boost, we can see that the performance of the non-boosted applications is in fact much higher than that of seBoost. Moreover, the average performance of the boosted applications is also slightly higher than that of seBoost. However, although the average boosted performance is slightly higher (only 3%), Turbo Boost fails to *constantly* meet the minimum runtime performance requirements for the boosted applications by a very small amount (around 2 frequency steps), particularly, only satisfying the surges during 49% of the total boosting time.

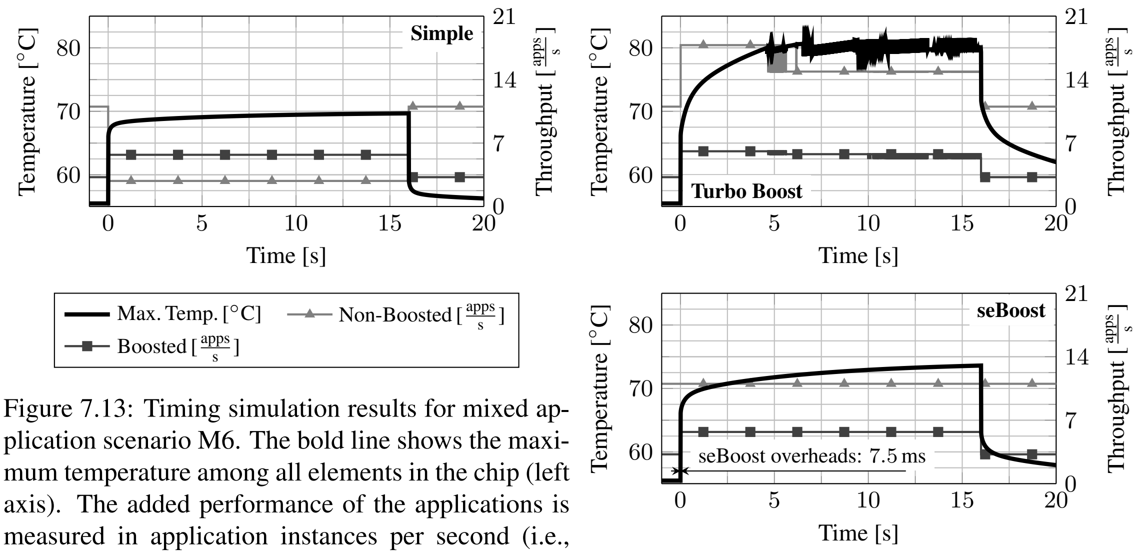


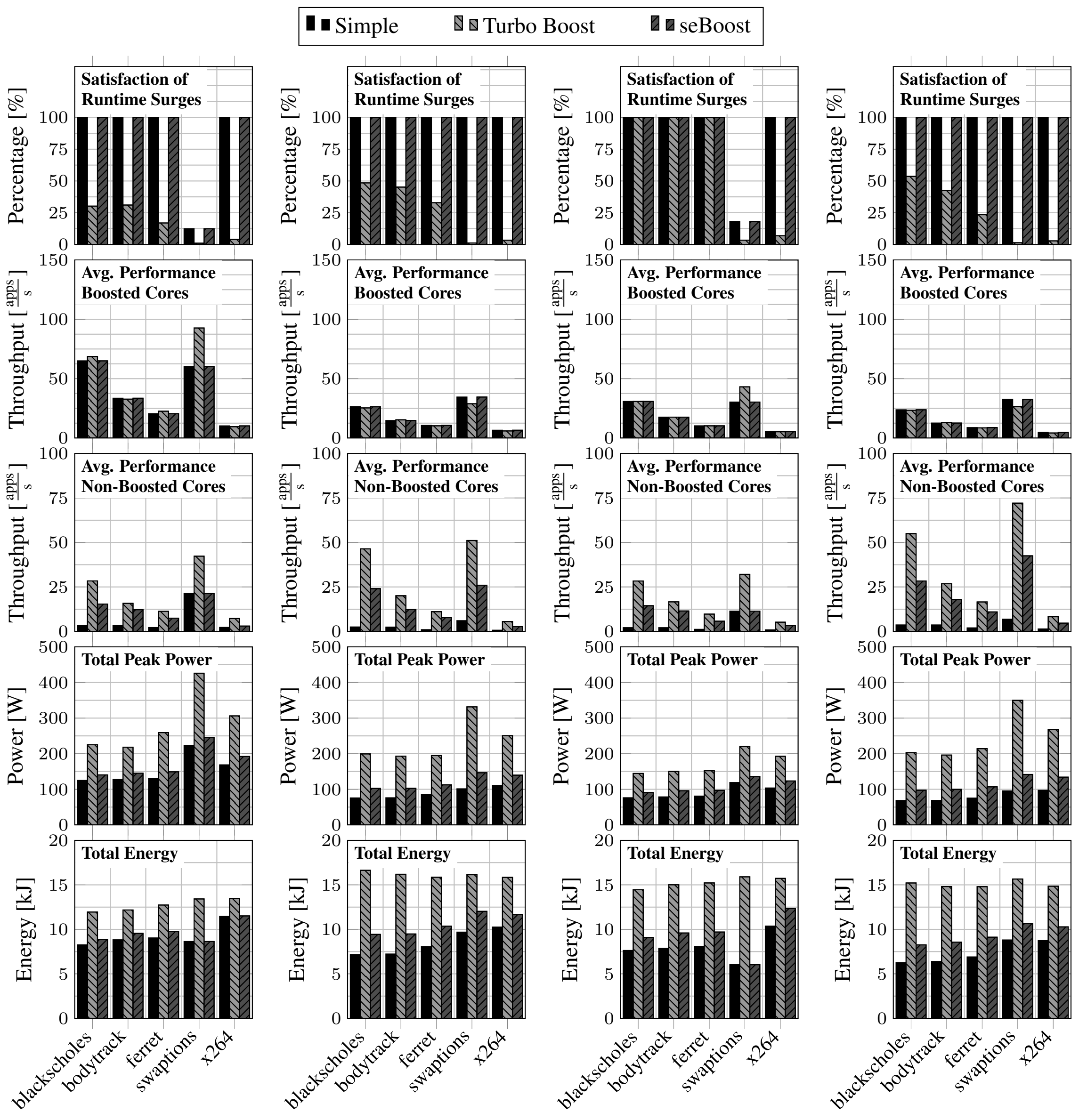
Figure 7.13: Timing simulation results for mixed application scenario M6. The bold line shows the maximum temperature among all elements in the chip (left axis). The added performance of the applications is measured in application instances per second (i.e., throughput).

The details of the mapping scenarios used for the experiment considering individual applications can be found in Table 7.1, and Figure 7.14 shows the corresponding results for 100 s of execution for all the evaluated scenarios. Similarly, Figure 7.15 shows the mapping scenarios and results for 100 s of execution for all the evaluated scenarios in the experiment considering mixed applications.

Scenario	**Alpha OOO**	**Alpha** ***simple***	**Cortex-A15**	**Cortex-A7**
I1 Boost: 10s Period: 30s	a: $4^{[3.6]}$ threads $4^{[3.6]}$ threads b: $4^{[3.6]}$ threads $4^{[3.6]}$ threads c: 4 threads 4 threads	a: $4^{[4.0]}$ threads b: $4^{[4.0]}$ threads c: $4^{[4.0]}$ threads d: $4^{[4.0]}$ threads	a: 4 threads b: 4 threads c: 4 threads d: 4 threads	a: 4 threads b: 4 threads c: 4 threads d: 4 threads
I2 Boost: 20s Period: 25s	a: 8 threads b: 8 threads c: $2^{[4.0]}$ threads $6^{[4.0]}$ threads	a: - b: - c: 3 threads d: 2 threads	a: $4^{[2.0]}$ threads b: $2^{[2.0]}$ threads c: $2^{[2.0]}$ threads d: -	a: - b: $2^{[1.4]}$ threads c: $2^{[1.4]}$ threads d: $4^{[1.4]}$ threads
I3 Boost: ∞ Period: -	a: $5^{[4.0]}$ threads $3^{[4.0]}$ threads b: - c: 7 threads 1 thread	a: $4^{[4.0]}$ threads b: 1 thread c: 2 threads d: -	a: $2^{[2.0]}$ threads b: - c: 1 thread d: 4 threads	a: $4^{[1.4]}$ threads b: - c: - d: 2 threads
I4 Boost: 7s Period: 10s	a: $6^{[4.0]}$ threads $2^{[4.0]}$ threads b: 8 threads c: 4 threads 4 threads	a: 3 threads b: 4 threads c: - d: -	a: - b: - c: 4 threads d: 2 threads	a: - b: 1 thread c: $4^{[1.4]}$ threads d: -

Table 7.1: Details of the application mapping scenarios for the experiment with individual applications. Indexes a, b, c, and d represent the cluster ID as explained in Figure 4.4. Every line corresponds to an application instance executed in the corresponding cluster with the indicated number of threads, where "-" means that a cluster is not executing any application. A super-index enclosed in brackets next to the number of threads implies that the specific application instance will have runtime performance surges, where the target frequency is the number between the brackets (in GHz). The duration of the surges is detailed in the *Scenario* column (under *Boost*), while *Period* details how often such surges arrive.

In Figure 7.14 and Figure 7.15, we can see the percentage of time that the runtime performance requirements surges are satisfied for each boosting policy, the total average performance for the boosted applications/cores, the total average performance for the non-boosted applications/cores, the total peak power consumption, and the total energy consumption. In both figures, there are a few cases in which, for the given initial temperatures, it is not possible to satisfy the runtime requirements surges without violating the hardware constraints, and therefore neither seBoost or the simple throttling-down method manage to satisfy the requirements 100% of the time. For the rest of the evaluated cases, seBoost and the simple throttling-down method satisfy the requirements during the entire boosting interval (except for seBoost's small computation overheads of a few milliseconds), as seen in Figure 7.14 and Figure 7.15. Nevertheless, the simple throttling-down boosting method does so while incurring unnecessary performance losses for the non-boosted applications.



(a) Scenario I1 (b) Scenario I2 (c) Scenario I3 (d) Scenario I4

Figure 7.14: Evaluation results for individual applications from the PARSEC benchmark suite. We consider multiple instances of the same application, with different number of threads per instance, different thread-to-core mappings, different arrival periods for the runtime performance surges, and different maximum expected boosting times. Details can be found in Table 7.1.

In regards to Turbo Boost, for all the evaluated cases, although Turbo Boost achieves higher average performance than seBoost for the non-boosted cores, Turbo Boost *rarely* manages to satisfy the runtime requirements during the entire boosting interval, and the specific percentages vary drastically depending on the application scenarios. Sometimes this occurs because the non-boosted cores are executed at higher DVFS levels than their nominal requirements, unnecessarily increasing the power consumption. However, given that Turbo Boost is not aware of the performance that the applications require, sometimes the failure to satisfy the runtime requirements during the entire boosting interval occurs because the DVFS levels of the boosted cores are set to values much higher than necessary, resulting in long cool-down times. As seen in Figure 7.14 and Figure 7.15, compared to seBoost, the *over-boosting* incurred by Turbo Boost might sometimes result in

Scenario	**Alpha OOO**	**Alpha *simple***	**Cortex-A15**	**Cortex-A7**
M1 Boost: ∞ Period: -	a: 2 *blacks.* 6 *x264* b: $4^{[3.6]}$ *ferret* $4^{[3.6]}$ *bodytr.* c: 8 *dedup*	a: - b: $2^{[3.8]}$ *dedup* c: 1 *ferret* d: $3^{[3.8]}$ *bodytr.*	a: - b: 1 *facesim* c: $4^{[1.9]}$ *fluidan.* d: 2 *blacks.*	a: 2 *ferret* b: - c: $2^{[1.0]}$ *vips* d: 4 *streamcl.*
M2 Boost: 3s Period: 45s	a: $8^{[4.0]}$ *bodytr.* b: $1^{[4.0]}$ *x264* c: $8^{[4.0]}$ *dedup*	a: 1 *dedup* b: 4 *canneal* c: 3 *x264* d: 2 *blacks.*	a: 4 *bodytr.* b: $2^{[1.8]}$ *facesim* c: $1^{[1.8]}$ *swapt.* d: 1 *x264*	a: 4 *swapt.* b: 2 *vips* c: 2 *freqm.* d: 2 *x264*
M3 Boost: 27s Period: 35s	a: $5^{[4.0]}$ *dedup* $3^{[4.0]}$ *blacks.* b: - c: $7^{[3.4]}$ *ferret* $1^{[3.4]}$ *dedup*	a: - b: - c: - d: $4^{[4.0]}$ *dedup*	a: - b: - c: - d: 1 streamcl.	a: 2 *blacks.* b: 1 *bodytr.* c: 4 *x264* d: 4 *fluidan.*
M4 Boost: 30s Period: 50s	a: 8 *bodytr.* b: 7 *x264* 1 *ferret* c: 8 *blacks.*	a: - b: - c: $4^{[4.0]}$ *x264* d: $4^{[4.0]}$ *canneal*	a: $2^{[1.9]}$ *streamcl.* b: $2^{[1.8]}$ *swapt.* c: $1^{[2.0]}$ *bodytr.* d: $1^{[2.0]}$ *ferret*	a: $4^{[1.2]}$ *x264* b: $2^{[1.2]}$ *freqm.* c: - d: -
M5 Boost: 23s Period: 40s	a: 1 *bodytr.* 7 *x264* b: $2^{[4.0]}$ *blacks.* $6^{[4.0]}$ *dedup* c: -	a: 2 *ferret* b: - c: - d: -	a: 1 *bodytr.* b: 2 *blacks.* c: 1 *facesim* d: 2 *swapt.*	a: 4 *ferret* b: 2 *facesim* c: $4^{[1.4]}$ *fluidan.* d: 1 *streamcl.*
M6 Boost: 16s Period: 45s	a: $3^{[4.0]}$ *canneal* $2^{[4.0]}$ *x264* b: 8 *ferret* c: 8 *bodytr.*	a: $1^{[3.4]}$ *blacks.* b: $2^{[3.2]}$ *bodytr.* c: 3 *dedup* d: $4^{[4.0]}$ *x264*	a: 4 *freqm.* b: 1 *ferret* c: 1 *streamcl.* d: 4 *vips*	a: 1 *swapt.* b: 2 *x264* c: 4 *blacks.* d: 2 *bodytr.*
M7 Boost: 10s Period: 30s	a: $4^{[3.6]}$ *x264* $4^{[3.6]}$ *canneal* b: $4^{[3.6]}$ *blacks.* $4^{[3.6]}$ *swapt.* c: 4 *dedup* 4 *x264*	a: 4 *swapt.* b: 4 *ferret* c: $4^{[4.0]}$ *ferret* d: $4^{[4.0]}$ *blacks.*	a: $4^{[2.0]}$ *fluidan.* b: 4 *freqm.* c: 4 *facesim* d: $4^{[1.9]}$ *streamcl.*	a: 4 *vips* b: $4^{[1.4]}$ *freqm.* c: $4^{[1.2]}$ *x264* d: 4 *bodytr.*
M8 Boost: 32s Period: 40s	a: - b: 8 *dedup* c: $4^{[4.0]}$ *ferret* $4^{[4.0]}$ *x264*	a: - b: - c: 3 *x264* d: 3 *dedup*	a: 1 *x264* b: 4 *bodytr.* c: 4 *streamcl.* d: 1 *ferret*	a: $1^{[1.4]}$ *blacks.* b: $2^{[1.3]}$ *vips* c: $2^{[1.0]}$ *swapt.* d: $4^{[1.4]}$ *facesim*
M9 Boost: ∞ Period: -	a: $1^{[3.8]}$ *ferret* $7^{[3.8]}$ *blacks.* b: $4^{[4.0]}$ *canneal* c: 8 *dedup*	a: 1 *dedup* b: 3 *bodytr.* c: 2 *ferret* d: 4 *swapt.*	a: 1 *swapt.* b: $4^{[2.0]}$ *blacks.* c: - d: -	a: $1^{[1.3]}$ *vips* b: 2 *ferret* c: 4 *streamcl.* d: $2^{[1.3]}$ *facesim*
M10 Boost: ∞ Period: -	a: $5^{[4.0]}$ *ferret* $3^{[4.0]}$ *bodytr.* b: - c: 7 *canneal* 1 *swapt.*	a: $4^{[3.4]}$ *x264* b: 1 *dedup* c: 2 *blacks.* d: -	a: $2^{[1.9]}$ *freqm.* b: - c: 1 *x264* d: 4 *vips*	a: $4^{[1.4]}$ *facesim* b: - c: - d: 2 *streamcl.*

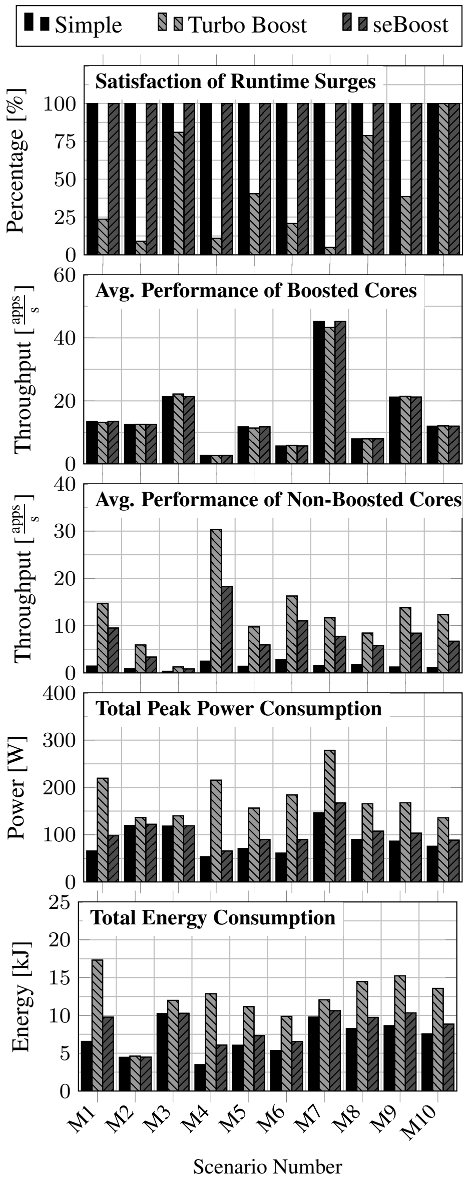


Figure 7.15: Evaluation results for mixed applications from the PARSEC benchmark suite (right). We consider different applications, with different number of threads per application instance, different thread-to-core mappings, different arrival periods for the runtime performance surges, and different maximum expected boosting times. Details can be found in the table (left), which is very similar to Table 7.1. The main difference is that in this table we detail *which* application type is executed in each cluster, and the word *threads* is omitted.

higher average performance for the boosted cores. However, the performance gains for using Turbo Boost are relatively very small and arguably unjustified when considering the big increments to the total peak power and energy consumption. For example, Turbo Boost achieves 3% higher boosted average performance for scenario M6, while resulting in a peak power and energy consumption of 105% and 51%, respectively, higher than that of seBoost.

With respect to the computational overheads incurred by seBoost, when implemented in software (C++) as a single threaded application, the worst-case measured execution time for seBoost for all the evaluated cases was below 7.5 ms. This shows that seBoost is in fact a *lightweight* technique which might be suitable for runtime usage. Implementing seBoost as a dedicated hardware controller would result in negligible time overheads, especially given that the computations done by MatEx to estimate the future temperatures in the algorithms illustrated in Figure 7.5 and Figure 7.8 can be fully parallelized, as already explained in Chapter 6.

7.7 Summary

This chapter presented seBoost, an efficient and lightweight boosting technique based on future transient temperature estimation (specifically, based on MatEx, presented in Chapter 6). This technique guarantees meeting any performance requirements surges that can occur at runtime, and this is achieved by executing the boosted cores at the required DVFS levels for the entire boosting intervals, while throttling down the non-boosted cores. In order to minimize the performance losses of the applications being executed on the non-boosted cores, the throttling down levels are chosen such that the maximum temperature throughout the chip reaches the critical threshold temperature precisely when the boosting is expected to expire. Our experiments show that seBoost can in fact guarantee the required runtime performance surges, while the state-of-the-art closed-looped control-based boosting techniques fail to constantly satisfy these runtime requirements, even while consuming more power and energy.

Chapter 8

Energy and Peak Power Efficiency Analysis for Simple Approximation Schemes

8.1 Overview

For multicore and manycore cluster-based architectures with multiple voltage islands that are executing performance-constrained applications or real-time tasks, the task partitioning stage and DVFS schedule play a major role for peak power reduction and energy minimization. Here, *task partitioning* refers to the process of grouping all the tasks into separate sets of tasks (or tasks sets), where each task set can hold one or more tasks, such that all tasks belonging to the same task set will be mapped and executed on a single core by using preemption. Moreover, the *DVFS schedule* refers to which specific DVFS levels are chosen in each cluster/-core at different points in time. A combination of a task partitioning policy and a DVFS schedule policy is said to be *feasible* when all tasks meet their performance constraints or hard real-time deadlines, and it is said to be *optimal* in terms of energy consumption or peak power consumption, if it results in the minimum energy consumption or peak power consumption, respectively.

With regards to task partitioning, having in consideration the convexity of the dynamic power consumption of a core (as discussed in Chapter 3.3), the energy consumption for executing some workload in one core at a specific frequency will generally be bigger than executing the same workload (perfectly distributed) in two cores at half of the frequency (as suggested by Chapter 3.4). This suggests that in most cases, for a group of unpartitioned tasks assigned to a cluster, a task partitioning strategy that balances the workloads throughout all the cores in the cluster would have the minimum dynamic energy consumption. However, deriving such a balanced solution involves very high complexity and it might not be feasible for most practical cases. Therefore, a good option is to use a polynomial-time algorithm based on load balancing, like the Largest Task First (LTF) strategy [109] (further details in Section 8.1.2), or our own extension of LTF, the Double Largest Task First (DLTF) strategy (further details in Section 8.1.3).

For a given cluster, once the tasks are assigned onto the cores in the cluster, it is now necessary to apply a DVFS scheduling policy that decides the voltage of the cluster and the frequencies of the cores for execution. For performance-constrained applications or real-time tasks assigned to a cluster, the *simplest* and *most intuitive* policy, denoted as the Single Frequency Approximation (SFA) scheme, is to use a single voltage and frequency for execution during the entire hyper-period (as opposed to a technique that uses different DVFS levels at different points in time), particularly, the lowest voltage and frequency that satisfies the timing constraints of all the cores in the cluster (further details in Section 8.1.4). A similar alternative (when the hardware platform allows it), denoted as the Single Voltage Approximation (SVA) scheme, would be to also use a single voltage for execution during the entire hyper-period (particularly, the same single voltage as in SFA), and independently choose the frequency of every core (kept constant during the entire hyper-period), such that the frequency on each core is set to the lowest value that satisfies the timing constraints of all its tasks, but possibly having different execution frequencies for different cores in the same cluster (further details in Section 8.1.5).

Combining DLTF with either SFA or SVA (referred to as DLTF-SFA and DLTF-SVA, respectively) will not derive optimal solutions in terms of peak power reduction or energy minimization. However, they are two *simple* and *practical* solutions that significantly reduce the overheads for changing the supply voltage of the clusters and the frequencies of cores, as neither SFA nor SVA require voltage or frequency changes at runtime. Furthermore, when using SFA, any uni-core DPM technique can be adopted individually in each core without additional effort, because SFA does not require any DVFS alignment between cores. Similarly, given that when using SVA all the cores are executing tasks at all times, SVA does not require any DPM technique to reduce the energy consumption for idling. However, the worst-case performance of DLTF-SFA and DLTF-SVA in terms of peak power reduction and energy efficiency remains an open problem.

Therefore, motivated by the above discussions, for performance-constrained applications or real-time tasks that are already assigned to a specific cluster (or for systems with a global supply voltage), this chapter presents and theoretically analyzes the worst-case behavior (in terms of energy and peak power efficiency) of two *simple* and *practical* polynomial-time strategies, particularly, DLTF-SFA and DLTF-SVA, which use DLTF for task partitioning and SFA or SVA to decide the voltage of the clusters and the frequencies of the cores [68, 69, 70, 71]. For the theoretical analysis, we compare both schemes against the optimal energy and peak power solutions, especially, for the state-of-the-art designs that only have a few cores inside every cluster.

8.1.1 Problem Definition

In this chapter, we focus on an individual cluster/island. Among all the periodic performance-constrained/real-time tasks that have to be executed on the chip, we assume that different sets of tasks are already assigned to a specific cluster, such that every individual cluster has to execute the specific tasks that are assigned to it. Therefore, for simplicity of presentation, in this chapter (and not on other chapters of this dissertation) we redefine two symbols introduced in Chapter 3. Specifically, instead of considering that R represents the total number of tasks to be executed on the entire chip, in this chapter we assume that R represents the total number of tasks to be executed on a specific (arbitrary) cluster. Similarly, instead of considering that M represents the total number of cores in the entire chip and M_k^{cluster} represents the total number of cores in cluster k, in this chapter we assume that M represents the total number of cores in a specific (arbitrary) cluster.

Therefore, for R periodic performance-constrained/real-time tasks that are assigned to a cluster/island, in this chapter we present a practical solution, that partitions the R tasks assigned to the cluster onto the M cores in the cluster, and then applies a simple DVFS schedule, such that the energy consumption in the voltage island is minimized and the peak power consumption is reduced. Particularly, we consider that the tasks are partitioned using the DLTF strategy, and that the DVFS scheduling policy is either SFA or SVA, such that these combinations are denoted as DLTF-SFA and DLTF-SVA, respectively. More importantly, we theoretically analyze the approximation factor (i.e., the worst-case behavior) of these two approaches both for energy minimization and for peak power reduction, against the optimal task partition, optimal DVFS schedule, and optimal DPM schedule, i.e., against the *optimal solution*, for each case, defined $\text{AF}_{\text{DLTF-SFA}}^{\text{energy}}$ and $\text{AF}_{\text{DLTF-SFA}}^{\text{peak power}}$ for DLTF-SFA, respectively, and $\text{AF}_{\text{DLTF-SVA}}^{\text{energy}}$ and $\text{AF}_{\text{DLTF-SVA}}^{\text{peak power}}$ for DLTF-SVA, respectively.

The approximation factor of DLTF-SFA for energy minimization can be computed as shown in Equation (8.1), where E_{OPT}^* represents the optimal energy consumption for the optimal solution (i.e., the optimal task partition, the optimal DVFS schedule, and the optimal DPM schedule) during a hyper-period, $E_{\text{SFA}}^{\text{DLTF}}$ represents the energy consumption during a hyper-period for partitioning tasks with DLTF and using SFA to decide the DVFS levels on individual clusters, and $E_{\downarrow}^*$ represents a lower bound for the optimal energy consumption during a hyper-period.

$$\text{AF}_{\text{DLTF-SFA}}^{\text{energy}} = \max \frac{E_{\text{SFA}}^{\text{DLTF}}}{E_{\text{OPT}}^*} \leq \max \frac{E_{\text{SFA}}^{\text{DLTF}}}{E_{\downarrow}^*} \tag{8.1}$$

Similarly, the approximation factor of DLTF-SFA for peak power reduction can be computed as shown in Equation (8.2), where $\hat{P}_{\text{OPT}}^*$ represents the optimal peak power consumption for the optimal solution (i.e., the optimal task partition, the optimal DVFS schedule, and the optimal DPM schedule), $\hat{P}_{\text{SFA}}^{\text{DLTF}}$ represents the peak power consumption for partitioning tasks with DLTF and using SFA to decide the DVFS levels on individual clusters, and $\hat{P}_{\downarrow}^*$ represents a lower bound for the optimal peak power consumption. Given that E_{OPT}^* and $\hat{P}_{\text{OPT}}^*$ cannot be easily obtained, in the analysis we use their lower bounds $E_{\downarrow}^*$ and $\hat{P}_{\downarrow}^*$, that should not too far away from E^*_{OPT} and $\hat{P}^*_{\text{OPT}}$, respectively.

$$\text{AF}^{\text{peak power}}_{\text{DLTF-SFA}} = \max \frac{\hat{P}^{\text{DLTF}}_{\text{SFA}}}{\hat{P}^*_{\text{OPT}}} \leq \max \frac{\hat{P}^{\text{DLTF}}_{\text{SFA}}}{\hat{P}^*_{\downarrow}} \tag{8.2}$$

The approximation factors for DLTF-SVA can be computed in a similar way, as shown in Equation (8.3) and Equation (8.4)

$$\text{AF}^{\text{energy}}_{\text{DLTF-SVA}} = \max \frac{E^{\text{DLTF}}_{\text{SVA}}}{E^*_{\text{OPT}}} \leq \max \frac{E^{\text{DLTF}}_{\text{SVA}}}{E^*_{\downarrow}} \tag{8.3}$$

$$\text{AF}^{\text{peak power}}_{\text{DLTF-SVA}} = \max \frac{\hat{P}^{\text{DLTF}}_{\text{SVA}}}{\hat{P}^*_{\text{OPT}}} \leq \max \frac{\hat{P}^{\text{DLTF}}_{\text{SVA}}}{\hat{P}^*_{\downarrow}} \tag{8.4}$$

As briefly mentioned above, when we talk about energy minimization, the *optimal task partition for energy minimization* is defined as a task partitioning solution that results in the minimum energy consumption when used in combination with the optimal DVFS and DPM schedule for energy minimization, and obtaining such a task partition is an $\mathcal{NP}$-hard problem [109]. Similarly, when we talk about peak power reduction, the *optimal task partitioning for peak power reduction* is defined as a task partitioning solution that results in the minimum peak power consumption when used in combination with the optimal DVFS and DPM schedule for peak power reduction. Later, in Section 8.2, we show that both optimal task partitions are in fact equivalent for the lower bounds of energy and peak power consumption. Therefore, for simplicity of presentation, we use a single notation for *both* cases, such the optimal task partition for energy minimization and peak power reduction results in task sets $\{\mathbf{S}^*_1, \mathbf{S}^*_2, \ldots, \mathbf{S}^*_M\}$. Without loss of generality, we can assume that task set $\mathbf{S}^*_i$ is assigned on core i, and we define its *cycle utilization* when running on a core of type q as $w^*_{q,i} = \sum_{\tau_n \in \mathbf{S}^*_i} \frac{e_{q,n}}{d_n}$, with unit $\frac{\text{cycles}}{\text{second}}$. Given that in this chapter we focus on homogeneous systems, we can omit parameter q from this notation for simplicity of presentation, such that $w^*_{q,i}$ simply becomes w^*_i. By defining $w^*_0 = 0$ for notational purposes and without loss of generality, the task sets of the optimal task partition are ordered such that $0 = w^*_0 \leq w^*_1 \leq w^*_2 \leq \cdots \leq w^*_M$.

For the power model, we use the expressions from Equation (3.2) and Equation (3.3) shown in Chapter 3.3.

$$P_{\text{core}}\left(f_{\text{cluster}}, f\right) = \alpha \cdot {f_{\text{cluster}}}^{\gamma-1} \cdot f + \beta \cdot f_{\text{cluster}} + \kappa \tag{3.2 revisited}$$

$$P_{\text{core}}\left(f\right) = \alpha \cdot f^{\gamma} + \beta \cdot f + \kappa \tag{3.3 revisited}$$

Similarly, for the energy model, we use the expressions from Equation (3.5), Equation (3.6), and Equation (3.7), presented in Chapter 3.4 .

$$E_{\text{core}}\left(f_{\text{cluster}}, f\right) = \left(\alpha \cdot {f_{\text{cluster}}}^{\gamma-1} \cdot f + \beta \cdot f_{\text{cluster}} + \kappa\right) \frac{\Delta c}{f} \tag{3.5 revisited}$$

$$E_{\text{core}}\left(f\right) = \left(\alpha \cdot f^{\gamma} + \beta \cdot f + \kappa\right) \frac{\Delta c}{f} \tag{3.6 revisited}$$

$$f_{\text{crit}} = \sqrt[\gamma]{\frac{\kappa}{\left(\gamma - 1\right)\alpha}} \tag{3.7 revisited}$$

When a core finishes executing all the workload available in its ready queue, the core has to wait until a new task instance arrives. During this waiting interval, the core can remain idle (i.e., clock-gated), consuming $P^{\text{idle}}_{\text{core}}\left(f_{\text{cluster}}\right) = \beta \cdot f_{\text{cluster}} + \kappa$. On the other hand, the core can also enter a low-power mode (e.g., sleep, power-gated, etc.) that consumes $\kappa^{\text{sleep}} \geq 0$ power. Given that the optimal solutions for peak power reduction or energy minimization cannot optimize for κ^{sleep} (i.e., κ^{sleep} is an offset that is always present), without loss of generality, we can transfer the power consumption κ^{sleep} to some other part of the system, such that we can set $P_{\text{core}}\left(f_{\text{cluster}}, f\right)$ to $P_{\text{core}}\left(f_{\text{cluster}}, f\right) - \kappa^{\text{sleep}}$, set $P_{\text{core}}\left(f\right)$ to $P_{\text{core}}\left(f\right) - \kappa^{\text{sleep}}$, and set $P^{\text{idle}}_{\text{core}}\left(f_{\text{cluster}}\right)$ to $P^{\text{idle}}_{\text{core}}\left(f_{\text{cluster}}\right) - \kappa^{\text{sleep}}$, such that we can disregard the effect of the power consumption of a core in a low-power mode. In this way, since not even the optimal solution can optimize for κ^{sleep}, we only focus on the *effective optimization region* and we avoid possible masking problems in systems with large κ^{sleep} values.

For the analysis in Section 8.3 and Section 8.4, we consider continuous DVFS levels in the range of $[F_{\min}, F_{\max}]$, such that $F_{\min} = F^{\text{type}}_{q,1}$ and $F_{\max} = F^{\text{type}}_{q,\hat{F}^{\text{type}}_q}$, where there is only one possible type of core q since in this entire chapter we focus on homogeneous systems. These results are then extended in Section 8.6 by considering discrete voltage and frequency pairs $\left\{F^{\text{type}}_{q,1}, F^{\text{type}}_{q,2}, \ldots, F^{\text{type}}_{q,\hat{F}^{\text{type}}_q}\right\}$, as defined in Chapter 3.2.

8.1.2 Largest Task First (LTF) Scheme

A good and widely used algorithm for task partitioning is the LTF scheme [109], which is a *Worst-Fit-Decreasing* heuristic algorithm. The LTF scheme is mainly a reformulation of the Longest Processing Time (LPT) algorithm [20] for the *Makespan* problem. Particularly, LTF partitions the R periodic tasks $\{\tau_1, \tau_2, \ldots, \tau_R\}$ into M groups of disjoint task sets, denoted as $\left\{\mathbf{S}^{\text{LTF}}_1, \mathbf{S}^{\text{LTF}}_2, \ldots, \mathbf{S}^{\text{LTF}}_M\right\}$. For a given type of core q, the corresponding cycle utilizations of these task sets are $0 = w^{\text{LTF}}_{q,0} \leq w^{\text{LTF}}_{q,1} \leq w^{\text{LTF}}_{q,2} \leq \cdots \leq w^{\text{LTF}}_{q,M}$, ordered without loss of generality. Given that in this chapter we focus on homogeneous systems, we can omit parameter q from this notation for simplicity of presentation, resulting in $0 = w^{\text{LTF}}_0 \leq w^{\text{LTF}}_1 \leq w^{\text{LTF}}_2 \leq \cdots \leq w^{\text{LTF}}_M$. For completeness, a pseudo-code for LTF is presented in Algorithm 6, with worst-case time complexity $O\left(R\left(\log R + \log M\right) + M\right)$ [109]. Throughout this chapter, we implicitly assume that for a homogeneous system composed by cores of type q, it holds that $w^{\text{LTF}}_M \leq F^{\text{type}}_{q,\hat{F}^{\text{type}}_q}$, as otherwise LTF does not derive a feasible solution under EDF [59] (i.e. there would be no guarantee that all the tasks assigned to the cluster can meet their timing constraints).

Algorithm 6 Largest Task First (LTF) scheme

```
Input: Tasks $\{\tau_1, \tau_2, \ldots, \tau_R\}$;
Output: Task sets $\left\{\mathbf{S}^{\text{LTF}}_1, \mathbf{S}^{\text{LTF}}_2, \ldots, \mathbf{S}^{\text{LTF}}_M\right\}$;
1: Sort all tasks in a non-increasing order of their cycle utilizations;
2: for all $M$ task sets (i.e., for all $i = 1, 2, \ldots, M$) do
3:     $\mathbf{S}^{\text{LTF}}_i \leftarrow \emptyset$; $w^{\text{LTF}}_i \leftarrow 0$; {Initialize all task sets to be empty, with zero cycle utilization}
4: end for
5: for all $R$ tasks (i.e., for all $n = 1, 2, \ldots, R$) do
6:     Find the task set $\mathbf{S}^{\text{LTF}}_i$ with the smallest cycle utilization $w^{\text{LTF}}_i$;
7:     $\mathbf{S}^{\text{LTF}}_i \leftarrow \mathbf{S}^{\text{LTF}}_i + \{\tau_n\}$; {Assign task $\tau_n$ to task set $\mathbf{S}^{\text{LTF}}_i$}
8: end for
9: Re-order all task sets decreasingly according to their cycle utilization;
10: return $\left\{\mathbf{S}^{\text{LTF}}_1, \mathbf{S}^{\text{LTF}}_2, \ldots, \mathbf{S}^{\text{LTF}}_M\right\}$;
```

The LTF scheme has a few properties inherited from the LPT algorithm [20] for the *Makespan* problem. Consider the optimal task partition, previously defined in Section 8.1.1 as $\{\mathbf{S}^*_1, \mathbf{S}^*_2, \ldots, \mathbf{S}^*_M\}$, for which w^*_M is the maximum cycle utilization among all cores in the cluster for the optimal solution. Naturally, the *total cycle utilization* of all tasks is constant regardless of how they are partitioned, and therefore it holds that

$$\sum_{i=1}^{M} w^{\text{LTF}}_i = \sum_{i=1}^{M} w^*_i.$$

Moreover, if after partitioning the tasks with LTF we have that there is *only one* task in resulting task set $\mathbf{S}^{\text{LTF}}_M$, then, given that the cycle utilization of the task in set $\mathbf{S}^{\text{LTF}}_M$ is the lower bound of the maximum cycle utilization in any possible task partition (including the optimal task partition), it holds that

$$w^{\text{LTF}}_M \leq w^*_M.$$

Furthermore, if after partitioning the tasks with LTF we have that there are *at least two* tasks in resulting task set $\mathbf{S}^{\text{LTF}}_M$, then the property presented in Equation (8.5) holds, where θ_{LTF} is the approximation factor of the LTF scheme in terms of task partitioning, due to the approximation factor of the LPT algorithm [20] for the *Makespan* problem. Finally, under this same condition, the work in [109] also proved the property presented in Equation (8.6). Figure 8.1 and Figure 8.2 illustrate these properties of the LTF scheme.

$$\frac{w_M^{\text{LTF}}}{w_M^*} \leq \theta_{\text{LTF}} = \frac{4}{3} - \frac{1}{3M} \tag{8.5}$$

$$\frac{w_1^{\text{LTF}}}{w_M^{\text{LTF}}} \geq \frac{1}{2} \tag{8.6}$$

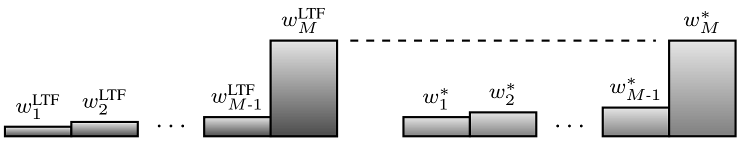
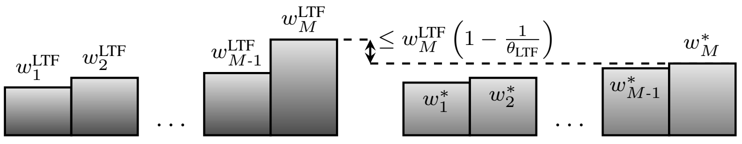


Figure 8.1: Example of the cycle utilization relations of the LTF scheme when $w_M^{\text{LTF}} \leq w_M^*$.



w_1^{LTF}
w_2^{LTF}
$w_{M\text{-}1}^{\text{LTF}}$
w_M^{LTF}
$\leq w_M^{\text{LTF}}\left(1 - \frac{1}{\theta_{\text{LTF}}}\right)$
w_1^*
w_2^*
$w_{M\text{-}1}^*$
w_M^*

Figure 8.2: Example of the cycle utilization relations of the LTF scheme when there are at least two tasks in task set $\mathbf{S}_M^{\text{LTF}}$ and $w_M^* < w_M^{\text{LTF}}$.

8.1.3 Double Largest Task First (DLTF) Scheme

This section presents our task partitioning scheme for idle energy reduction, called DLTF, well-suited to apply in combination with SFA or SVA. Task partitioning scheme DLTF considers LTF as an initial solution, which means that after executing LTF we have that $\left\{\mathbf{S}_1^{\text{DLTF}} = \mathbf{S}_1^{\text{LTF}}, \mathbf{S}_2^{\text{DLTF}} = \mathbf{S}_2^{\text{LTF}}, \ldots, \mathbf{S}_M^{\text{DLTF}} = \mathbf{S}_M^{\text{LTF}}\right\}$, with the corresponding cycle utilizations $0 = w_0^{\text{DLTF}} \leq w_1^{\text{DLTF}} = w_1^{\text{LTF}} \leq w_2^{\text{DLTF}} = w_2^{\text{LTF}} \leq \cdots \leq w_M^{\text{DLTF}} = w_M^{\text{LTF}}$, ordered without loss of generality. Then, in order to reduce the energy consumption for idling under SFA and SVA, we regroup the tasks, such that as many cores as possible are put into low-power mode during the entire hyper-period, and this is achieved without increasing the energy consumption for execution. For such a purpose, we define the auxiliary cycle utilization $w_{\text{max}}^{\text{DLTF}}$ as $\max\left\{f_{\text{crit}}, w_M^{\text{LTF}}\right\}$, which is used as a maximum cycle utilization for the task regrouping.

The regrouping is an iterative procedure, in which we migrate tasks from a *source task set*, denoted as $\mathbf{S}_i^{\text{DLTF}}$, to a *destination task set*, denoted as $\mathbf{S}_j^{\text{DLTF}}$. We iterate through all the *source task sets* (increasingly with respect to their cycle utilizations), through all the tasks inside every *source task set*, and through all the possible *destination task sets* (decreasingly with respect to their cycle utilizations) for every task iterated from the *source task set*. Particularly, we start by choosing the *source task set* $\mathbf{S}_i^{\text{DLTF}}$, such that its cycle utilization w_i^{DLTF} is the smallest cycle utilization among all task sets. Given that from LTF the task sets are ordered increasingly with respect to their cycle utilizations, the first chosen *source task set* will be $\mathbf{S}_1^{\text{DLTF}}$ (i.e., $i = 1$). We then iterate (with no specific order) through all tasks τ_n inside *source task set* $\mathbf{S}_i^{\text{DLTF}}$, where for the given type of core q for the homogeneous system, the cycle utilization of τ_n is computed as $\frac{e_{q,n}}{d_n}$. For every τ_n, we further iterate through the *destination task sets* $\mathbf{S}_j^{\text{DLTF}}$ for all $j = M, M-1, \ldots, i+2, i+1$, i.e., we start from the task set with the highest cycle utilization and we stop just before reaching the *source task set* $\mathbf{S}_i^{\text{DLTF}}$. In case τ_n fits inside the *destination task set* $\mathbf{S}_j^{\text{DLTF}}$ without exceeding $w_{\text{max}}^{\text{DLTF}}$, i.e., if it holds that $\frac{e_{q,n}}{d_n} + w_j^{\text{DLTF}} \leq w_{\text{max}}^{\text{DLTF}}$, then we migrate τ_n to the *destination task set* $\mathbf{S}_j^{\text{DLTF}}$ and update the value of w_j^{DLTF} accordingly. In case τ_n does not fit inside the *destination task set* $\mathbf{S}_j^{\text{DLTF}}$ without exceeding $w_{\text{max}}^{\text{DLTF}}$, then we update j to $j-1$ and try again with a new *destination task set*, until j reaches the value of i, point in which we move to the next task inside the *source task set* $\mathbf{S}_i^{\text{DLTF}}$. This process is repeated for all the tasks inside the *source task set* $\mathbf{S}_i^{\text{DLTF}}$, and then the value of i is updated to $i+1$, until i reaches M.

Note that, since before migrating a task to a specific *destination task set* all other *destination task sets* with larger cycle utilizations have already been unsuccessfully tested, this means that this specific task will be migrated to the *destination task set* with the largest cycle utilization in which the task can feasible fit. Therefore, considering that migrating a task more than once under this policy will always fail, once a task is migrated from a *source task set* to a *destination task set*, the task can be marked as *migrated*, such that DLTF does not attempt to migrate it again. Furthermore, given that we do not migrate a task to a *destination task set* unless $w_{\max}^{\text{DLTF}}$ is not exceeded, then the maximum cycle utilization among all task sets either remains constant (in case $f_{\text{crit}} \leq w_M^{\text{LTF}}$), or it can grow, but never beyond f_{crit} (in case $f_{\text{crit}} > w_M^{\text{LTF}}$). Because of this reason, DLTF does not increase the energy consumption for execution in the cluster under either SFA or SVA, as the voltage of the cluster and the frequency of the core with the highest cycle utilization is either unchanged or set to more energy efficient values (more details in Section 8.1.4 and Section 8.1.5). Finally, the task sets that had a high cycle utilization after partitioning tasks with LTF will now have a higher cycle utilization after the regrouping procedure, and vice-versa. Most importantly, any core with a resulting cycle utilization after regrouping equal to 0 can be further put into low-power mode for the entire hyper-period, thus saving energy for idling. Consequently, the resulting number of cores with cycle utilization larger than 0, i.e., the number of cores that remain active after regrouping, is defined as $M^{\neq 0}$. A pseudo-code for DLTF is presented in Algorithm 7, with worst-case time complexity $O\left(M^2 R\right)$ after LTF is executed. Figure 8.3 shows a brief example comparing an initial task partition obtained by applying LTF and the resulting task partition after the regrouping procedure of DLTF, both cases using SFA as their DVFS schedule.

Algorithm 7 Double Largest Task First (DLTF) scheme

```
Input: Tasks $\{\tau_1, \tau_2, \ldots, \tau_R\}$;
Output: Task sets $\left\{\mathbf{S}_1^{\text{DLTF}}, \mathbf{S}_2^{\text{DLTF}}, \ldots, \mathbf{S}_M^{\text{DLTF}}\right\}$;
1: Execute LTF for tasks $\{\tau_1, \tau_2, \ldots, \tau_R\}$; {Algorithm 6}
2: $w_{\max}^{\text{DLTF}} \leftarrow \max\left\{f_{\text{crit}}, w_M^{\text{LTF}}\right\}$; {Set the value of the maximum cycle utilization for the task regrouping}
3: $\left\{\mathbf{S}_1^{\text{DLTF}}, \mathbf{S}_2^{\text{DLTF}}, \ldots, \mathbf{S}_M^{\text{DLTF}}\right\} \leftarrow \left\{\mathbf{S}_1^{\text{LTF}}, \mathbf{S}_2^{\text{LTF}}, \ldots, \mathbf{S}_M^{\text{LTF}}\right\}$; {DLTF considers LTF as an initial solution}
4: for all source task sets, increasingly (i.e., for all $i = 1, 2, \ldots, M-1$) do
5:   for all tasks inside the current source task set (i.e., for all $\tau_n \in \mathbf{S}_i^{\text{DLTF}}$) do
6:     for all destination task sets, decreasingly (i.e., for all $j = M, M-1, \ldots, i+2, i+1$) do
         {Check if task $\tau_n$ fits inside the current destination task set without exceeding $w_{\max}^{\text{DLTF}}$}
7:       if $\frac{e_{q,n}}{d_n} + w_j^{\text{DLTF}} \leq w_{\max}^{\text{DLTF}}$ then
8:         $\mathbf{S}_j^{\text{DLTF}} \leftarrow \mathbf{S}_j^{\text{DLTF}} + \{\tau_n\}$; {Assign task $\tau_n$ to destination task set $\mathbf{S}_j^{\text{DLTF}}$}
9:         Remove $\tau_n$ from source task set $\mathbf{S}_i^{\text{DLTF}}$;
10:        break for loop $j$; {If a task is migrated, we continue with the next task in the source task set}
11:      end if
12:    end for
13:  end for
14: end for
15: return $\left\{\mathbf{S}_1^{\text{DLTF}}, \mathbf{S}_2^{\text{DLTF}}, \ldots, \mathbf{S}_M^{\text{DLTF}}\right\}$;
```

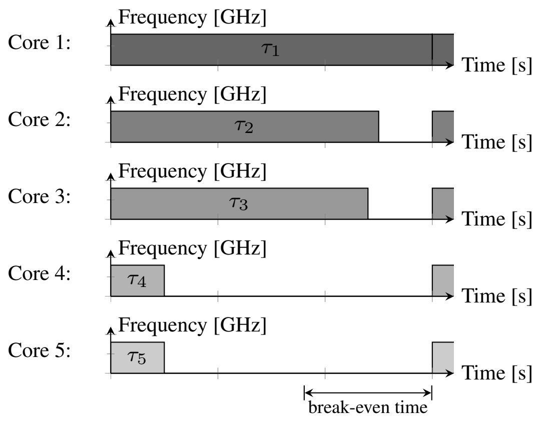
Just like when using LTF, given that the *total cycle utilization* of all tasks is constant for all possible task partitions, under DLTF it holds that

$$\sum_{i=1}^{M} w_i^{\text{DLTF}} = \sum_{i=1}^{M} w_i^{\text{LTF}} = \sum_{i=1}^{M} w_i^{*}. \tag{8.7}$$

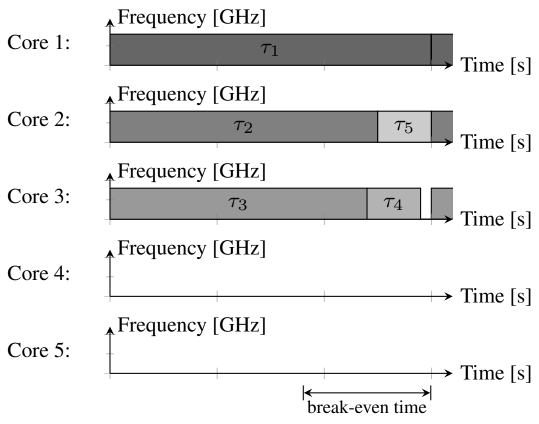
Furthermore, if $w_M^{\text{LTF}} \geq f_{\text{crit}}$, given that $w_{\max}^{\text{DLTF}}$ is set to w_M^{LTF} in this case, then no task can be migrated to *destination task set* $\mathbf{S}_M^{\text{DLTF}}$. Therefore, in this case we have that $\mathbf{S}_M^{\text{DLTF}} = \mathbf{S}_M^{\text{LTF}}$, with $w_M^{\text{DLTF}} = w_M^{\text{LTF}}$. Moreover, in regards to the properties of LTF that can be inherited to DLTF, the only property of LTF that does not hold in DLTF for this case (due to the regrouping of tasks) is Equation (8.6). In other words, when $w_M^{\text{LTF}} \geq f_{\text{crit}}$, if after partitioning tasks with DLTF there is *only one* task inside set $\mathbf{S}_M^{\text{DLTF}}$, then it holds that

$$w_M^{\text{DLTF}} \leq w_M^{*},$$

given that the cycle utilization of the task inside set $\mathbf{S}_M^{\text{DLTF}}$ is the lower bound of the maximum cycle utilization in any possible task partition (including the optimal task partition). For this same case in which $w_M^{\text{LTF}} \geq f_{\text{crit}}$,



(a) Example: LTF and SFA



(b) Example: DLTF and SFA

Figure 8.3: Brief example comparing an initial task partition obtained by applying LTF and the resulting task partition after the regrouping procedure of DLTF, both cases using SFA as their DVFS schedule in this example. All tasks share a common arrival, deadline, and period (i.e., they are frame-based tasks). In LTF, core 2 and core 3 have to remain idle after they finish the execution of a task instance, since their idle times are shorter than the break-even time. Under DLTF, the idle energy consumption of core 2 and core 3 is reduced, while core 4 and core 5 are always kept in a low-power mode.

if after partitioning task with DLTF there are *at least two* tasks inside set $\mathbf{S}_M^{\text{DLTF}}$, then it holds that

$$\frac{w_M^{\text{DLTF}}}{w_M^*} \leq \theta_{\text{LTF}} = \frac{4}{3} - \frac{1}{3M}. \tag{8.8}$$

When $w_M^{\text{LTF}} < f_{\text{crit}}$, given that in this case, in comparison with the original $\mathbf{S}_M^{\text{LTF}}$, we might add some tasks into task set $\mathbf{S}_M^{\text{DLTF}}$, then most likely we will have a higher cycle utilization in task set $\mathbf{S}_M^{\text{DLTF}}$ than in task set $\mathbf{S}_M^{\text{LTF}}$. Nevertheless, in this case the value of w_M^{DLTF} will also remain below f_{crit}, such that $w_M^{\text{LTF}} \leq w_M^{\text{DLTF}} < f_{\text{crit}}$, and all the cores in the cluster should be executed at f_{crit} in order to minimize the energy consumption for execution. Clearly, given that in this case the optimal solution (i.e., the optimal task partition, the optimal DVFS schedule, and the optimal DPM schedule) will consume the same energy for execution, resulting in an approximation factor of 1, *for the analysis of the approximation factor we will only focus in the case in which* $w_M^{LTF} \geq f_{crit}$.

8.1.4 Single Frequency Approximation (SFA) Scheme

The SFA scheme is the *simplest* and *most intuitive* policy for DVFS scheduling of performance-constrained applications or real-time tasks. Under SFA, all the cores in a cluster use a single voltage and frequency for execution during the entire hyper-period, instead of dynamically changing the DVFS levels at different points in time. Particularly, SFA uses the lowest DVFS levels that satisfy the timing constraints of all the tasks assigned to the cores inside a cluster. After the task partitioning is completed, SFA has linear time complexity $O(M)$, which comes only from evaluating which core in the cluster has the highest cycle utilization. Namely, given that different tasks are assigned to different cores, the cores inside a cluster might have different frequency requirements in order to meet the timing constraints. Therefore, the core with the highest cycle utilization (i.e., the core with the highest frequency demands) inside a cluster will determine the required execution frequency of SFA for the entire cluster. Nevertheless, in order to be energy efficient, SFA will never select DVFS levels that are lower than the associated critical frequency. Even though SFA is not an optimal DVFS scheduling policy for energy minimization, it significantly reduces the management overheads for changing the DVFS levels on the clusters, as it does not require voltage or frequency changes at runtime. Furthermore, given that all the cores in a cluster execute at a single frequency and no frequency alignment for DVFS between cores is needed, any uni-core DPM technique for reducing the energy consumption for idling can be easily incorporated individually on every core. Figure 8.4 presents a brief example of a cluster with four cores using SFA as the DVFS scheduling policy.

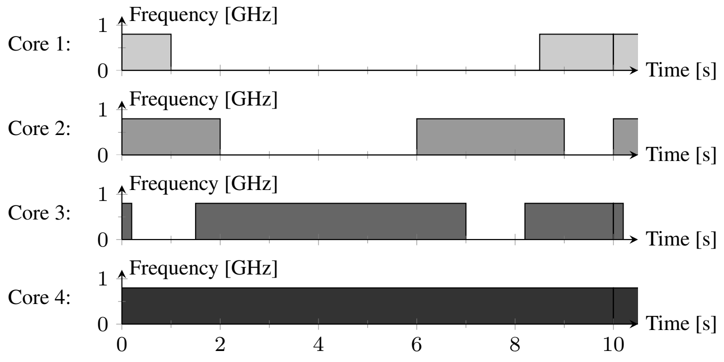


Figure 8.4: SFA example for a cluster with four cores. The hyper-period of all tasks is 10 seconds. In order to meet all deadlines, the frequency demands of the cores are 0.2 GHz, 0.4 GHz, 0.6 GHz, and 0.8 GHz. Hence, the single frequency of SFA is set to 0.8 GHz. In order to save energy, cores individually enter a low-power mode when there is no workload on their ready queues.

The SFA scheme has been adopted by several researchers in the past (e.g., by [15] when the tasks do not complete earlier than the estimated worst-case execution times, or by [65]). Moreover, the applicability of SFA with slack reclamation in order to deal with early completion of tasks can be found in [15]. SFA is indeed a good strategy when the workload is *perfectly* balanced throughout the cores in a cluster (i.e., when all the cores are assigned with similar cycle utilizations). Contrarily, when the cycle utilizations of the cores inside a cluster are *skewed* (i.e., one core has a high cycle utilization while all others have a very low cycle utilization), then SFA might consume much more energy than the optimal solution, especially when the number of cores in the cluster grows. This comes from the cases in which cores with light cycle utilizations are forced to execute at higher DVFS levels than they require in order to meet their timing constraints.

In order to meet the timing constraints of all tasks in the cluster while attempting to minimize the energy consumption, the frequencies used by the SFA scheme are (1) f_{crit} in case that $w_M^{\text{DLTF}} \leq f_{\text{crit}}$, and (2) w_M^{DLTF} otherwise. The worst-case peak power consumption on the cluster occurs when all cores execute tasks simultaneously, e.g., at the beginning of a hyper-period when at least one task in each core has arrival time zero, as shown in the examples in Figure 8.3 and Figure 8.4. Hence, if from Equation (3.3) the power consumption on a core executing at frequency f is computed as $P_{\text{core}}(f) = \alpha \cdot f^{\gamma} + \beta \cdot f + \kappa$, then the peak power consumption on the cluster for combining DLTF and SFA (i.e., DLTF-SFA), denoted as $\hat{P}_{\text{SFA}}^{\text{DLTF}}$ and expressed as seen in Equation (8.9), depends mostly on the number of active cores $M^{\neq 0}$ and the frequencies selected by SFA. Furthermore, in case $\sum_{i=1}^{M} w_i^{\text{DLTF}} \leq f_{\text{crit}}$, then DLTF will partition all the tasks into a single core, and this becomes a single core DVFS problem. For such a case, from Equation (3.5) and Figure 3.5, we know that running at slow frequencies might consume excessive energy due to the leakage and independent power consumptions, and therefore all tasks are assigned to a single core that is executed at frequency f_{crit}.

$$\hat{P}_{\text{SFA}}^{\text{DLTF}} = \begin{cases} \alpha \cdot {f_{\text{crit}}}^{\gamma} + \beta \cdot f_{\text{crit}} + \kappa & \text{if } \sum_{i=1}^{M} w_i^{\text{DLTF}} \leq f_{\text{crit}} \\ M^{\neq 0} \left(\alpha \cdot {f_{\text{crit}}}^{\gamma} + \beta \cdot f_{\text{crit}} + \kappa\right) & \text{if } w_M^{\text{DLTF}} \leq f_{\text{crit}} \\ M^{\neq 0} \left(\alpha \cdot {w_M^{\text{DLTF}}}^{\gamma} + \beta \cdot w_M^{\text{DLTF}} + \kappa\right) & \text{otherwise} \end{cases} \tag{8.9}$$

From Equation (8.9) and by looking at the example in Figure 8.3, we can observe that under SFA, partitioning tasks with DLTF can drastically reduce the peak power consumption in comparison to LTF. This occurs thanks to the regrouping procedure that reduces the number of active cores from M to $M^{\neq 0}$.

With respect to energy consumption, if from Equation (3.6) the energy consumption on a core executing at frequency f is computed as $E_{\text{core}}(f) = (\alpha \cdot f^{\gamma} + \beta \cdot f + \kappa) \frac{\Delta c}{f}$, and by considering that the total amount of computation Δc to be finished in a hyper-period by core i is $D \cdot w_i^{\text{DLTF}}$, then we have that the energy consumption on core i running at frequency f during a hyper-period is computed as

$$E_{\text{core}_i}^{\text{DLTF}}(f) = D\left(\alpha \cdot f^{\gamma} + \beta \cdot f + \kappa\right) \frac{w_i^{\text{DLTF}}}{f}.$$

Therefore, considering all the cores in the cluster and according to the frequencies selected by SFA, the energy consumption in the cluster for DLTF-SFA during a hyper-period, denoted as $E_{\text{SFA}}^{\text{DLTF}}$, can be expressed as seen in Equation (8.10).

$$E_{\text{SFA}}^{\text{DLTF}} = \begin{cases} D\left(\alpha \cdot {f_{\text{crit}}}^{\gamma} + \beta \cdot f_{\text{crit}} + \kappa\right) \frac{\sum_{i=1}^{M} w_i^{\text{DLTF}}}{f_{\text{crit}}} & \text{if } w_M^{\text{DLTF}} \leq f_{\text{crit}} \\ D\left(\alpha \cdot {w_M^{\text{DLTF}}}^{\gamma} + \beta \cdot w_M^{\text{DLTF}} + \kappa\right) \frac{\sum_{i=1}^{M} w_i^{\text{DLTF}}}{w_M^{\text{DLTF}}} & \text{otherwise} \end{cases} \tag{8.10}$$

From Equation (8.10) we can see that the energy consumption of DLTF-SFA only depends on the frequency selected by SFA and on the *total cycle utilization* of all tasks, which is constant for all possible task partitions, as shown in Equation (8.7). That is, the total energy consumption will depend on the cycle utilization requirements of the core with the largest workload; however, how the rest of the tasks are partitioned in the other cores is not relevant in this case. Furthermore, as described in Section 8.1.3, if we have that $w_M^{\text{LTF}} < f_{\text{crit}}$, then it holds that $w_M^{\text{LTF}} \leq w_M^{\text{DLTF}} < f_{\text{crit}}$; while if we have that $w_M^{\text{LTF}} \geq f_{\text{crit}}$, then it holds that $w_M^{\text{DLTF}} = w_M^{\text{LTF}}$.

Therefore, if we assume negligible overhead for entering and leaving a low-power mode (i.e., *negligible overhead for sleeping*), then the total energy consumption for using LTF under SFA is the same as the total energy consumption for using DLTF under SFA. However, when such overheads are non-negligible (i.e., for the practical scenario in real systems), DLTF saves some energy for idling in comparison to plain LTF. Moreover, given that the energy consumption under SFA can be reduced when the highest cycle utilization among all cores is reduced and both LTF and DLTF are worst-fit-decreasing heuristics that attempt to minimize this cycle utilization, we can thus conclude that LTF and DLTF are efficient task partitioning schemes for energy minimization under SFA.

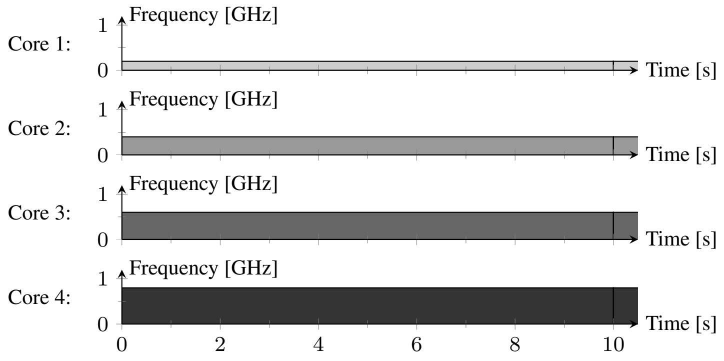
8.1.5 Single Voltage Approximation (SVA) Scheme

The SVA scheme is inspired by the SFA scheme described in Section 8.1.4. In SVA, all the cores in a cluster are also executed using a single voltage for execution during the entire hyper-period, particularly, the same single voltage as in SFA, determined by the core with the highest frequency requirements (i.e., the core assigned with the highest cycle utilization task set w_M^{DLTF}). However, unlike SFA, in SVA (when the hardware platform allows it) the frequency of every core is individually chosen (kept constant during the entire hyper-period), such that the frequency on each core is set to the lowest value that satisfies the timing constraints of all its tasks, but possibly having different execution frequencies for different cores in the same cluster. Hence, all the cores are executing tasks at all times in case all the tasks require their worst-case execution times in order to finish every task instance, and there is no need for the adopted DPM technique to place cores in a low-power mode in order to reduce the energy consumption for idling.

Particularly, as mentioned above, under SVA the voltage of the cluster and the frequencies of the cores are kept constant during the entire hyper-period, i.e., they do not change at runtime. After the task partitioning stage is completed, every core i is assigned with the corresponding task set $\mathbf{S}_i^{\text{DLTF}}$, with cycle utilization w_i^{DLTF}. Thus, under SVA, in order to just meet the timing constraints when using EDF [59] for scheduling tasks in individual cores, the frequency of core i is set to w_i^{DLTF} for all $i = 1, 2, \ldots, M$. In this way, the highest frequency among all cores is w_M^{DLTF}. Naturally, in order to consume less power and to save energy, the voltage of the cluster is set to the minimum available voltage such that frequency w_M^{DLTF} can be stably achieved. The time complexity of SVA is $O(M)$, where M is the number of cores in the cluster and this complexity comes from evaluating the highest cycle utilization among all cores and choosing the individual frequencies of every other core. Figure 8.5 presents a brief example of a cluster with four cores using SVA as the DVFS scheduling policy.

As mentioned above, under SVA, in case all the tasks require their worst-case execution times in order to finish every task instance, then all the cores are always busy such that they just meet their timing constraints. This means that for such a case, the power consumption on every core, and therefore the power consumption in the entire cluster, is constant through the entire hyper-period, resulting in a peak power consumption which is equivalent to the average power consumption. Therefore, if from Equation (3.2) the power consumption on a core executing at frequency f, where the voltage of the cluster is determined by frequency f_{cluster}, is

Figure 8.5: SVA example for a cluster with four cores. The hyper-period of all tasks is 10 seconds. In order to meet all deadlines, the frequency demands of the cores are 0.2 GHz, 0.4 GHz, 0.6 GHz, and 0.8 GHz. The frequency on each core is set according to its demands, and the voltage is set according to 0.8 GHz. All cores are always busy such that they just meet their timing constraints.



computed as $P_{\text{core}}\left(f_{\text{cluster}}, f\right) = \alpha \cdot {f_{\text{cluster}}}^{\gamma-1} \cdot f + \beta \cdot f_{\text{cluster}} + \kappa$, then the power consumption for core i under DLTF-SVA is

$$P_{\text{core}_i}^{\text{DLTF}}\left(f_{\text{cluster}}, f\right) = \alpha \cdot {w_M^{\text{DLTF}}}^{\gamma-1} \cdot w_i^{\text{DLTF}} + \beta \cdot w_M^{\text{DLTF}} + \kappa,$$

such that, considering that there are $M^{\neq 0}$ active cores always consuming leakage and independent power, the peak power consumption on a cluster, defined as $\hat{P}_{\text{SVA}}^{\text{DLTF}}$, is expressed as

$$\hat{P}_{\text{SVA}}^{\text{DLTF}} = \alpha \cdot {w_M^{\text{DLTF}}}^{\gamma-1} \sum_{i=1}^{M} w_i^{\text{DLTF}} + M^{\neq 0}\left(\beta \cdot w_M^{\text{DLTF}} + \kappa\right).$$

In order to consider the worst cases, we have to assume that when $w_M^{\text{DLTF}} \leq f_{\text{crit}}$, after the regrouping with DLTF the voltage of the cluster can be set up to the value associated with f_{crit}. Moreover, as already mentioned in Section 8.1.4, in case that $\sum_{i=1}^{M} w_i^{\text{DLTF}} \leq f_{\text{crit}}$, then all tasks are assigned to a single core that is executed at frequency f_{crit}. Finally, $\hat{P}_{\text{SVA}}^{\text{DLTF}}$ can be computed as shown in Equation (8.11).

$$\hat{P}_{\text{SVA}}^{\text{DLTF}} \leq \begin{cases} \alpha \cdot {f_{\text{crit}}}^{\gamma} + \beta \cdot f_{\text{crit}} + \kappa & \text{if } \sum_{i=1}^{M} w_i^{\text{DLTF}} \leq f_{\text{crit}} \\ \alpha \cdot {f_{\text{crit}}}^{\gamma-1} \sum\limits_{i=1}^{M} w_i^{\text{DLTF}} + M^{\neq 0}\left(\beta \cdot f_{\text{crit}} + \kappa\right) & \text{if } w_M^{\text{DLTF}} \leq f_{\text{crit}} \\ \alpha \cdot {w_M^{\text{DLTF}}}^{\gamma-1} \sum\limits_{i=1}^{M} w_i^{\text{DLTF}} + M^{\neq 0}\left(\beta \cdot w_M^{\text{DLTF}} + \kappa\right) & \text{otherwise} \end{cases} \tag{8.11}$$

Furthermore, if from Equation (3.5) the energy consumption during a hyper-period for a core executing at frequency f, where the voltage of the cluster is determined by frequency f_{cluster}, is computed as $E_{\text{core}}\left(f_{\text{cluster}}, f\right) = \left(\alpha \cdot {f_{\text{cluster}}}^{\gamma-1} \cdot f + \beta \cdot f_{\text{cluster}} + \kappa\right) \frac{\Delta c}{f}$, by considering that the workload to be completed on core i during the hyper-period is $D \cdot w_i^{\text{DLTF}}$, then the energy consumption for core i under DLTF-SVA during a hyper-period is computed as

$$E_{\text{core}_i}^{\text{DLTF}}\left(f_{\text{cluster}}, f\right) = D\left(\alpha \cdot {w_M^{\text{DLTF}}}^{\gamma-1} \cdot w_i^{\text{DLTF}} + \beta \cdot w_M^{\text{DLTF}} + \kappa\right)$$

Therefore, by having the same considerations in regards to f_{crit} as when computing the peak power consumption $\hat{P}_{\text{SVA}}^{\text{DLTF}}$, the total energy consumption on a cluster for DLTF-SVA during a hyper-period, defined as $E_{\text{SVA}}^{\text{DLTF}}$, is expressed as shown in Equation (8.12).

$$E_{\text{SVA}}^{\text{DLTF}} \leq \begin{cases} D\left(\alpha \cdot {f_{\text{crit}}}^{\gamma} + \beta \cdot f_{\text{crit}} + \kappa\right) \frac{\sum_{i=1}^{M} w_i^{\text{DLTF}}}{f_{\text{crit}}} & \text{if } \sum_{i=1}^{M} w_i^{\text{DLTF}} \leq f_{\text{crit}} \\ D\left[\alpha \cdot {f_{\text{crit}}}^{\gamma-1} \sum\limits_{i=1}^{M} w_i^{\text{DLTF}} + M^{\neq 0}\left(\beta \cdot f_{\text{crit}} + \kappa\right)\right] & \text{if } w_M^{\text{DLTF}} \leq f_{\text{crit}} \\ D\left[\alpha \cdot {w_M^{\text{DLTF}}}^{\gamma-1} \sum\limits_{i=1}^{M} w_i^{\text{DLTF}} + M^{\neq 0}\left(\beta \cdot w_M^{\text{DLTF}} + \kappa\right)\right] & \text{otherwise} \end{cases} \tag{8.12}$$

8.2 Lower Bounds

This section provides a lower bound for the optimal energy consumption and for the optimal peak power consumption for periodic performance-constrained applications or real-time tasks, which are needed to obtain the approximation factors introduced in Equations (8.1), (8.2), (8.3), and (8.4). In order to obtain these lower bounds, we start by *unrolling* all the periodic tasks executed in a hyper-period into frame-based real-time tasks, such that *all* instances of *all* tasks (i.e., *all* jobs) arrive at time 0, and have a period and deadline equal to the hyper-period D. Note that this consideration is a special case of periodic tasks (as discussed in Chapter 3.1) that does not affect the energy or peak power consumption of DLTF-SFA of DLTF-SVA, which implies that this approach is not pessimistic but rather a worst case. Furthermore, although neither DLTF-SFA nor DLTF-SVA require any DVFS capabilities at runtime, for deriving the lower bounds, we consider negligible overheads for changing the DVFS levels of the cores, as well as continuous values for the frequencies (and the corresponding voltages) between $(0, F_{\text{max}}]$. This approach results in a safe lower bound for the optimal energy and peak power consumptions.

8.2.1 Lower Bound for the Energy Consumption

As described in Chapter 2.2.2, for homogeneous systems with global DVFS, negligible leakage/independent power consumption, and negligible overheads for entering/leaving low-power modes, Yang *et al.* [109] have presented an optimal DVFS schedule for energy minimization when executing periodic frame-based real-time tasks. Their solution is based on an accelerating DVFS schedule and the *deep sleeping property* (which states that every core in the system is put to sleep after executing all the workload in its ready queue), as already shown in the example in Figure 2.4. In this solution, after the task partitioning is finished, the cores in the cluster are ordered increasingly according to their cycle utilizations, and every periodic frame is then divided into M fragments. During the i-th fragment, all active cores are executing at frequency f_i (the voltage is set to the minimum value such that f_i can be stably achieved) during time t_i. After time t_i elapses, the i-th core finishes all its workload for the current frame and it enters a low-power mode, such that there are $M - i + 1$ active cores in the i-th fragment. Moreover, every active core executes $\Delta c_i = D\left(w_i^* - w_{i-1}^*\right)$ core cycles during time t_i, such that $t_i = \frac{\Delta c_i}{f_i}$. Similar to Figure 8.4 and Figure 8.5, a brief example of a cluster with four cores using such an accelerating schedule as the DVFS scheduling policy is presented Figure 8.6.

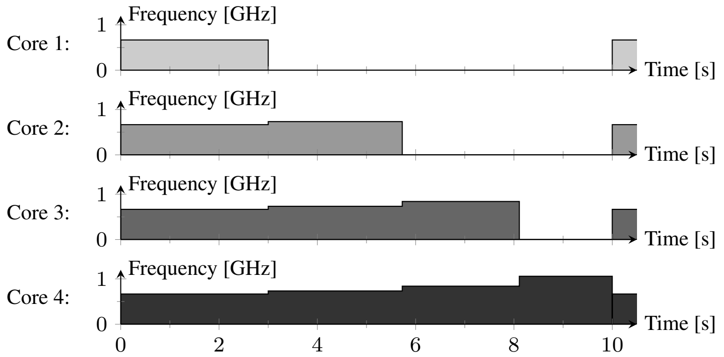


Figure 8.6: Example of an accelerating schedule satisfying the *deep sleeping property*. The hyper-period is 10 seconds, and the frequency demands of the cores are 0.2 GHz, 0.4 GHz, 0.6 GHz, and 0.8 GHz. Such a schedule can result in the optimal solution if the DVFS levels are chosen such that the total power consumption is constant and the core with the highest cycle utilization is always busy.

Considering our general power and energy models from Equation (3.3) and Equation (3.6) (which are more general than the model from [109] that considered negligible leakage and independent power consumptions), the energy consumed by the active cores running at frequency $f_i = \frac{\Delta c_i}{t_i}$ during time t_i in the i-th fragment can be computed as $(M - i + 1)\,(\alpha \cdot \frac{\Delta c_i{}^{\gamma}}{t_i{}^{\gamma}} + \beta \cdot \frac{\Delta c_i}{t_i} + \kappa)t_i$, such that the total energy consumption for all the cores in the cluster during a hyper-period is computed as shown in Equation (8.13).

$$E_{\downarrow}^{*} = \sum_{i=1}^{M} (M - i + 1) \left(\alpha \cdot \frac{\Delta c_i{}^{\gamma}}{t_i{}^{\gamma}} + \beta \cdot \frac{\Delta c_i}{t_i} + \kappa\right) t_i \tag{8.13}$$

In order to obtain the lower bound for the energy consumption, we apply the Kuhn-Tucker conditions [90] to Equation (8.13) under constraints $\sum_{i=1}^{M} t_i \leq D$ and $t_i \geq 0$ for $i = 1, 2, \ldots, M$, such that all time fragments are non-negative real numbers and their summation does not exceed the hyper-period. Particularly, the Lagrangian $\mathcal{L}$ is expressed as

$$\begin{aligned}
\mathcal{L} &= -\sum_{i=1}^{M}(M-i+1)\left(\alpha\frac{\Delta c_i{}^{\gamma}}{t_i{}^{\gamma}}+\beta\cdot\frac{\Delta c_i}{t_i}+\kappa\right)t_i+\lambda\left(D-\sum_{i=1}^{M}t_i\right)\\
&= -\sum_{i=1}^{M}\left[(M-i+1)\left(\alpha\frac{\Delta c_i{}^{\gamma}}{t_i{}^{\gamma}}+\beta\cdot\frac{\Delta c_i}{t_i}+\kappa\right)+\lambda\right]t_i+\lambda D\\
&= -\sum_{i=1}^{M}\left\{\left[(M-i+1)\,\kappa+\lambda\right]t_i+(M-i+1)\,\beta\cdot\Delta c_i+(M-i+1)\,\alpha\cdot\frac{\Delta c_i{}^{\gamma}}{t_i{}^{\gamma-1}}\right\}+\lambda D
\end{aligned}$$

where λ is the Lagrange multiplier, and $E^*_{\downarrow}(t_i)$ is introduced inside $\mathcal{L}$ with a negative multiplier so that we can use this for minimization instead of maximization. Given that all t_i are time intervals which are clearly non-negative real numbers, then further considerations about condition $t_i \geq 0$ are not necessary, such that the *effective* necessary Kuhn-Tucker conditions for a solution to be minimal are shown in Equation (8.14).

$$\begin{array}{lll}
\frac{\partial\mathcal{L}}{\partial t_i}=0 & \forall i=1,2,\ldots,M & \\
\sum_{i=1}^{M}t_i\leq D & \lambda\geq 0 & \lambda\left(D-\sum_{i=1}^{M}t_i\right)=0
\end{array}\tag{8.14}$$

In order to obtain the value of all t_i and f_i that minimize the energy consumption, the first-order derivative of the Lagrangian is set to zero, i.e.,

$$\frac{\partial\mathcal{L}}{\partial t_i}=(M-i+1)\,\kappa+\lambda-(\gamma-1)\,(M-i+1)\,\alpha\frac{\Delta c_i{}^{\gamma}}{t_i{}^{\gamma}}=0,$$

from which we can derive the set of t_i for all $i=1,2,\ldots,M$ that minimizes the energy consumption for the lower bound, as shown in Equation (8.15).

$$t_i=\sqrt[\gamma]{\frac{\alpha\,(\gamma-1)\,(M-i+1)}{(M-i+1)\,\kappa+\lambda}}\Delta c_i\tag{8.15}$$

Following, in order to meet Kuhn-Tucker condition $\lambda(D-\sum_{i=1}^{M}t_i)=0$ from Equation (8.14), there are two cases. First, in case $\sum_{i=1}^{M}t_i<D$, then λ has to be equal to 0, such that t_i in Equation (8.15) becomes $t_i=\frac{\Delta c_i}{f_{\text{crit}}}$ for all $i=1,2,\ldots,M$, and all the active cores run at frequency f_{crit} for the entire hyper-period, which is a feasible solution when $w^*_M \leq f_{\text{crit}}$. Here, the physical interpretation is that, if the summation of the duration of all i fragments is smaller than the hyper-period, then even the core with the highest workload is not always busy in the optimal solution, and it entered a low-power mode when it finished all its workload in a hyper-period. According to the behavior of the energy model (as shown in Figure 3.5), we know that the optimal frequency for execution when there is negligible overhead for entering a low-power mode is the critical frequency, and therefore, even if it is feasible to execute a core at a lower frequency, this should be avoided from an energy efficiency point of view. Therefore, the only case in the optimal solution in which the core with the highest workload would choose to enter a low-power mode, is in case that all the tasks assigned to it can feasible meet their deadlines when the core always executes at f_{crit}.

For the second case, in which $\lambda>0$, then to meet Kuhn-Tucker condition $\lambda(D-\sum_{i=1}^{M}t_i)=0$ it should hold that $\sum_{i=1}^{M}t_i=D$. Physically, this means that it is no longer feasible to meet the timing constraints of all tasks by running all cores at f_{crit}. Therefore, from Equation (8.15), the equality shown in Equation (8.16) should hold.

$$\sum_{i=1}^{M}t_i=\sum_{i=1}^{M}\sqrt[\gamma]{\frac{\alpha\,(\gamma-1)\,(M-i+1)}{(M-i+1)\,\kappa+\lambda}}\Delta c_i=D\tag{8.16}$$

Given that Equation (8.16) is strictly decreasing with respect to λ, and λ is the only unknown variable in this expression, one possibility to compute the value of λ is to apply the Newton-Raphson method [7]. However, the Newton-Raphson method only gives numerical results for a specific case study, but there is no explicit form to solve Equation (8.16) arithmetically. Therefore, in order to obtain an analytical expression for the lower bound of the energy consumption which can be used for the worst-case analysis for the general cases, we approximate $E_{\downarrow}^{*}$ by defining an auxiliary frequency, denoted as f_{dyn}, such that $f_{\text{crit}} < f_{\text{dyn}} < F_{\max}$. For such a purpose, when we have that $w_M^* \leq f_{\text{dyn}}$, then we approximate $E_{\downarrow}^{*}$ by considering that all cores run at f_{crit}, which although it would not be a feasible schedule, we know that it is the lowest value that the energy consumption could ever achieve. Furthermore, it is clear that when w_M^* is high, the energy consumption resulting from the dynamic power and leakage power play a more important role than the independent power consumption. Hence, when $w_M^* > f_{\text{dyn}}$, we approximate $E_{\downarrow}^{*}$ by considering that $\kappa = 0$, such that we ignore the effects of the independent power consumption, but Equation (8.16) can be solved arithmetically. The idea behind using auxiliary frequency f_{dyn} is to provide a tighter lower bound than simply choosing one approximation over the other. Particularly, from Equation (8.16) we have that

$$D = \sum_{j=1}^{M} \sqrt[\gamma]{\frac{\alpha(\gamma-1)(M-j+1)}{\lambda}} \Delta c_j \quad \Rightarrow \quad \frac{1}{\sqrt[\gamma]{\lambda}} = \frac{D}{\sqrt[\gamma]{\alpha(\gamma-1)} \sum_{j=1}^{M} \sqrt[\gamma]{M-j+1} \Delta c_j}$$

and therefore, also considering that $\kappa = 0$ in Equation (8.15), and by replacing this expression of $\frac{1}{\sqrt[\gamma]{\lambda}}$ inside Equation (8.15), we have that

$$t_i = \frac{\sqrt[\gamma]{\alpha(\gamma-1)(M-i+1)}}{\sqrt[\gamma]{\lambda}} \Delta c_i = \frac{D \sqrt[\gamma]{\alpha(\gamma-1)(M-i+1)} \Delta c_i}{\sqrt[\gamma]{\alpha(\gamma-1)} \sum_{j=1}^{M} \sqrt[\gamma]{M-j+1} \Delta c_j} = \frac{D \sqrt[\gamma]{M-i+1} \Delta c_i}{\sum_{j=1}^{M} \sqrt[\gamma]{M-j+1} \Delta c_j}$$

such that, since $\Delta c_i = D\left(w_i^* - w_{i-1}^*\right)$ and $f_i = \frac{\Delta c_i}{t_i}$, the solution of t_i and f_i for all $i = 1, 2, \ldots, M$ is computed as shown in Equation (8.17).

$$t_i = \frac{D\left(w_i^* - w_{i-1}^*\right) \sqrt[\gamma]{M-i+1}}{\sum_{j=1}^{M} \left(w_j^* - w_{j-1}^*\right) \sqrt[\gamma]{M-j+1}} \quad \text{and} \quad f_i = \frac{\sum_{j=1}^{M} \left(w_j^* - w_{j-1}^*\right) \sqrt[\gamma]{M-j+1}}{\sqrt[\gamma]{M-i+1}} \tag{8.17}$$

Therefore, assuming that in case that $w_M^* \leq f_{\text{dyn}}$ we approximate $E_{\downarrow}^{*}$ by considering that all cores run at f_{crit}, by knowing that $\sum_{i=1}^{M} \left(w_i^* - w_{i-1}^*\right)(M-i+1)$ is equal to $\sum_{i=1}^{M} w_i^*$ (proven by simple unrolling of the summation), and by replacing the results from Equation (8.17) inside Equation (8.13) in case that $w_M^* > f_{\text{dyn}}$, then the lower bound for the total energy consumption for all the cores in the cluster during a hyper-period is computed as shown in Equation (8.18).

$$E_{\downarrow}^{*} = \begin{cases} D\left(\alpha \cdot {f_{\text{crit}}}^{\gamma} + \beta \cdot f_{\text{crit}} + \kappa\right) \frac{\sum_{i=1}^{M} w_i^*}{f_{\text{crit}}} & \text{if } w_M^* \leq f_{\text{dyn}} \\ D \cdot \alpha \left[\sum_{i=1}^{M} \left(w_i^* - w_{i-1}^*\right) \sqrt[\gamma]{M-i+1}\right]^{\gamma} + D \cdot \beta \sum_{i=1}^{M} w_i^* & \text{otherwise} \end{cases} \tag{8.18}$$

Up to this point, we have assumed that the optimal task partition and the associated cycle utilizations $w_1^*, w_2^*, \cdots, w_M^*$ are known. However, since obtaining the optimal task partition is an $\mathcal{NP}$-hard problem [109], we further need to derive a lower bound for the optimal task partition. Thus, Lemma 1 presents a cycle utilization adjustment for the optimal task partition that results in a reduced energy consumption for the lower bound. An example of such a cycle utilization adjustment, when $w_M^* \geq w_M^{\text{DLTF}}$, is shown in Figure 8.7.

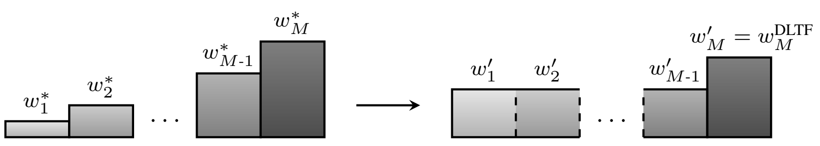


Figure 8.7: Example of cycle utilizations adjustment from $w_1^*, \ldots, w_M^*$ to $w_1', \ldots, w_M'$, when $w_M^* \geq w_M^{\text{DLTF}}$.

Lemma 1 *After applying DLTF for partitioning tasks, if we adjust $w_1^*, w_2^*, \ldots, w_M^*$ to $w_1', w_2', \ldots, w_M'$, then the lower bound for the energy consumption $E_{\downarrow}^{*}$ in Equation (8.18) by using $w_1^*, w_2^*, \ldots, w_M^*$ is further* *reduced by using $w'_1, w'_2, \ldots, w'_M$, in which $w'_1 = w'_2 = \cdots = w'_{M-1} = \frac{\sum_{i=1}^{M-1} w'_i}{M-1}$ and*

$$w'_M = \begin{cases} w_M^{DLTF} & \textit{if } w_M^* \geq w_M^{DLTF} \\ \max\left\{ \frac{w_M^{DLTF}}{\theta_{LTF}}, \frac{\sum_{i=1}^{M} w_i^*}{M} \right\} & \textit{otherwise.} \end{cases}$$

Proof. This lemma can be first proved by reorganizing the cycle utilizations $w_1^*, w_2^*, \ldots, w_M^*$ to cycle utilizations $w'_1, w'_2, \ldots, w'_M$, such that $w'_M = w_M^*$ and $w'_1 = w'_2 = \cdots = w'_{M-1} = \frac{\sum_{i=1}^{M-1} w_i^*}{M-1}$. Particularly, the value of the first part of Equation (8.18) (in which $w_M^* \leq f_{\text{dyn}}$) only depends on the *total cycle utilization*, which is constant for any task partition, i.e., $\sum_{i=1}^{M} w_i^* = \sum_{i=1}^{M} w'_i$. Hence, in this case the adjustment of the cycle utilizations has no impact in the value of $E_\downarrow^*$. For the second part of Equation (8.18) (in which $w_M^* > f_{\text{dyn}}$), we introduce the scaling factors r_i^* and r'_i, from which we can rephrase the cycle utilizations w_i^* and w'_i for all $i = 0, 1, \ldots, M$ as $w_i^* = r_i^* \cdot w_M^*$ and $w'_i = r'_i \cdot w'_M$, respectively, where

$$0 = r_0^* \leq r_1^* \leq r_2^* \leq \ldots \leq r_{M-1}^* \leq r_M^* = 1 \qquad \text{and} \qquad 0 = r'_0 \leq r'_1 \leq r'_2 \leq \ldots \leq r'_{M-1} \leq r'_M = 1.$$

Therefore, since $\gamma > 1$ and at this point we still have that $w'_M = w_M^*$, given a change in the cycle utilizations from $r_1^*, \ldots, r_M^*$ to $r'_1, \ldots, r'_M$, such that $r_M^* = r'_M = 1$ and $\sum_{i=1}^{M-1} r_i^* = \sum_{i=1}^{M-1} r'_i$, then the value of $E_\downarrow^*$ is reduced when $\sum_{i=1}^{M} \left(w_i^* - w_{i-1}^*\right) \sqrt[\gamma]{M-i+1} = w_M^* \left[1 - r_{M-1}^* + \sum_{i=1}^{M-1} \left(r_i^* - r_{i-1}^*\right) \sqrt[\gamma]{M-i+1}\right]$ is reduced. Therefore, in order to obtain the cycle utilization adjustment that minimizes such an expression, we formulate the following linear programming:

$$\text{Minimize: } 1 - r'_{M-1} + \sum_{i=1}^{M-1} \left(r'_i - r'_{i-1}\right) \sqrt[\gamma]{M-i+1}$$

$$\text{such that: } r'_{i-1} - r'_i \leq 0 \qquad \text{for all } i = 1, 2, \ldots, M-1$$

$$\sum_{i=1}^{M-1} r'_i = \sum_{i=1}^{M-1} r_i^*$$

$$0 \leq r_i^* \leq 1 \qquad \text{for all } i = 1, 2, \ldots, M-1.$$

We only sketch the proof for the optimization in the above linear programming. By the *extreme point theorem*, if there is a bounded and feasible solution, an optimal solution can be found in one of the extreme points in the constructed polyhedron. In this linear programming, the only extreme point happens when the constraints $\frac{\sum_{j=1}^{M-1} r_j^*}{M-1} = r'_{i-1} = r'_i$ for all $i = 2, 3, \ldots, M-1$ are tight. *In other words,* when $w'_M = w_M^*$ is fixed, *the lower bound of the energy consumption is minimized when the cycle utilizations are balanced for the first $M-1$ task sets.*

Then, further, we can easily reduce w'_M by a constant ϵ while increasing the other cycle utilizations $w'_1, w'_2, \ldots, w'_{M-1}$ by a constant $\frac{\epsilon}{M-1}$. Therefore, for the first case in the statement of the lemma, in which $w_M^* \geq w_M^{\text{DLTF}}$, we reduce w'_M (while increasing $w'_1, w'_2, \ldots, w'_{M-1}$) until w'_M reaches w_M^{DLTF}, such that this cycle utilizations adjustment from $w_1^*, w_2^*, \ldots, w_M^*$ to $w'_1, w'_2, \ldots, w'_M$ reduces the energy consumption lower bound $E_\downarrow^*$ in Equation (8.18). An example of this cycle utilizations adjustment can be seen in Figure 8.7.

For the second case in the statement of the lemma, in which $w_M^* < w_M^{\text{DLTF}}$, we know that $w_M^* \geq \frac{w_M^{\text{DLTF}}}{\theta_{\text{LTF}}}$ due to Equation (8.8). Therefore, we can greedily reduce the cycle utilization w'_M (while increasing $w'_1, w'_2, \ldots, w'_{M-1}$), until (a) w'_M reaches $\frac{w_M^{\text{DLTF}}}{\theta_{\text{LTF}}}$, point in which we cannot further reduce w'_M without violating $w_M^* \geq \frac{w_M^{\text{DLTF}}}{\theta_{\text{LTF}}}$; or either until (b) w'_M reaches the average cycle utilization, for which it holds that $w'_1 = w'_2 = \cdots = w'_M$ and we cannot further reduce w'_M without having that $w'_M < w'_{M-1}$.

According to the above cycle utilizations adjustment, both cases reduce the lower bound of the energy consumption $E_\downarrow^*$ in Equation (8.18), while ensuring that $w'_1 \leq \cdots \leq w'_M$. Thus, the lemma is proven. □

Finally, considering the cycle utilization adjustment presented in Lemma 1, the lower bound for the energy consumption from Equation (8.18) can be further reduced, such that $E_\downarrow^*$ is computed as shown in Equation (8.19), for which Figure 8.8 presents an example by considering power parameters $\gamma = 3$, $\alpha = 0.27 \frac{\text{W}}{\text{GHz}^3}$,

$\beta = 0.52 \frac{\text{W}}{\text{GHz}}$, and $\kappa = 0.5\,\text{W}$, modeled for a 22 nm OOO Alpha 21264 core (as detailed in Chapter 3.3).

$$E^*_{\downarrow} = \begin{cases} D\left(\alpha \cdot f_{\text{crit}}{}^{\gamma} + \beta \cdot f_{\text{crit}} + \kappa\right) \frac{\sum_{i=1}^{M} w'_i}{f_{\text{crit}}} & \text{if } w^*_M \leq f_{\text{dyn}} \\ D \cdot \alpha \cdot w'_M{}^{\gamma} \left[1 + \frac{w'_1}{w'_M}\left(\sqrt[\gamma]{M} - 1\right)\right]^{\gamma} + D \cdot \beta \sum_{i=1}^{M} w'_i & \text{otherwise} \end{cases} \tag{8.19}$$

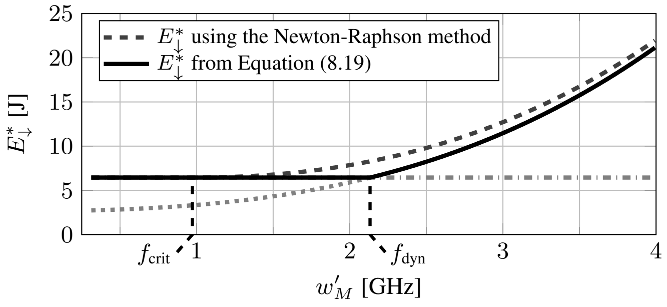


Figure 8.8: Example of $E^*_{\downarrow}$ as a function of w'_M, by using the Newton-Raphson method to solve Equation (8.16), and of the approximated lower bound from Equation (8.19), with $\gamma = 3$, $\alpha = 0.27 \frac{\text{W}}{\text{GHz}^3}$, $\beta = 0.52 \frac{\text{W}}{\text{GHz}}$, $\kappa = 0.5\,\text{W}$, $M = 16$, $D = 1\,\text{s}$, constant $\sum_{i=1}^{M} w'_i = 5 \cdot 10^9 \frac{\text{cycles}}{\text{second}}$, and $w'_1 = w'_2 = \cdots = w'_{M-1} = \frac{\sum_{i=1}^{M} w'_i - w'_M}{M-1}$.

There is an important conclusion to derive from Figure 8.8. Namely, given that $E^*_{\downarrow}$ is in the denominator of Equations (8.1) and (8.3), in order to derive an approximation factor without unnecessary pessimism, the value of f_{dyn} in Equation (8.19) should be chosen such that $E^*_{\downarrow}$ becomes a continuous function (as later formally proven in Lemma 3 and illustrated in Figure 8.10). Hence, we present Lemma 2, in which we choose the value of f_{dyn} for this condition to hold. For notational brevity, we define auxiliary function $U(\delta)$ as shown in Equation (8.20), with δ and $\delta^{\max}$ defined in Equation (8.21). Figure 8.9 illustrates $U(\delta)$ and $U(\delta^{\max})$ for $\gamma = 3$.

$$U(\delta) = \frac{1 - \delta + \delta \cdot M}{\left(1 - \delta + \delta \cdot \sqrt[\gamma]{M}\right)^{\gamma}} \leq U\left(\delta^{\max}\right) \tag{8.20}$$

$$\delta = \frac{\sum_{i=1}^{M-1} w'_i}{w'_M (M-1)} \quad \text{and} \quad \delta^{\max} = \frac{\gamma - 1 + M - \gamma \cdot \sqrt[\gamma]{M}}{(\gamma - 1)\left(M \cdot \sqrt[\gamma]{M} - M - \sqrt[\gamma]{M} + 1\right)} \tag{8.21}$$

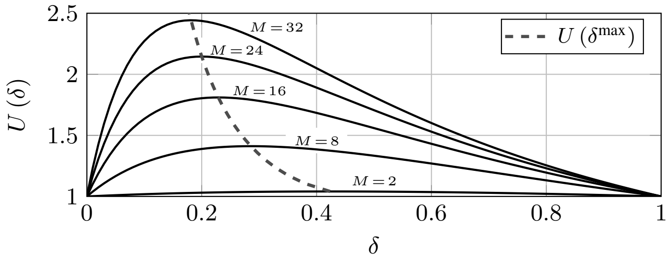


Figure 8.9: Example of function $U(\delta)$ with $\gamma = 3$, for different values of M, where we also highlight $U(\delta^{\max})$.

Lemma 2 *The lower bound of the energy consumption $E^*_{\downarrow}$ in Equation* (8.19) *is a continuous function when the value of f_{dyn} is equal to*

$$f_{dyn} = f_{crit} \left[\gamma \cdot U(\delta)\right]^{\frac{1}{\gamma - 1}},$$

and the maximum value of f_{dyn}, defined as f^{max}_{dyn}, occurs at

$$f^{max}_{dyn} = f_{crit} \left[\gamma \cdot U\left(\delta^{max}\right)\right]^{\frac{1}{\gamma - 1}},$$

with $U(\delta)$, δ, and δ^{max} defined in Equation (8.20) *and Equation* (8.21).

Proof. For $E^*_{\downarrow}$ in Equation (8.19) to be continuous, we match both parts of its expression and find the value of w'_M for which the equality holds. That is,

$$\alpha \cdot w'_M{}^{\gamma} \left[1 + \frac{w'_1}{w'_M}\left(\sqrt[\gamma]{M} - 1\right)\right]^{\gamma} + \beta \sum_{i=1}^{M} w'_i = \left(\alpha \cdot f_{\text{crit}}{}^{\gamma} + \beta \cdot f_{\text{crit}} + \kappa\right) \frac{\sum_{i=1}^{M} w'_i}{f_{\text{crit}}}.$$

Furthermore, since from Equation (3.7) the value of f_{crit} is computed as $\sqrt[\gamma]{\frac{\kappa}{(\gamma-1)\alpha}}$, it holds that

$$\frac{\alpha \cdot {f_{\text{crit}}}^{\gamma} + \beta \cdot f_{\text{crit}} + \kappa}{f_{\text{crit}}} = \frac{\frac{\kappa}{\gamma-1} + \kappa}{f_{\text{crit}}} + \beta = \frac{\kappa \cdot \gamma}{(\gamma-1)\, f_{\text{crit}}} + \beta = \gamma \cdot \alpha \cdot {f_{\text{crit}}}^{\gamma-1} + \beta,$$

from which the above equality becomes

$$\alpha \cdot {w'_M}^{\gamma} \left[1 - \delta\left(\sqrt[\gamma]{M} - 1\right)\right]^{\gamma} + \beta \sum_{i=1}^{M} w'_i = \left(\alpha \cdot \gamma \cdot {f_{\text{crit}}}^{\gamma-1} + \beta\right) \sum_{i=1}^{M} w'_i.$$

Then, since from Lemma 1 and Equation (8.21) it holds that $\sum_{i=1}^{M} w'_i = w'_M \left[1 + \delta\,(M-1)\right]$, we have that

$${w'_M}^{\gamma-1} = \frac{\gamma \cdot {f_{\text{crit}}}^{\gamma-1}\,(1 - \delta + \delta \cdot M)}{\left[1 - \delta\left(\sqrt[\gamma]{M} - 1\right)\right]^{\gamma}} = {f_{\text{crit}}}^{\gamma-1} \cdot \gamma \cdot U(\delta),$$

where this w'_M is the value of f_{dyn} for $E^*_{\downarrow}$ in Equation (8.19) to be continuous.

Finally, since $U(\delta)$ is a convex function with respect to δ when $\gamma > 1$, by taking the first-order derivative of $U(\delta)$ with respect to δ, its maximum value happens when δ is $\delta^{\max}$. Therefore, the lemma is proven. □

8.2.2 Lower Bound for the Peak Power Consumption

Given that energy is the integration of power through time, when it holds that $w_M^{\text{DLTF}} \geq f_{\text{crit}}$, minimizing the energy consumption while satisfying the timing constraints of all tasks is equivalent to minimizing the average power consumption while also satisfying the timing constraints. This implies that the lower bound for the peak power consumption, denoted as $\hat{P}^*_{\downarrow}$, is found when the power consumption is constant during the entire hyper-period, and equivalent to the minimum average power consumption. Namely, when $w_M^{\text{DLTF}} \geq f_{\text{crit}}$, we have that $E^*_{\downarrow} = \hat{P}^*_{\downarrow} \cdot D$.

However, the critical frequency f_{crit} is not involved when talking about power consumption, and hence executing at low DVFS levels (as long as the timing constraints of the tasks allow it) will always result in small power consumption values. Therefore, we only consider the part of $E^*_{\downarrow}$ in Equation (8.18) that does not involve f_{crit}. Furthermore, given that *at least one core* will be active at some point in time (independent of the task partition), there will be at least a power consumption of κ also present. Finally, the value of $\hat{P}^*_{\downarrow}$ can be computed by dividing the second part of Equation (8.18) by D, resulting in Equation (8.22).

$$\hat{P}^*_{\downarrow} = \alpha \cdot {w'_M}^{\gamma} \left[1 + \frac{w'_1}{w'_M}\left(\sqrt[\gamma]{M} - 1\right)\right]^{\gamma} + \beta \sum_{i=1}^{M} w'_i + \kappa \tag{8.22}$$

8.3 Approximation Factor Analysis: DLTF-SFA

8.3.1 Energy Minimization Analysis for DLTF-SFA

This section presents the approximation factor analysis of DLTF-SFA in terms of energy consumption, i.e., the worst-case behavior of DLTF-SFA for energy minimization, against the optimal solution (i.e., the optimal task partition, the optimal DVFS schedule, and the optimal DPM schedule). For simplicity of presentation, we first focus on deriving the approximation factor of DLTF-SFA for energy minimization when assuming *negligible overhead for sleeping*, defined as $\text{AF}_{\text{DLTF-SFA}}^{\text{energy}\ \cancel{\text{overheads}}}$. Then, for systems with *non-negligible overhead for sleeping* (i.e., for the practical scenario in real systems), we show how to extend our analysis to derive the approximation factor of combining DLTF-SFA with a DPM technique, specifically, the Left to Right (LTR) algorithm from [37]. To start, after deriving the lower bound of the energy consumption (i.e., $E^*_{\downarrow}$) in Equation (8.19) and the energy consumption of DLTF-SFA (i.e., $E_{\text{SFA}}^{\text{DLTF}}$) in Equation (8.10), we can replace the different possible values of $E^*_{\downarrow}$ and $E_{\text{SFA}}^{\text{DLTF}}$ from Equation (8.19) and Equation (8.10) inside Equation (8.1). Note that expression $E_{\text{SFA}}^{\text{DLTF}}$ in Equation (8.10) provides the energy consumption of DLTF-SFA by implicitly assuming *negligible* *overhead for sleeping*, and therefore, this replacement will result in an expression for $\text{AF}^{\text{energy }\sout{\text{overheads}}}_{\text{DLTF-SFA}}$. Furthermore, since as shown in Section 8.2.1 it holds that $\frac{\alpha \cdot f_{\text{crit}}{}^{\gamma} + \beta \cdot f_{\text{crit}} + \kappa}{f_{\text{crit}}} = \gamma \cdot \alpha \cdot f_{\text{crit}}{}^{\gamma-1} + \beta$, then we can express $\text{AF}^{\text{energy }\sout{\text{overheads}}}_{\text{DLTF-SFA}}$ as a function of w'_M and w_M^{DLTF} for a given f_{dyn}, as shown in Equation (8.23).

$$\text{AF}^{\text{energy }\sout{\text{overheads}}}_{\text{DLTF-SFA}}\left(w'_M\right) \leq \max \left\{ \begin{cases} 1 & \text{if } w_M^{\text{DLTF}} \leq f_{\text{crit}} \\ \frac{\alpha \cdot w_M^{\text{DLTF}\,\gamma} + \beta \cdot w_M^{\text{DLTF}} + \kappa}{w_M^{\text{DLTF}}\left(\gamma \cdot \alpha \cdot f_{\text{crit}}{}^{\gamma-1} + \beta\right)} & \begin{array}{l}\text{if } w_M^{\text{DLTF}} > f_{\text{crit}} \\ \text{and } f_{\text{crit}} < w'_M \leq f_{\text{dyn}}\end{array} \\ \frac{\left(\alpha \cdot w_M^{\text{DLTF}\,\gamma} + \beta \cdot w_M^{\text{DLTF}} + \kappa\right)\sum_{i=1}^{M} w_i^{\text{DLTF}}}{\alpha \cdot w_M^{\text{DLTF}} \cdot w'_M{}^{\gamma}\left[1 + \frac{w'_1}{w'_M}\left(\sqrt[\gamma]{M} - 1\right)\right]^{\gamma} + \beta \cdot w_M^{\text{DLTF}} \sum_{i=1}^{M} w'_i} & \text{otherwise} \end{cases} \right\}$$

(8.23)

As shown in Lemma 2, in order to derive an approximation factor without unnecessary pessimism, the value of f_{dyn} can be chosen such that $E^*_{\downarrow}$ is a continuous function. Given that $E_{\text{SFA}}^{\text{DLTF}}$ in Equation (8.10) is also a continuous function, setting $f_{\text{dyn}} = f_{\text{crit}}\left[\gamma \cdot U\left(\delta\right)\right]^{\frac{1}{\gamma-1}}$ (from Lemma 2) will ensure that $\text{AF}^{\text{energy }\sout{\text{overheads}}}_{\text{DLTF-SFA}}\left(w'_M\right)$ in Equation (8.23) is also a continuous function. Therefore, we now need to find the value of w'_M that maximizes $\text{AF}^{\text{energy }\sout{\text{overheads}}}_{\text{DLTF-SFA}}\left(w'_M\right)$ in Equation (8.23), which occurs when w'_M is equal to f_{dyn}, as proven in Lemma 3 and shown in the example in Figure 8.10 (where we also show poor choices of f_{dyn} that result in discontinuous functions and more pessimistic approximation factors). Furthermore, given that the maximum value of f_{dyn} occurs when $\delta = \delta^{\max}$, then it holds that the maximum value of $\text{AF}^{\text{energy }\sout{\text{overheads}}}_{\text{DLTF-SFA}}\left(w'_M\right)$ happens when $w'_M = f_{\text{dyn}}^{\max}$, as also proven in Lemma 3.

Lemma 3 *The maximum value of* $\mathit{AF}^{\mathit{energy}\ \sout{\mathit{overheads}}}_{\mathit{DLTF\text{-}SFA}}\left(w'_M\right)$ *in Equation* (8.23) *happens when* $w'_M = f_{\mathit{dyn}}$. *Furthermore, given that the maximum value of* f_{dyn} *occurs when* $\delta = \delta^{\mathit{max}}$, *it holds that*

$$\mathit{AF}^{\mathit{energy}\ \sout{\mathit{overheads}}}_{\mathit{DLTF\text{-}SFA}}\left(w'_M\right) \leq \mathit{AF}^{\mathit{energy}\ \sout{\mathit{overheads}}}_{\mathit{DLTF\text{-}SFA}}\left(f_{\mathit{dyn}}\right) \leq \mathit{AF}^{\mathit{energy}\ \sout{\mathit{overheads}}}_{\mathit{DLTF\text{-}SFA}}\left(f_{\mathit{dyn}}^{\mathit{max}}\right).$$

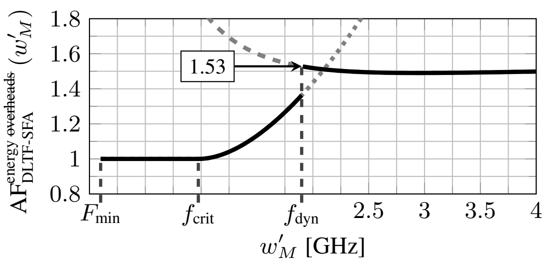
Proof. Given that α, β, κ, γ, and f_{crit} are all constants, we can easily see that $\text{AF}^{\text{energy }\sout{\text{overheads}}}_{\text{DLTF-SFA}}\left(w'_M\right)$ is a convex and increasing function with respect to w_M^{DLTF} in case that $w_M^{\text{DLTF}} > f_{\text{crit}}$ and $f_{\text{crit}} < w'_M \leq f_{\text{dyn}}$, as seen in the example in Figure 8.10. Furthermore, although at this point the precise relationship between w'_M and w_M^{DLTF} is not relevant, from Lemma 1 we know that the relationship will be somehow bounded, such that for this case $\text{AF}^{\text{energy }\sout{\text{overheads}}}_{\text{DLTF-SFA}}\left(w'_M\right)$ is a convex and increasing function also with respect to w'_M. Therefore, for this case, the value of $\text{AF}^{\text{energy }\sout{\text{overheads}}}_{\text{DLTF-SFA}}\left(w'_M\right)$ will be maximized at the border condition in which w'_M cannot grow anymore, i.e., when $w'_M = f_{\text{dyn}}$, such that it holds that $\text{AF}^{\text{energy }\sout{\text{overheads}}}_{\text{DLTF-SFA}}\left(w'_M\right) \leq \text{AF}^{\text{energy }\sout{\text{overheads}}}_{\text{DLTF-SFA}}\left(f_{\text{dyn}}\right)$. Moreover, since the maximum value of f_{dyn} occurs when $\delta = \delta^{\max}$, then the statement in the lemma holds.

For the other case, in which $w_M^{\text{DLTF}} > f_{\text{crit}}$ and $f_{\text{dyn}} < w'_M$, we have shown in Lemma 1 that the lower bound of the energy consumption is reduced, and therefore the approximation factor $\text{AF}^{\text{energy }\sout{\text{overheads}}}_{\text{DLTF-SFA}}\left(w'_M\right)$ is maximized, when $w'_1 = \cdots = w'_{M-1} = \frac{\sum_{i=1}^{M-1} w'_i}{M-1}$. From the definition of δ in Equation (8.21), this also means that $\delta = \frac{w'_1}{w'_M}$ and $\sum_{i=1}^{M} w'_i = w'_M\left(1 - \delta + \delta \cdot M\right)$. Hence, we can rephrase $\text{AF}^{\text{energy }\sout{\text{overheads}}}_{\text{DLTF-SFA}}\left(w'_M\right)$ in Equation (8.23) when $w_M^{\text{DLTF}} > f_{\text{crit}}$ and $f_{\text{dyn}} < w'_M$ as

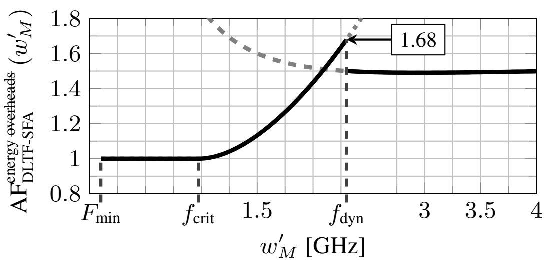
$$\frac{\left(\alpha \cdot w_M^{\text{DLTF}\,\gamma} + \beta \cdot w_M^{\text{DLTF}} + \kappa\right)\sum_{i=1}^{M} w_i^{\text{DLTF}}}{\alpha \cdot w_M^{\text{DLTF}} \cdot w'_M{}^{\gamma}\left[1 + \frac{w'_1}{w'_M}\left(\sqrt[\gamma]{M} - 1\right)\right]^{\gamma} + \beta \cdot w_M^{\text{DLTF}} \sum_{i=1}^{M} w'_i} =$$

$$= \frac{\left(\alpha \cdot w_M^{\text{DLTF}\,\gamma-1} + \beta + \frac{\kappa}{w_M^{\text{DLTF}}}\right)\left(1 - \delta + \delta \cdot M\right)}{\alpha \cdot w'_M{}^{\gamma-1}\left(1 - \delta + \delta \cdot \sqrt[\gamma]{M}\right)^{\gamma} + \beta\left(1 - \delta + \delta \cdot M\right)} = \frac{\alpha \cdot w_M^{\text{DLTF}\,\gamma-1} + \beta + \frac{\kappa}{w_M^{\text{DLTF}}}}{\alpha \cdot w'_M{}^{\gamma-1} + \beta \cdot U\left(\delta\right)} \cdot U\left(\delta\right)$$

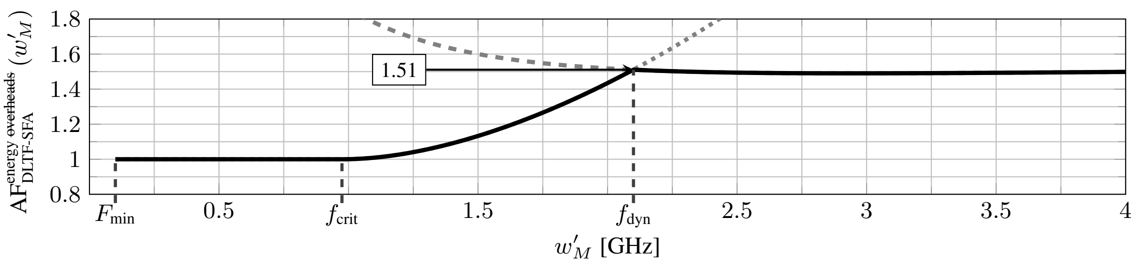
From this last expression, for both possible relations of w'_M and w_M^{DLTF} from Lemma 1, given that α, β, γ, and M, are constants, and given that $U\left(\delta\right)$ is also constant for a given *total cycle utilization*, then it holds that $\text{AF}^{\text{energy }\sout{\text{overheads}}}_{\text{DLTF-SFA}}\left(w'_M\right)$ in Equation (8.23) is a decreasing function with respect to w'_M for the case in which $w_M^{\text{DLTF}} > f_{\text{crit}}$ and $f_{\text{dyn}} < w'_M$, as seen in the example if Figure 8.10. Therefore, for this case, the value of $\text{AF}^{\text{energy}\ \cancel{\text{overheads}}}_{\text{DLTF-SFA}}(w'_M)$ will be maximized at the border condition in which w'_M cannot decrease anymore, i.e., when $w'_M = f_{\text{dyn}}$, such that it holds that $\text{AF}^{\text{energy}\ \cancel{\text{overheads}}}_{\text{DLTF-SFA}}(w'_M) \leq \text{AF}^{\text{energy}\ \cancel{\text{overheads}}}_{\text{DLTF-SFA}}(f_{\text{dyn}}) \leq \text{AF}^{\text{energy}\ \cancel{\text{overheads}}}_{\text{DLTF-SFA}}(f^{\max}_{\text{dyn}})$. Thus, the lemma is proven. □



(a) f_{dyn} that would result in a pessimistic approximation factor



(b) f_{dyn} that would result in a pessimistic approximation factor



(c) Lowest approximation factor by properly choosing f_{dyn} such that $E^*_{\downarrow}$ is a continuous function

Figure 8.10: Example of $\text{AF}^{\text{energy}\ \cancel{\text{overheads}}}_{\text{DLTF-SFA}}(w'_M)$ when $\gamma = 3$, $\alpha = 0.27\frac{\text{W}}{\text{GHz}^3}$, $\beta = 0.52\frac{\text{W}}{\text{GHz}}$, and $\kappa = 0.5\,\text{W}$ (i.e., for 22 nm OOO Alpha 21264 cores, as detailed in Chapter 3.3), for $M = 16$, with $w'_i = 0.51 \cdot w'_M$ for all $i = 1, \ldots, M-1$, and $w_M^{\text{DLTF}} = w'_M$, considering different choices for f_{dyn}.

At this point, we have to consider the specific possible relations between w'_M and w_M^{DLTF} from Lemma 1. For such a purpose, for expression $\text{AF}^{\text{energy}\ \cancel{\text{overheads}}}_{\text{DLTF-SFA}}(w'_M)$ in Equation (8.23), we proceed to replace w_M^{DLTF} by w'_M according to Lemma 1. For the first case, i.e., when $w^*_M \geq w_M^{\text{DLTF}}$, it holds that $w_M^{\text{DLTF}} = w'_M$, such that

$$\frac{\alpha \cdot {w_M^{\text{DLTF}}}^{\gamma} + \beta \cdot w_M^{\text{DLTF}} + \kappa}{w_M^{\text{DLTF}}\left(\gamma \cdot \alpha \cdot {f_{\text{crit}}}^{\gamma-1} + \beta\right)} = \frac{\alpha \cdot {w'_M}^{\gamma-1} + \beta + \frac{\kappa}{w'_M}}{\gamma \cdot \alpha \cdot {f_{\text{crit}}}^{\gamma-1} + \beta}.$$

Thus, given that according to Lemma 3 the value $\text{AF}^{\text{energy}\ \cancel{\text{overheads}}}_{\text{DLTF-SFA}}(w'_M)$ is maximized at $f^{\max}_{\text{dyn}}$, then from the definition of $f^{\max}_{\text{dyn}}$ in Lemma 2, the maximum value of $\text{AF}^{\text{energy}\ \cancel{\text{overheads}}}_{\text{DLTF-SFA}}(w'_M)$ when $w^*_M \geq w_M^{\text{DLTF}}$ can be computed as shown in Equation (8.24).

$$\text{AF}^{\text{energy}\ \cancel{\text{overheads}}}_{\text{DLTF-SFA}}(w'_M) \leq \text{AF}^{\text{energy}\ \cancel{\text{overheads}}}_{\text{DLTF-SFA}}\left(f^{\max}_{\text{dyn}}\right) = \frac{\alpha \cdot {f_{\text{crit}}}^{\gamma-1} \cdot \gamma \cdot U\left(\delta^{\max}\right) + \beta + \frac{\kappa}{f_{\text{crit}}\left[\gamma \cdot U\left(\delta^{\max}\right)\right]^{\frac{1}{\gamma-1}}}}{\gamma \cdot \alpha \cdot {f_{\text{crit}}}^{\gamma-1} + \beta} \tag{8.24}$$

For the second case of the relationship between w'_M and w_M^{DLTF} from Lemma 1, i.e., when $w^*_M < w_M^{\text{DLTF}}$, it holds that $w'_M = \max\left\{\frac{w_M^{\text{DLTF}}}{\theta_{\text{LTF}}}, \frac{\sum_{i=1}^{M} w^*_i}{M}\right\}$. Moreover, even though Lemma 2 states that $U(\delta) \leq U(\delta^{\max})$, for this case, the relation $\frac{w'_1}{w'_M}$ is in fact constrained, such that δ never reaches the value of $\delta^{\max}$, as proven in Lemmas 4 and 5. Particularly, when $w^*_M < w_M^{\text{DLTF}}$, Lemma 4 proves that $U(\delta) \geq \frac{4M+1}{6M}$, and Lemma 5 proves that $U(\delta)$ is a strictly decreasing function with respect to $U(\delta)$ when $U(\delta) \geq 0.5$. Therefore, since from Lemma 4 it holds that $\delta \geq \frac{4M+1}{6M} \geq \frac{2}{3} \geq 0.5$, from Lemma 5 we know that the maximum value of $U(\delta)$ when $w^*_M < w_M^{\text{DLTF}}$ will be $U\left(\frac{4M+1}{6M}\right)$, as seen in Figure 8.11. In this way, f_{dyn} will also reach its maximum value when $\delta = \frac{4M+1}{6M}$, resulting in a less pessimistic approximation factor than simply considering $f^{\max}_{\text{dyn}}$.

Lemma 4 *When applying DLTF for task partitioning, in case that $w_M^* < w_M^{DLTF}$, it holds that*

$$\frac{w_1'}{w_M'} \geq \frac{4M+1}{6M},$$

with $w_1', w_2', \ldots, w_M'$ defined in Lemma 1.

Proof. In case that $w_M^* < w_M^{\text{DLTF}}$, by using $w_1', w_2', \ldots, w_M'$ from Lemma 1, there are two cases to consider: (a) $w_M' = \frac{\sum_{i=1}^{M} w_i^*}{M}$, or (b) $w_M' = \frac{w_M^{\text{LTF}}}{\theta_{\text{LTF}}}$. For case (a), it is clear that $\frac{w_1'}{w_M'} = 1$. Therefore, we focus on case (b), for which starting from Equation (8.7) and remembering that $w_M^{\text{DLTF}} = w_M^{\text{LTF}}$, we have that

$$w_M^{\text{LTF}} + \sum_{i=1}^{M-1} w_i^{\text{LTF}} = w_M' + \sum_{i=1}^{M-1} w_i' = \frac{w_M^{\text{LTF}}}{\theta_{\text{LTF}}} + (M-1)\, w_1' \quad \Rightarrow \quad w_1' = \frac{w_M^{\text{LTF}}\,(\theta_{\text{LTF}} - 1)}{\theta_{\text{LTF}}\,(M-1)} + \frac{\sum_{i=1}^{M-1} w_i^{\text{LTF}}}{(M-1)}$$

$$\Rightarrow \quad \frac{w_1'}{w_M'} = \frac{\theta_{\text{LTF}} - 1}{M-1} + \frac{\theta_{\text{LTF}}}{w_M^{\text{LTF}}} \frac{\sum_{i=1}^{M-1} w_i^{\text{LTF}}}{(M-1)}.$$

Finally, considering Equation (8.6), it holds that

$$\frac{w_1'}{w_M'} \geq \frac{\theta_{\text{LTF}} - 1}{M-1} + \frac{\theta_{\text{LTF}}}{2} = \frac{\frac{4}{3} - \frac{1}{3M} - 1}{M-1} + \frac{\frac{4}{3} - \frac{1}{3M}}{2} = \frac{4M+1}{6M},$$

and thus the lemma is proven. □

Lemma 5 *Function $U(\delta)$, defined in Equation* (8.20) *as $U(\delta) = \frac{1-\delta+\delta\cdot M}{\left(1-\delta+\delta\cdot \sqrt[\gamma]{M}\right)^{\gamma}}$ with δ defined in Equation* (8.21), *is a strictly decreasing function with respect to δ for $M \geq 2$, $\gamma > 1$ and $U(\delta) \geq 0.5$.*

Proof. From Equation (8.20), illustrated in Figure 8.9, it is clear that $U(\delta)$ is a decreasing function with respect to δ when $\delta^{\max} \leq \delta \leq 1$, as long as $M \geq 2$, and $\gamma > 1$. By taking the first-order derivative of $\delta^{\max}$ with respect to M from Equation (8.21), we easily prove that $\delta^{\max}$ is a decreasing function with respect to M, since $\frac{\partial \delta^{\max}}{\partial M} \leq 0$ for all $M \geq 2$ and $\gamma > 1$. Thus, for a given γ, the highest value of $\delta^{\max}$ happens when $M = 2$ (as seen in Figure 8.9), which we define as $\delta_{M=2}^{\max} = \frac{\gamma - \gamma\sqrt[\gamma]{2} + 1}{(\gamma-1)\left(\sqrt[\gamma]{2}-1\right)}$.

Moreover, by taking the first-order derivative of $\delta^{\max}$ with respect to γ from Equation (8.21), we can also prove that $\delta^{\max}$ is an increasing function with respect to γ, since $\frac{\partial \delta^{\max}}{\partial \gamma} \geq 0$ for all $M \geq 2$ and $\gamma > 1$. Therefore, the highest value of $\delta^{\max}$ is obtained when $M \geq 2$ (i.e., for $\delta_{M=2}^{\max}$), and when $\gamma \to \infty$ (i.e., at $\lim_{\gamma\to\infty} \delta_{M=2}^{\max}$), which converges to $\frac{1}{\ln 2} - 1 = 0.443$.

Finally, since $\delta^{\max} < 0.5$ for any $M \geq 2$ and $\gamma > 1$, then $U(\delta)$ is a decreasing function when $\delta \geq 0.5$ for any $M \geq 2$ and $\gamma > 1$. Thus, the lemma is proven. □

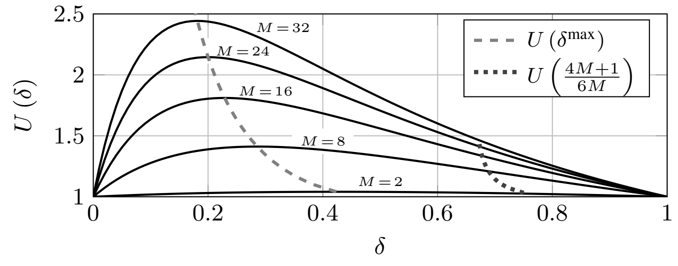


Figure 8.11: Example of function $U(\delta)$ with $\gamma = 3$, for different values of M, where we also highlight $U(\delta^{\max})$ and $U\left(\frac{4M+1}{6M}\right)$. From the figure we can see that if $\delta \geq \frac{4M+1}{6M}$, then $U(\delta)$ is maximized at $U\left(\frac{4M+1}{6M}\right)$.

In this way, following a similar procedure as above, by replacing w_M^{DLTF} with w_M' according to Lemma 1 when $w_M^* < w_M^{\text{DLTF}}$, it holds that $w_M^{\text{DLTF}} = \theta_{\text{LTF}} \cdot w_M'$, such that

$$\frac{\alpha \cdot {w_M^{\text{DLTF}}}^{\gamma} + \beta \cdot w_M^{\text{DLTF}} + \kappa}{w_M^{\text{DLTF}}\left(\gamma \cdot \alpha \cdot {f_{\text{crit}}}^{\gamma-1} + \beta\right)} = \frac{\alpha \cdot {\theta_{\text{LTF}}}^{\gamma-1} \cdot {w_M'}^{\gamma-1} + \beta + \frac{\kappa}{\theta_{\text{LTF}} \cdot w_M'}}{\gamma \cdot \alpha \cdot {f_{\text{crit}}}^{\gamma-1} + \beta}.$$

Therefore, given that according to Lemma 3 it holds that $\mathrm{AF}^{\text{energy}~\cancel{\text{overheads}}}_{\text{DLTF-SFA}}(w'_M) \leq \mathrm{AF}^{\text{energy}~\cancel{\text{overheads}}}_{\text{DLTF-SFA}}(f_{\text{dyn}})$, and given that according to Lemma 4 and Lemma 5 the maximum value of f_{dyn} for this case is $f_{\text{crit}}\left[\gamma \cdot U\left(\frac{4M+1}{6M}\right)\right]^{\frac{1}{\gamma-1}}$, then the maximum value of $\mathrm{AF}^{\text{energy}~\cancel{\text{overheads}}}_{\text{DLTF-SFA}}(w'_M)$ when $w^*_M < w^{\text{DLTF}}_M$ can be computed as shown in Equation (8.25).

$$\mathrm{AF}^{\text{energy}~\cancel{\text{overheads}}}_{\text{DLTF-SFA}}(w'_M) \leq \frac{\alpha \cdot \theta_{\text{LTF}}{}^{\gamma-1} \cdot f_{\text{crit}}{}^{\gamma-1} \cdot \gamma \cdot U\left(\frac{4M+1}{6M}\right) + \beta + \frac{\kappa}{\theta_{\text{LTF}} \cdot f_{\text{crit}}\left[\gamma \cdot U\left(\frac{4M+1}{6M}\right)\right]^{\frac{1}{\gamma-1}}}}{\gamma \cdot \alpha \cdot f_{\text{crit}}{}^{\gamma-1} + \beta} \tag{8.25}$$

Summarizing the expressions from Equation (8.24) and Equation (8.25), the approximation factor of DLTF-SFA for energy minimization when we assume *negligible overhead for sleeping*, defined as $\mathrm{AF}^{\text{energy}~\cancel{\text{overheads}}}_{\text{DLTF-SFA}}$, is presented in Theorem 1.

Theorem 1 *The approximation factor of DLTF-SFA for energy minimization, when we assume* negligible overhead for sleeping*, against the optimal energy consumption for the optimal task partition and the optimal DVFS schedule, is expressed as*

$$AF^{energy~\cancel{overheads}}_{DLTF\text{-}SFA} \leq \max\left\{ \frac{\alpha \cdot f_{crit}{}^{\gamma-1} \cdot \gamma \cdot U(\delta^{max}) + \beta + \frac{\kappa}{f_{crit}[\gamma \cdot U(\delta^{max})]^{\frac{1}{\gamma-1}}}}{\gamma \cdot \alpha \cdot f_{crit}{}^{\gamma-1} + \beta}, \right.$$
$$\left. \frac{\alpha \cdot \theta_{LTF}{}^{\gamma-1} \cdot f_{crit}{}^{\gamma-1} \cdot \gamma \cdot U\left(\frac{4M+1}{6M}\right) + \beta + \frac{\kappa}{\theta_{LTF} \cdot f_{crit}\left[\gamma \cdot U\left(\frac{4M+1}{6M}\right)\right]^{\frac{1}{\gamma-1}}}}{\gamma \cdot \alpha \cdot f_{crit}{}^{\gamma-1} + \beta} \right\}.$$

Proof. The theorem is simply proven by taking the maximum between Equation (8.24) and Equation (8.25). □

For systems with *non-negligible overhead for sleeping* (i.e., for the practical scenario in real systems), DLTF-SFA can be further combined with any uni-core DPM scheme to decide when to switch a core into a low-power mode. Particularly, we proceed to analyze the approximation factor of combining DLTF-SFA with the LTR algorithm from [37] for selecting the DPM schedule (i.e, LTR is used to decide whether/when each individual core should sleep/activate), against the optimal energy consumption for the optimal solution (i.e., the optimal task partition, the optimal DVFS schedule, and the optimal DPM schedule). There are mainly two cases that should be considered: (1) the total cycle utilization is smaller than f_{crit}, i.e., $\sum_{i=1}^{M} w^{\text{DLTF}}_i < f_{\text{crit}}$, and (2) the total cycle utilization is no less than f_{crit}, i.e., $\sum_{i=1}^{M} w^{\text{DLTF}}_i \geq f_{\text{crit}}$. For the first case, the DLTF task partitioning scheme assigns all the tasks to a single core and executes them at the critical frequency, making this a uniprocessor scheduling problem. For such a case, as proven in [37], the LTR algorithm achieves a 2-approximation from the optimal solution. For the second case, i.e., $\sum_{i=1}^{M} w^{\text{DLTF}}_i \geq f_{\text{crit}}$, for notational brevity, we isolate certain portions of the energy consumption for a certain schedule Ψ, as done in [37]:

- $\text{energy}(\Psi)$: The total energy consumed by schedule Ψ during a hyper-period.
- $\text{active}(\Psi)$: The energy consumed while the system is active, i.e., the energy consumption for *executing tasks*.
- $\text{idle}(\Psi)$: The energy consumption for keeping the system active or enter and leave a low-power mode during idle periods (depending on which action is more energy efficient).
- $\text{on}(\Psi)$: The energy consumption for keeping the system in the *on* state while the system is on.
- $\text{sleep}(\Psi)$: The energy consumption to leave the low-power mode at the end of each sleep interval.

Furthermore, we define Ψ_{OPT} as an optimal schedule, where $\Psi_{\text{OPT},m}$ is the corresponding schedule by considering only the m-th core. Similarly, we define $\Psi^{\text{SFA}}_{\text{LTR}}$ as the schedule that uses SFA for executing tasks and LTR for sleeping, where $\Psi^{\text{SFA}}_{\text{LTR},m}$ is the corresponding schedule by considering only the m-th core. Using this notation, according to Theorem 1, we know that

$$\text{active}\left(\Psi^{\text{SFA}}_{\text{LTR}}\right) \leq \mathrm{AF}^{\text{energy}~\cancel{\text{overheads}}}_{\text{DLTF-SFA}} \cdot \text{active}(\Psi_{\text{OPT}}). \tag{8.26}$$

Additionally, independently from the task execution on a single core m, according to Lemma 10 from [37], adopting LTR on a core ensures that $\text{idle}\left(\Psi^{\text{SFA}}_{\text{LTR},m}\right) \leq \text{on}\left(\Psi_{\text{OPT},m}\right) + 2 \cdot \text{sleep}\left(\Psi_{\text{OPT},m}\right)$. Therefore, considering the summation of all the cores in the cluster, we have that

$$\text{idle}\left(\Psi^{\text{SFA}}_{\text{LTR}}\right) \leq \text{on}\left(\Psi_{\text{OPT}}\right) + 2 \cdot \text{sleep}\left(\Psi_{\text{OPT}}\right). \tag{8.27}$$

Finally, considering Equation (8.26) and Equation (8.27), Theorem 2 presents the approximation factor of DLTF-SFA combined with LTR for energy minimization, when considering *non-negligible overhead for sleeping*.

Theorem 2 *The approximation factor of DLTF-SFA combined with LTR for energy minimization, against the optimal energy consumption for the optimal solution (i.e., the optimal task partition, the optimal DVFS schedule, and the optimal DPM schedule), is expressed as*

$$AF^{energy}_{DLTF\text{-}SFA} \leq AF^{energy\ \cancel{overheads}}_{DLTF\text{-}SFA} + 1.$$

Proof. From Equation (8.26) and Equation (8.27), and by considering that by definition it holds that $\text{AF}^{\text{energy}\ \cancel{\text{overheads}}}_{\text{DLTF-SFA}} \geq 1$, we have that

$$\begin{aligned}
\text{energy}\left(\Psi^{\text{SFA}}_{\text{LTR}}\right) &= \text{active}\left(\Psi^{\text{SFA}}_{\text{LTR}}\right) + \text{idle}\left(\Psi^{\text{SFA}}_{\text{LTR}}\right)\\
&\leq \text{AF}^{\text{energy}\ \cancel{\text{overheads}}}_{\text{DLTF-SFA}} \cdot \text{active}\left(\Psi_{\text{OPT}}\right) + \text{on}\left(\Psi_{\text{OPT}}\right) + 2 \cdot \text{sleep}\left(\Psi_{\text{OPT}}\right)\\
&\leq \text{AF}^{\text{energy}\ \cancel{\text{overheads}}}_{\text{DLTF-SFA}} \cdot \text{active}\left(\Psi_{\text{OPT}}\right) + \text{active}\left(\Psi_{\text{OPT}}\right) + \text{idle}\left(\Psi_{\text{OPT}}\right) + \text{sleep}\left(\Psi_{\text{OPT}}\right)\\
&\leq \left(\text{AF}^{\text{energy}\ \cancel{\text{overheads}}}_{\text{DLTF-SFA}} + 1\right) \cdot \text{active}\left(\Psi_{\text{OPT}}\right) + 2 \cdot \text{idle}\left(\Psi_{\text{OPT}}\right)\\
&\leq \max\left\{\text{AF}^{\text{energy}\ \cancel{\text{overheads}}}_{\text{DLTF-SFA}} + 1, 2\right\} \cdot \text{energy}\left(\Psi_{\text{OPT}}\right)\\
&\leq \left(\text{AF}^{\text{energy}\ \cancel{\text{overheads}}}_{\text{DLTF-SFA}} + 1\right) \cdot \text{energy}\left(\Psi_{\text{OPT}}\right).
\end{aligned}$$

Therefore, the theorem is proven. □

Finally, from Theorem 1 and Theorem 2, for a given hardware platform with a known power model, functions $U\left(\delta^{\max}\right)$, $U\left(\frac{4M+1}{6M}\right)$, and θ_{LTF} depend only on the number of cores in the cluster M. Therefore, we need to explore the impact of M on the approximation factor of DLTF-SFA for energy minimization, as shown in Figure 8.12 based on Theorem 1 and Theorem 2. Theoretically, the approximation factor could go up to ∞ when $M \to \infty$. Nevertheless, in real systems, the number of cores in a cluster is not a very large number. Particularly, we do not expect next-generation platforms to have more than 8 to 16 cores per cluster; however, Figure 8.12 shows the approximation factor up to 56 cores for us to observe where Equation (8.24) and Equation (8.25), inside Theorem 1, cross each other.

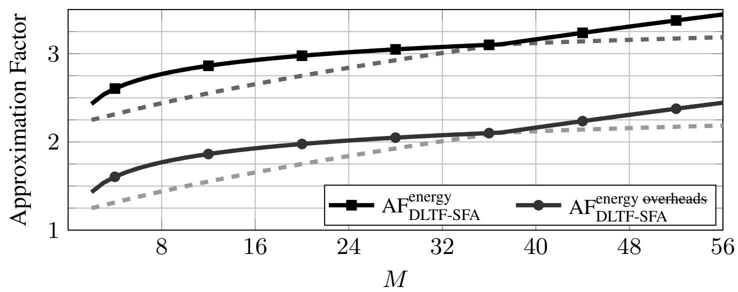


Figure 8.12: Example of the approximation factor for DLTF-SFA in terms of energy consumption when $\gamma = 3$, $\alpha = 0.27\frac{\text{W}}{\text{GHz}^3}$, $\beta = 0.52\frac{\text{W}}{\text{GHz}}$, and $\kappa = 0.5\,\text{W}$ (i.e., for 22 nm OOO Alpha 21264 cores, as detailed in Chapter 3.3).

8.3.2 Peak Power Reduction Analysis for DLTF-SFA

This section presents the approximation factor analysis for DLTF-SFA in terms of peak power reduction. i.e., the worst-case behavior of DLTF-SFA for peak power reduction, against the optimal peak power consumption for the optimal solution (i.e., the optimal task partition, the optimal DVFS schedule, and the optimal DPM schedule). For such a purpose, we first derive Lemma 6, in which we find an upper bound for the resulting number of active cores when using DLTF for task partitioning (i.e., $M^{\neq 0}$) with respect to the total cycle utilization. Namely, in case that DLTF did not manage to regroup the tasks such that one or more cores can be always put in a low-power mode, then the worst case for $M^{\neq 0}$ is clearly M, as there cannot be more active cores under DLTF than cores in the cluster. Nevertheless, if DLTF was successful in its regrouping procedure, then there are possibly less active cores under DLTF than total cores in the cluster, and the number of active cores can be upper bounded according to the total cycle utilization, as detailed in Lemma 6.

Lemma 6 *For a given w_M^{DLTF} and total cycle utilization $\sum_{i=1}^{M} w_i^{DLTF}$, an upper bound of $M^{\neq 0}$ when using DLTF for task partitioning can be expressed as*

$$M^{\neq 0} \leq \begin{cases} \min\left\{M, 2 \cdot \frac{\sum_{i=1}^{M} w_i^{DLTF}}{f_{crit}}\right\} & \text{if } w_M^{DLTF} \leq f_{crit} \\ \min\left\{M, 2 \cdot \frac{\sum_{i=1}^{M} w_i^{DLTF}}{w_M^{DLTF}} - 1\right\} & \text{if } w_M^{DLTF} > f_{crit}. \end{cases}$$

Proof. Algorithm DLTF is mainly a first-fit bin packing algorithm. Hence, starting from the initial solution of LTF, it attempts to pack the tasks into the minimum amount of bins of size $w_{\max}^{\text{DLTF}} = \max\left\{f_{\text{crit}}, w_M^{\text{LTF}}\right\}$. This implies that after the regrouping, in case that $w_M^{\text{LTF}} > f_{\text{crit}}$ in the initial solution of LTF, then task set M remains unchanged, such that $\mathbf{S}_M^{\text{DLTF}} = \mathbf{S}_M^{\text{LTF}}$ and $w_M^{\text{DLTF}} = w_M^{\text{LTF}} > f_{\text{crit}}$. Furthermore, in case $w_M^{\text{LTF}} \leq f_{\text{crit}}$, then most likely $\mathbf{S}_M^{\text{DLTF}} \neq \mathbf{S}_M^{\text{LTF}}$, but it will hold that $w_M^{\text{DLTF}} \leq f_{\text{crit}}$. From the properties of the first-fit bin packing algorithm [102], it holds that $M^{\neq 0} \leq \min\left\{M, 2\frac{\sum_{i=1}^{M} w_i^{\text{DLTF}}}{f_{\text{crit}}}\right\}$ in case that $w_M^{\text{DLTF}} \leq f_{\text{crit}}$, and it holds that $M^{\neq 0} \leq \min\left\{M, 1 + 2\frac{\sum_{i=1}^{M} w_i^{\text{DLTF}} - w_M^{\text{DLTF}}}{w_M^{\text{DLTF}}}\right\}$ in case that $w_M^{\text{DLTF}} > f_{\text{crit}}$. Thus, the lemma is proven. □

Similarly as in Section 8.3.1, we start by replacing the different possible values of $\hat{P}_{\downarrow}^{*}$ and $\hat{P}_{\text{SFA}}^{\text{DLTF}}$ from Equation (8.22) and Equation (8.9) inside Equation (8.2), such that $\text{AF}_{\text{DLTF-SFA}}^{\text{peak power}}$ can be expressed as shown in Equation (8.28). Note that, since we are focusing on peak power and not on energy consumption, the *overheads for sleeping* do not play any role in this analysis.

$$\text{AF}_{\text{DLTF-SFA}}^{\text{peak power}} \leq \max \left\{ \begin{cases} \frac{\alpha \cdot f_{\text{crit}}{}^{\gamma} + \beta \cdot f_{\text{crit}} + \kappa}{\alpha \cdot w_M'{}^{\gamma}\left[1 + \frac{w_1'}{w_M'}\left(\sqrt[\gamma]{M} - 1\right)\right]^{\gamma} + \beta \sum_{i=1}^{M} w_i' + \kappa} & \text{if } \sum_{i=1}^{M} w_i^{\text{DLTF}} \leq f_{\text{crit}} \\ \frac{M^{\neq 0}\left(\alpha \cdot f_{\text{crit}}{}^{\gamma} + \beta \cdot f_{\text{crit}} + \kappa\right)}{\alpha \cdot w_M'{}^{\gamma}\left[1 + \frac{w_1'}{w_M'}\left(\sqrt[\gamma]{M} - 1\right)\right]^{\gamma} + \beta \sum_{i=1}^{M} w_i' + \kappa} & \text{if } w_M^{\text{DLTF}} \leq f_{\text{crit}} \\ \frac{M^{\neq 0}\left(\alpha \cdot w_M^{\text{DLTF}\,\gamma} + \beta \cdot w_M^{\text{DLTF}} + \kappa\right)}{\alpha \cdot w_M'{}^{\gamma}\left[1 + \frac{w_1'}{w_M'}\left(\sqrt[\gamma]{M} - 1\right)\right]^{\gamma} + \beta \sum_{i=1}^{M} w_i' + \kappa} & \text{otherwise} \end{cases} \right\} \tag{8.28}$$

Now, the approximation factor comes from finding the worst-case ratios for each possible condition inside Equation (8.28), presented in the following lemmas. Particularly, Lemma 7 focuses on the case in which $\sum_{i=1}^{M} w_i^{\text{DLTF}} \leq f_{\text{crit}}$, Lemma 8 focuses on the case in which $\sum_{i=1}^{M} w_i^{\text{DLTF}} > f_{\text{crit}}$ and $w_M^{\text{DLTF}} \leq f_{\text{crit}}$, Lemma 9 focuses on the case in which $w_M^{\text{DLTF}} > f_{\text{crit}}$ and $w_M^{*} \geq w_M^{\text{DLTF}}$ (for the first case of the relation between w_M' and w_M^{DLTF} from Lemma 1), and Lemma 10 focuses on the case in which $w_M^{\text{DLTF}} > f_{\text{crit}}$ and $w_M^{*} < w_M^{\text{DLTF}}$ (for the second case of the relation between w_M' and w_M^{DLTF} from Lemma 1). Finally, Theorem 3 summarizes these lemmas by taking the maximum among all of them.

Lemma 7 *When $\sum_{i=1}^{M} w_i^{DLTF} \leq f_{crit}$, the approximation factor of DLTF-SFA for peak power reduction is expressed as*

$$AF_{DLTF\text{-}SFA}^{peak\ power \star 1} \leq \frac{\alpha \cdot f_{crit}{}^{\gamma} + \beta \cdot f_{crit} + \kappa}{\kappa}.$$

Proof. For this case, given that α, β, κ, γ, and f_{crit} are all constants, it holds that $\alpha \cdot f_{\text{crit}}{}^{\gamma} + \beta \cdot f_{\text{crit}} + \kappa$ in Equation (8.28) is also constant. Therefore, the worst case occurs when $\sum_{i=1}^{M} w_i^{\text{DLTF}} \to 0$, such that there is only independent power consumption in the lower bound, and thus the lemma is proven. □

Lemma 8 *When $\sum_{i=1}^{M} w_i^{DLTF} > f_{crit}$ and $w_M^{DLTF} \leq f_{crit}$, the approximation factor of DLTF-SFA for peak power reduction is expressed as*

$$AF_{DLTF\text{-}SFA}^{peak\,power\star 2} \leq \frac{\min\left\{\frac{M}{w'_M}, \frac{2(1-\delta+\delta M)}{f_{crit}}\right\} \frac{\alpha \cdot {f_{crit}}^{\gamma} + \beta \cdot f_{crit} + \kappa}{1-\delta+\delta M}}{\alpha \cdot {w'_M}^{\gamma-1} \cdot \frac{1}{U(\delta)} + \beta + \frac{\kappa}{w'_M(1-\delta+\delta M)}}.$$

For every value of M, we compute the maximum $AF_{DLTF\text{-}SFA}^{peak\,power\star 2}$ among all δ and all w'_M, such that $0 \leq \delta \leq 1$ and $\frac{f_{crit}}{1-\delta+\delta M} < w'_M \leq f_{crit}$.

Proof. For this case, inside Equation (8.28), we replace $\sum_{i=1}^{M} w_i^{\text{DLTF}}$ with $w'_M(1-\delta+\delta M)$, we replace $M^{\neq 0}$ with $\min\left\{M, 2 \cdot \frac{\sum_{i=1}^{M} w_i^{\text{DLTF}}}{f_{\text{crit}}}\right\}$ (according to Lemma 6), we replace $\frac{w'_1}{w'_M}$ with δ from Equation (8.21), and we replace $\frac{1-\delta+\delta \cdot M}{\left(1-\delta+\delta \cdot \sqrt[\gamma]{M}\right)^{\gamma}}$ with $U(\delta)$ from Equation (8.20). Thus, we reach the expression in the statement of the lemma and the lemma is proven. □

Lemma 9 *When $w_M^{DLTF} > f_{crit}$ and $w_M^* \geq w_M^{DLTF}$, the approximation factor of DLTF-SFA for peak power reduction is expressed as*

$$AF_{DLTF\text{-}SFA}^{peak\,power\star 3} \leq \frac{\min\{M, 1+2(M-1)\delta\} \frac{\alpha \cdot {w'_M}^{\gamma} + \beta \cdot w'_M + \kappa}{1-\delta+\delta M}}{\alpha \cdot {w'_M}^{\gamma} \cdot \frac{1}{U(\delta)} + \beta \cdot w'_M + \frac{\kappa}{1-\delta+\delta M}}.$$

For every value of M, we compute the maximum $AF_{DLTF\text{-}SVA}^{peak\,power\star 3}$ among all δ and all w'_M, such that $0 \leq \delta \leq 1$ and $f_{crit} < w'_M \leq F_{max}$.

Proof. For this case, similar to the proof of Lemma 8, inside Equation (8.28), we replace $\sum_{i=1}^{M} w_i^{\text{DLTF}}$ with $w'_M(1-\delta+\delta M)$, we replace $M^{\neq 0}$ with $\min\left\{M, 2 \cdot \frac{\sum_{i=1}^{M} w_i^{\text{DLTF}}}{w_M^{\text{DLTF}}} - 1\right\}$ (according to Lemma 6), we replace $\frac{w'_1}{w'_M}$ with δ from Equation (8.21), and we replace $\frac{1-\delta+\delta \cdot M}{\left(1-\delta+\delta \cdot \sqrt[\gamma]{M}\right)^{\gamma}}$ with $U(\delta)$ from Equation (8.20). Furthermore, from Lemma 1 we have that $w'_M = w_M^{\text{DLTF}}$. Thus, we reach the expression in the statement of the lemma and the lemma is proven. □

Lemma 10 *When $w_M^{DLTF} > f_{crit}$ and $w_M^* < w_M^{DLTF}$, the approximation factor of DLTF-SFA for peak power reduction is expressed as*

$$AF_{DLTF\text{-}SFA}^{peak\,power\star 4} \leq \frac{M \cdot \alpha \cdot (\theta_{LTF} \cdot w'_M)^{\gamma} + M \cdot \beta \cdot \theta_{LTF} \cdot w'_M + M \cdot \kappa}{\alpha \cdot {w'_M}^{\gamma} \cdot \frac{1-\delta+\delta M}{U(\delta)} + \beta \cdot w'_M(1-\delta+\delta M) + \kappa}.$$

For every value of M, we compute the maximum $AF_{DLTF\text{-}SVA}^{peak\,power\star 4}$ among all δ and w'_M, such that $\frac{4M+1}{6M} \leq \delta \leq 1$ and $\frac{f_{crit}}{\theta_{LTF}} < w'_M \leq F_{max}$.

Proof. For this case, similar to the proof of Lemma 9, inside Equation (8.28), we replace $\sum_{i=1}^{M} w_i^{\text{DLTF}}$ with $w'_M(1-\delta+\delta M)$, we replace $\frac{w'_1}{w'_M}$ with δ from Equation (8.21), and we replace $\frac{1-\delta+\delta \cdot M}{\left(1-\delta+\delta \cdot \sqrt[\gamma]{M}\right)^{\gamma}}$ with $U(\delta)$ from Equation (8.20). Moreover, according to Equation (8.6), it holds that $\frac{w_1^{\text{LTF}}}{w_M^{\text{LTF}}} \geq \frac{1}{2}$ if after partitioning the tasks with LTF there are *at least two* tasks in resulting task set $\mathbf{S}_M^{\text{LTF}}$. Hence, for such a case it also holds that

$$w_1^{\text{LTF}} \geq \frac{w_M^{\text{LTF}}}{2} \quad \Rightarrow \quad \sum_{i=1}^{M} w_i^{\text{LTF}} \geq w_1^{\text{LTF}}(M-1) + w_M^{\text{LTF}} \geq \frac{w_M^{\text{LTF}}(M-1)}{2} + w_M^{\text{LTF}}$$
$$\Rightarrow \quad \sum_{i=1}^{M} w_i^{\text{LTF}} \geq \frac{M+1}{2} w_M^{\text{LTF}}.$$

Given that according to Section 8.1.3, when $w_M^{\text{DLTF}} > f_{\text{crit}}$ it holds that $w_M^{\text{DLTF}} = w_M^{\text{LTF}}$, and the total cycle utilization is constant such that $\sum_{i=1}^{M} w_i^{\text{DLTF}} = \sum_{i=1}^{M} w_i^{\text{LTF}}$, then it holds that $M \leq 2\frac{\sum_{i=1}^{M} w_i^{\text{DLTF}}}{w_M^{\text{DLTF}}} - 1$ for all $M \geq 1$. Therefore, $M^{\neq 0}$ is set to M, and since $M^{\neq 0} = M$ is the worst case for $M^{\neq 0}$, then the other case in which there is *only one* task in $\mathbf{S}_M^{\text{LTF}}$ after applying LTF does not need to be considered.

Furthermore, from Lemma 1, we have that $w'_M = \max\left\{\frac{w_M^{\text{DLTF}}}{\theta_{\text{LTF}}}, \frac{\sum_{i=1}^{M} w_i^*}{M}\right\}$, where case $\frac{\sum_{i=1}^{M} w_i^*}{M}$ is only considered if the cycle utilization adjustment from Lemma 1 does not reach $\frac{w_M^{\text{DLTF}}}{\theta_{\text{LTF}}}$ without reducing w'_M below the average cycle utilization. This means that $w'_M = \frac{w_M^{\text{DLTF}}}{\theta_{\text{LTF}}}$ is the worst case for the relation between w'_M and w_M^{DLTF}, and hence the other case is not considered. With these considerations, $\text{AF}_{\text{DLTF-SFA}}^{\text{peak power}\star 4}$ is expressed as

$$\begin{aligned}
\text{AF}_{\text{DLTF-SFA}}^{\text{peak power}\star 4} &\leq \frac{M^{\neq 0}\left(\alpha \cdot {w_M^{\text{DLTF}}}^{\gamma} + \beta \cdot w_M^{\text{DLTF}} + \kappa\right)}{\alpha \cdot {w'_M}^{\gamma}\left[1 + \frac{w'_1}{w'_M}\left(\sqrt[\gamma]{M} - 1\right)\right]^{\gamma} + \beta \sum_{i=1}^{M} w'_i + \kappa} \\
&\leq \frac{M\left(\alpha \cdot {w_M^{\text{DLTF}}}^{\gamma} + \beta \cdot w_M^{\text{DLTF}} + \kappa\right)}{\alpha \cdot {w'_M}^{\gamma}\left(1 - \delta + \delta\sqrt[\gamma]{M}\right)^{\gamma} + \beta \cdot w'_M\left(1 - \delta + \delta M\right) + \kappa} \\
&\leq \frac{M \cdot \alpha \cdot \left(\theta_{\text{LTF}} \cdot w'_M\right)^{\gamma} + M \cdot \beta \cdot \theta_{\text{LTF}} \cdot w'_M + M \cdot \kappa}{\alpha \cdot {w'_M}^{\gamma}\frac{1-\delta+\delta M}{U(\delta)} + \beta \cdot w'_M\left(1 - \delta + \delta M\right) + \kappa}
\end{aligned}$$

Finally, from Lemma 4, we have that in case that $w_M^* < w_M^{\text{DLTF}}$, then it holds that $\delta = \frac{\sum_{i=1}^{M-1} w'_i}{w'_M(M-1)} \geq \frac{4M+1}{6M}$. Thus, the lemma is proven. □

Theorem 3 *The approximation factor of DLTF-SFA for peak power reduction, against the optimal peak power consumption for the optimal solution (i.e., the optimal task partition, the optimal DVFS schedule, and the optimal DPM schedule), is expressed as*

$$AF_{\textit{DLTF-SFA}}^{\textit{peak power}} \leq \max\left\{AF_{\textit{DLTF-SFA}}^{\textit{peak power}\star 1}, AF_{\textit{DLTF-SFA}}^{\textit{peak power}\star 2}, AF_{\textit{DLTF-SFA}}^{\textit{peak power}\star 3}, AF_{\textit{DLTF-SFA}}^{\textit{peak power}\star 4}\right\},$$

according to the definitions in Lemmas 7, 8, 9, and 10.

Proof. This comes simply from taking the maximum among all cases from Lemmas 7, 8, 9, and 10. □

Figure 8.13 shows some examples of $\text{AF}_{\text{DLTF-SFA}}^{\text{peak power}}$ for different values of M.

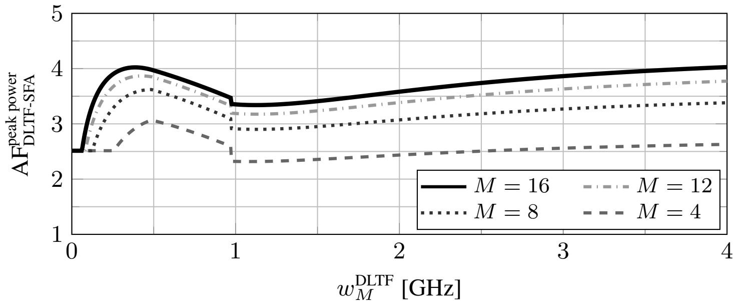


Figure 8.13: Example of the approximation factor for DLTF-SFA in terms of peak power consumption, for different values of M, when $\gamma = 3$, $\alpha = 0.27\frac{\text{W}}{\text{GHz}^3}$, $\beta = 0.52\frac{\text{W}}{\text{GHz}}$, and $\kappa = 0.5\,\text{W}$ (i.e., for 22 nm OOO Alpha 21264 cores, as detailed in Chapter 3.3).

8.4 Approximation Factor Analysis: DLTF-SVA

8.4.1 Energy Minimization Analysis for DLTF-SVA

This section presents the approximation factor analysis of DLTF-SVA in terms of energy consumption, i.e., the worst-case behavior of DLTF-SVA for energy minimization, against the optimal solution. Similarly as in

Section 8.3.1, we start by replacing the different possible values of $E^*_{\downarrow}$ and $E^{\text{DLTF}}_{\text{SVA}}$ from Equation (8.19) and Equation (8.12) inside Equation (8.3), such that $\text{AF}^{\text{energy}}_{\text{DLTF-SVA}}$ can be expressed as shown in Equation (8.29), for which Figure 8.14 shows an example (including an example of DLTF-SFA for comparison). Note that, since under SVA all the cores are executing tasks at all times in case that all the tasks require their worst-case execution times in order to finish every task instance, there is no need for the adopted DPM technique to place cores in a low-power mode in order to reduce the energy consumption for idling, and hence the *overheads for sleeping* do not play any role in this analysis.

$$\text{AF}^{\text{energy}}_{\text{DLTF-SVA}} \leq \max \left\{ \begin{cases} 1 & \text{if } \sum_{i=1}^{M} w_i^{\text{DLTF}} \leq f_{\text{crit}} \\ \frac{\alpha \cdot {f_{\text{crit}}}^{\gamma-1} \cdot \sum_{i=1}^{M} w_i^{\text{DLTF}} + M^{\neq 0}(\beta \cdot f_{\text{crit}} + \kappa)}{(\alpha \cdot \gamma \cdot {f_{\text{crit}}}^{\gamma-1} + \beta) \sum_{i=1}^{M} w_i'} & \text{if } w_M^{\text{DLTF}} \leq f_{\text{crit}} \text{ and } w_M^* \leq f_{\text{dyn}} \\ \frac{\alpha \cdot {w_M^{\text{DLTF}}}^{\gamma-1} \cdot \sum_{i=1}^{M} w_i^{\text{DLTF}} + M^{\neq 0}(\beta \cdot w_M^{\text{DLTF}} + \kappa)}{(\alpha \cdot \gamma \cdot {f_{\text{crit}}}^{\gamma-1} + \beta) \sum_{i=1}^{M} w_i'} & \text{if } w_M^{\text{DLTF}} > f_{\text{crit}} \text{ and } w_M^* \leq f_{\text{dyn}} \\ \frac{\alpha \cdot {w_M^{\text{DLTF}}}^{\gamma-1} \cdot \sum_{i=1}^{M} w_i^{\text{DLTF}} + M^{\neq 0}(\beta \cdot w_M^{\text{DLTF}} + \kappa)}{\alpha \cdot {w_M'}^{\gamma} \left[1 + \frac{w_1'}{w_M'} \left(\sqrt[\gamma]{M} - 1 \right) \right]^{\gamma} + \beta \sum_{i=1}^{M} w_i'} & \text{otherwise} \end{cases} \right\} \quad (8.29)$$

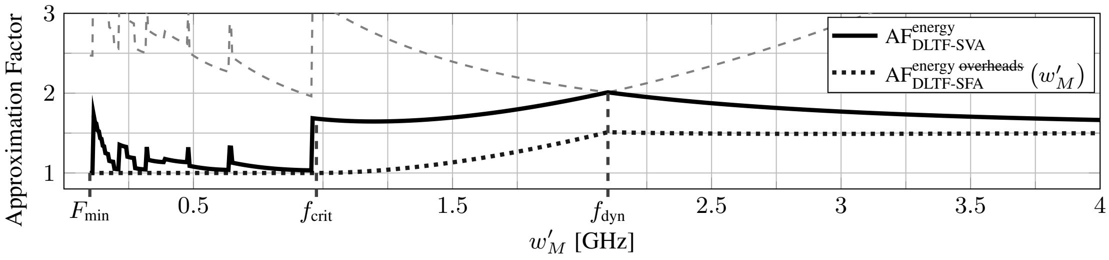


Figure 8.14: Example of $\text{AF}^{\text{energy}}_{\text{DLTF-SVA}}$ and $\text{AF}^{\text{energy}~\text{overheads}}_{\text{DLTF-SFA}}(w'_M)$, when $\gamma = 3$, $\alpha = 0.27 \frac{\text{W}}{\text{GHz}^3}$, $\beta = 0.52 \frac{\text{W}}{\text{GHz}}$, $\kappa = 0.5\,\text{W}$, and $M = 16$, with $w_i' = w_i^{\text{DLTF}} = 0.51 \cdot w_M'$ for all $i = 1, \ldots, M-1$, and $w_M' = w_M^{\text{DLTF}}$.

Now, similarly to Section 8.3.1 and Section 8.3.2, by considering Lemma 2 and Lemma 6, the approximation factor comes from finding the worst-case ratios for each possible condition inside Equation (8.29), presented in the following lemmas. Particularly, Lemma 11 focuses on the case in which $w_M^{\text{DLTF}} \leq f_{\text{crit}}$, $w_M^* \leq f_{\text{dyn}}$, and $\sum_{i=1}^{M} w_i^{\text{DLTF}} > f_{\text{crit}}$. Lemma 12 focuses on the case in which $w_M^{\text{DLTF}} > f_{\text{crit}}$, $w_M^* \leq f_{\text{dyn}}$, and $w_M^* \geq w_M^{\text{DLTF}}$. Lemma 13 focuses on the case in which $w_M^{\text{DLTF}} > f_{\text{crit}}$, $w_M^* \leq f_{\text{dyn}}$, and $w_M^* < w_M^{\text{DLTF}}$. Finally, Theorem 4 summarizes these lemmas by taking the maximum among all of them.

Lemma 11 *When $w_M^{DLTF} \leq f_{crit}$, $w_M^* \leq f_{dyn}$, and $\sum_{i=1}^{M} w_i^{DLTF} > f_{crit}$, the approximation factor of DLTF-SVA for energy minimization is expressed as*

$$AF^{energy}_{DLTF\text{-}SVA}{}^{\star 1} \leq \frac{\alpha \cdot {f_{crit}}^{\gamma-1} + 2 \cdot \beta + \frac{2 \cdot \kappa}{f_{crit}}}{\alpha \cdot \gamma \cdot {f_{crit}}^{\gamma-1} + \beta}$$

Proof. For this case, by replacing $M^{\neq 0}$ with $\min\left\{ M, 2 \cdot \frac{\sum_{i=1}^{M} w_i^{\text{DLTF}}}{f_{\text{crit}}} \right\}$ (from Lemma 6) in Equation (8.29), we have that

$$\text{AF}^{\text{energy}}_{\text{DLTF-SVA}}{}^{\star 1} \leq \frac{\alpha \cdot {f_{\text{crit}}}^{\gamma-1} + \min\left\{ \frac{M}{\sum_{i=1}^{M} w_i^{\text{DLTF}}}, \frac{2}{f_{\text{crit}}} \right\} (\beta \cdot f_{\text{crit}} + \kappa)}{\alpha \cdot \gamma \cdot {f_{\text{crit}}}^{\gamma-1} + \beta}.$$

Given that α, β, κ, γ, and f_{crit} are all constants, the above expression is maximized if and only if the *min* function is maximized. Hence, since $\frac{2}{f_{\text{crit}}}$ is constant, and this *min* function simply considers the minimum value between $\frac{M}{\sum_{i=1}^{M} w_i^{\text{DLTF}}}$ and $\frac{2}{f_{\text{crit}}}$, then clearly the maximum value this function can take is $\frac{2}{f_{\text{crit}}}$, thus proving the lemma by reaching the expression in its statement. □

Lemma 12 *When $w_M^{DLTF} > f_{crit}$, $w_M^* \leq f_{dyn}$, and $w_M^* \geq w_M^{DLTF}$, the approximation factor of DLTF-SVA for energy minimization is expressed as*

$$AF_{DLTF\text{-}SVA}^{energy\ \star 2} \leq \frac{\alpha \cdot \gamma \cdot U(\delta) \cdot f_{crit}{}^{\gamma-1} + \frac{\min\{M, 1+2(M-1)\delta\}}{1-\delta+\delta M}\left(\beta + \frac{\kappa}{f_{crit} \cdot [\gamma \cdot U(\delta)]^{\frac{1}{\gamma-1}}}\right)}{\alpha \cdot \gamma \cdot f_{crit}{}^{\gamma-1} + \beta}.$$

For every value of M, we compute the maximum $AF_{DLTF\text{-}SVA}^{energy\ \star 2}$ among all δ, such that $0 \leq \delta \leq 1$.

Proof. For this case, inside Equation (8.29), we replace $M^{\neq 0}$ with $\min\left\{M, 2\frac{\sum_{i=1}^{M} w_i^{\text{DLTF}}}{w_M^{\text{DLTF}}} - 1\right\}$ (according to Lemma 6), we replace w_M^{DLTF} with w'_M (according to Lemma 1), and we replace $\sum_{i=1}^{M} w_i^{\text{DLTF}}$ with $w'_M(1-\delta+\delta M)$. Moreover, we replace w'_M with f_{dyn} according to Lemma 2, such that $E_{\downarrow}^*$ becomes a continuous function. Thus, considering the definitions of δ from Equation (8.21) and $U(\delta)$ from Equation (8.20), we prove the lemma by reaching the expression in its statement. □

Lemma 13 *When $w_M^{DLTF} > f_{crit}$, $w_M^* \leq f_{dyn}$, and $w_M^* < w_M^{DLTF}$, the approximation factor of DLTF-SVA for energy minimization is expressed as*

$$AF_{DLTF\text{-}SVA}^{energy\ \star 3} \leq \frac{\alpha \cdot \gamma \cdot U(\delta) \cdot (\theta_{LTF} \cdot f_{crit})^{\gamma-1} + \frac{M \cdot \beta \cdot \theta_{LTF} + \frac{M \cdot \kappa}{f_{crit}[\gamma \cdot U(\delta)]^{\frac{1}{\gamma-1}}}}{1-\delta+\delta M}}{\alpha \cdot \gamma \cdot f_{crit}{}^{\gamma-1} + \beta},$$

For every value of M, we compute the maximum $AF_{DLTF\text{-}SVA}^{energy\ \star 3}$ among all δ, such that $\frac{4M+1}{6M} \leq \delta \leq 1$.

Proof. Similar to the proof of Lemma 12, inside Equation (8.29), we replace $\sum_{i=1}^{M} w_i^{\text{DLTF}}$ with $w'_M(1-\delta+\delta M)$, and we replace w'_M with f_{dyn} according to Lemma 2, such that $E_{\downarrow}^*$ becomes a continuous function. From Lemma 1 we have that $w'_M = \max\left\{\frac{w_M^{\text{DLTF}}}{\theta_{\text{LTF}}}, \frac{\sum_{i=1}^{M} w_i^*}{M}\right\}$, and from Lemma 4 we have that $\delta = \frac{\sum_{i=1}^{M-1} w'_i}{w'_M(M-1)} \geq \frac{4M+1}{6M}$. Moreover, as shown in the proof of Lemma 10, for this case it holds that $M \leq 2\frac{\sum_{i=1}^{M} w_i^{\text{DLTF}}}{w_M^{\text{DLTF}}} - 1$ for all $M \geq 1$, such that $M^{\neq 0}$ is set to M, which is the worst case for $M^{\neq 0}$. Finally, also as shown in the proof of Lemma 10, it holds that $w'_M = \frac{w_M^{\text{DLTF}}}{\theta_{\text{LTF}}}$ is the worst case for the relation between w'_M and w_M^{DLTF}. Thus, the lemma is proven. □

Theorem 4 *The approximation factor of DLTF-SVA for energy minimization, against the optimal peak power consumption for the optimal solution, is expressed as*

$$AF_{DLTF\text{-}SVA}^{energy} \leq \max\left\{AF_{DLTF\text{-}SVA}^{energy\ \star 1}, AF_{DLTF\text{-}SVA}^{energy\ \star 2}, AF_{DLTF\text{-}SVA}^{energy\ \star 3}\right\},$$

according to the definitions in Lemmas 11, 12, and 13.

Proof. This comes simply from taking the maximum among all cases from Lemmas 11, 12, and 13. □

Figure 8.15 shows an example of $\text{AF}_{\text{DLTF-SVA}}^{\text{energy}}$, for a power function modeled from simulations of a 22 nm OOO Alpha 21264 core, that results in power parameters $\gamma = 3$, $\alpha = 0.27\frac{\text{W}}{\text{GHz}^3}$, $\beta = 0.52\frac{\text{W}}{\text{GHz}}$, and $\kappa = 0.5\,\text{W}$ (as detailed in Chapter 3.3). For the given hardware parameters, Figure 8.15 shows that the approximation factor of DLTF-SVA in terms of energy consumption is at most 1.95 (2.21, 2.42, 2.59, respectively), when the cluster has up to 4 (8, 16, 32, respectively) cores.

8.4.2 Peak Power Reduction Analysis for DLTF-SVA

This section presents the approximation factor analysis for DLTF-SVA in terms of peak power reduction. i.e., the worst-case behavior of DLTF-SFA for peak power reduction, against the optimal peak power consumption for the optimal solution. Similarly as in Section 8.3.2, we start by replacing the different possible values of $\hat{P}_{\downarrow}^*$ and $\hat{P}_{\text{SVA}}^{\text{DLTF}}$ from Equation (8.22) and Equation (8.11) inside Equation (8.4), such that $\text{AF}_{\text{DLTF-SVA}}^{\text{peak power}}$ can be

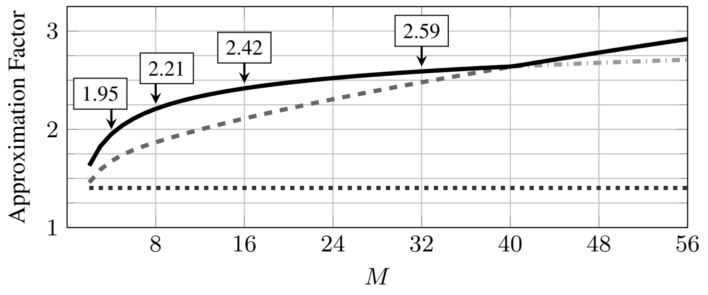


Figure 8.15: Example of the approximation factor for DLTF-SVA in terms of energy consumption when $\gamma = 3$, $\alpha = 0.27\frac{\text{W}}{\text{GHz}^3}$, $\beta = 0.52\frac{\text{W}}{\text{GHz}}$, and $\kappa = 0.5\,\text{W}$.

expressed as shown in Equation (8.30). As in Section 8.3.2, note that, since we are focusing on peak power and not on energy consumption, the *overheads for sleeping* do not play any role in this analysis.

$$\text{AF}^{\text{peak power}}_{\text{DLTF-SVA}} \leq \max \left\{ \begin{cases} \frac{\alpha \cdot {f_{\text{crit}}}^{\gamma} + \beta \cdot f_{\text{crit}} + \kappa}{\alpha \cdot {w'_M}^{\gamma} \left[1 + \frac{w'_1}{w'_M}\left(\sqrt[\gamma]{M} - 1\right)\right]^{\gamma} + \beta \sum_{i=1}^{M} w'_i + \kappa} & \text{if } \sum\limits_{i=1}^{M} w_i^{\text{DLTF}} \leq f_{\text{crit}} \\ \frac{\alpha \cdot {f_{\text{crit}}}^{\gamma-1} \sum_{i=1}^{M} w_i^{\text{DLTF}} + M^{\neq 0}(\beta \cdot f_{\text{crit}} + \kappa)}{\alpha \cdot {w'_M}^{\gamma} \left[1 + \frac{w'_1}{w'_M}\left(\sqrt[\gamma]{M} - 1\right)\right]^{\gamma} + \beta \sum_{i=1}^{M} w'_i + \kappa} & \text{if } w_M^{\text{DLTF}} \leq f_{\text{crit}} \\ \frac{\alpha \cdot {w_M^{\text{DLTF}}}^{\gamma-1} \sum_{i=1}^{M} w_i^{\text{DLTF}} + M^{\neq 0}\left(\beta \cdot w_M^{\text{DLTF}} + \kappa\right)}{\alpha \cdot {w'_M}^{\gamma} \left[1 + \frac{w'_1}{w'_M}\left(\sqrt[\gamma]{M} - 1\right)\right]^{\gamma} + \beta \sum_{i=1}^{M} w'_i + \kappa} & \text{otherwise} \end{cases} \right\} \tag{8.30}$$

Now, similarly as in Section 8.3.2, the approximation factor comes from finding the worst-case ratios for each possible condition inside Equation (8.30), presented in the following lemmas. Particularly, Lemma 14 focuses on the case in which $\sum_{i=1}^{M} w_i^{\text{DLTF}} \leq f_{\text{crit}}$, Lemma 15 focuses on the case in which $\sum_{i=1}^{M} w_i^{\text{DLTF}} > f_{\text{crit}}$ and $w_M^{\text{DLTF}} \leq f_{\text{crit}}$, Lemma 16 focuses on the case in which $w_M^{\text{DLTF}} > f_{\text{crit}}$ and $w_M^{*} \geq w_M^{\text{DLTF}}$ (for the first case of the relation between w'_M and w_M^{DLTF} from Lemma 1), and Lemma 17 focuses on the case in which $w_M^{\text{DLTF}} > f_{\text{crit}}$ and $w_M^{*} < w_M^{\text{DLTF}}$ (for the second case of the relation between w'_M and w_M^{DLTF} from Lemma 1). Finally, Theorem 5 summarizes these lemmas by taking the maximum among all of them.

Lemma 14 *When $\sum_{i=1}^{M} w_i^{DLTF} \leq f_{crit}$, the approximation factor of DLTF-SVA for peak power reduction is expressed as*

$$AF^{peak\ power \star 1}_{DLTF\text{-}SVA} \leq \frac{\alpha \cdot {f_{crit}}^{\gamma} + \beta \cdot f_{crit} + \kappa}{\kappa}.$$

Proof. This proof is equivalent to the proof of Lemma 7. □

Lemma 15 *When $\sum_{i=1}^{M} w_i^{DLTF} > f_{crit}$ and $w_M^{DLTF} \leq f_{crit}$, the approximation factor of DLTF-SVA for peak power reduction is expressed as*

$$AF^{peak\ power \star 2}_{DLTF\text{-}SVA} \leq \frac{\alpha \cdot {f_{crit}}^{\gamma-1} + \min\left\{\frac{M}{w'_M}, \frac{2(1-\delta+\delta M)}{f_{crit}}\right\} \frac{\beta \cdot f_{crit} + \kappa}{1-\delta+\delta M}}{\alpha \cdot {w'_M}^{\gamma-1} \cdot \frac{1}{U(\delta)} + \beta + \frac{\kappa}{w'_M(1-\delta+\delta M)}}.$$

For every value of M, we compute the maximum $AF^{peak\ power \star 2}_{DLTF\text{-}SVA}$ among all δ and all w'_M, such that $0 \leq \delta \leq 1$ and $\frac{f_{crit}}{1-\delta+\delta M} < w'_M \leq f_{crit}$.

Proof. This proof is similar to the proof of Lemma 8. For this case, inside Equation (8.30), we replace $\sum_{i=1}^{M} w_i^{\text{DLTF}}$ with $w'_M(1 - \delta + \delta M)$, we replace $M^{\neq 0}$ with $\min\left\{M, 2 \cdot \frac{\sum_{i=1}^{M} w_i^{\text{DLTF}}}{f_{\text{crit}}}\right\}$ (according to Lemma 6), we replace $\frac{w'_1}{w'_M}$ with δ from Equation (8.21), and we replace $\frac{1-\delta+\delta \cdot M}{\left(1-\delta+\delta \cdot \sqrt[\gamma]{M}\right)^{\gamma}}$ with $U(\delta)$ from Equation (8.20). Thus, we reach the expression in the statement of the lemma and the lemma is proven. □

Lemma 16 *When $w_M^{DLTF} > f_{crit}$ and $w_M^* \geq w_M^{DLTF}$, the approximation factor of DLTF-SVA for peak power reduction is expressed as*

$$AF_{DLTF\text{-}SVA}^{peak\,power\star 3} \leq \frac{\alpha \cdot {w'_M}^{\gamma} + \min\left\{M, 1 + 2\left(M-1\right)\delta\right\} \frac{\beta \cdot w'_M + \kappa}{1-\delta+\delta M}}{\alpha \cdot {w'_M}^{\gamma} \cdot \frac{1}{U(\delta)} + \beta \cdot w'_M + \frac{\kappa}{1-\delta+\delta M}}.$$

For every value of M, we compute the maximum $AF_{DLTF\text{-}SVA}^{peak\,power\star 3}$ among all δ and all w'_M, such that $0 \leq \delta \leq 1$ and $f_{crit} < w'_M \leq F_{max}$.

Proof. This proof is similar to the proof of Lemma 9. For this case, inside Equation (8.30), we replace $\sum_{i=1}^{M} w_i^{\text{DLTF}}$ with $w'_M\left(1-\delta+\delta M\right)$, we replace $M^{\neq 0}$ with $\min\left\{M, 2 \cdot \frac{\sum_{i=1}^{M} w_i^{\text{DLTF}}}{w_M^{\text{DLTF}}} - 1\right\}$, we replace $\frac{w'_1}{w'_M}$ with δ from Equation (8.21), and we replace $\frac{1-\delta+\delta\cdot M}{\left(1-\delta+\delta\cdot\sqrt[\gamma]{M}\right)^{\gamma}}$ with $U\left(\delta\right)$ from Equation (8.20). Furthermore, from Lemma 1 we have that $w'_M = w_M^{\text{DLTF}}$. Thus, we reach the expression in the statement of the lemma and the lemma is proven. □

Lemma 17 *When $w_M^{DLTF} > f_{crit}$ and $w_M^* < w_M^{DLTF}$, the approximation factor of DLTF-SVA for peak power reduction is expressed as*

$$AF_{DLTF\text{-}SVA}^{peak\,power\star 4} \leq \frac{\alpha \cdot {\theta_{LTF}}^{\gamma-1} \cdot {w'_M}^{\gamma} + M \cdot \frac{\beta \cdot \theta_{LTF} \cdot w'_M + \kappa}{1-\delta+\delta M}}{\alpha \cdot {w'_M}^{\gamma} \cdot \frac{1}{U(\delta)} + \beta \cdot w'_M + \frac{\kappa}{1-\delta+\delta M}}.$$

For every value of M, we compute the maximum $AF_{DLTF\text{-}SVA}^{peak\,power\star 4}$ among all δ and w'_M, such that $\frac{4M+1}{6M} \leq \delta \leq 1$ and $\frac{f_{crit}}{\theta_{LTF}} < w'_M \leq F_{max}$.

Proof. This proof is similar to the proof of Lemma 10. For this case, inside Equation (8.30), we replace $\sum_{i=1}^{M} w_i^{\text{DLTF}}$ with $w'_M\left(1-\delta+\delta M\right)$, we replace $\frac{w'_1}{w'_M}$ with δ from Equation (8.21), and we replace $\frac{1-\delta+\delta\cdot M}{\left(1-\delta+\delta\cdot\sqrt[\gamma]{M}\right)^{\gamma}}$ with $U\left(\delta\right)$ from Equation (8.20). From Lemma 1 we have that $w'_M = \max\left\{\frac{w_M^{\text{DLTF}}}{\theta_{\text{LTF}}}, \frac{\sum_{i=1}^{M} w_i^*}{M}\right\}$, and from Lemma 4 we have that $\delta = \frac{\sum_{i=1}^{M-1} w'_i}{w'_M(M-1)} \geq \frac{4M+1}{6M}$. Moreover, as shown in the proof of Lemma 10, for this case it holds that $M \leq 2\frac{\sum_{i=1}^{M} w_i^{\text{DLTF}}}{w_M^{\text{DLTF}}} - 1$ for all $M \geq 1$, such that $M^{\neq 0}$ is set to M, which is the worst case for $M^{\neq 0}$. Finally, also as shown in the proof of Lemma 10, it holds that $w'_M = \frac{w_M^{\text{DLTF}}}{\theta_{\text{LTF}}}$ is the worst case for the relation between w'_M and w_M^{DLTF}. Thus, the lemma is proven. □

Theorem 5 *The approximation factor of DLTF-SVA for peak power reduction, against the optimal peak power consumption for the optimal solution, is expressed as*

$$AF_{DLTF\text{-}SVA}^{peak\,power} \leq \max\left\{AF_{DLTF\text{-}SVA}^{peak\,power\star 1}, AF_{DLTF\text{-}SVA}^{peak\,power\star 2}, AF_{DLTF\text{-}SVA}^{peak\,power\star 3}, AF_{DLTF\text{-}SVA}^{peak\,power\star 4}\right\},$$

according to the definitions in Lemmas 14, 15, 16, and 17.

Proof. This comes simply from taking the maximum among all cases from Lemmas 14, 15, 16, and 17. □

Figure 8.16 shows some examples of $\text{AF}_{\text{DLTF-SVA}}^{\text{peak power}}$ for different values of M.

8.5 Comparing DLTF-SFA and DLTF-SVA

Now that we have closed-form expressions for all approximation factors, this section compares the worst-case efficiency of DLTF-SFA against the worst-case efficiency of DLTF-SVA, for an example power function modeled from simulations for a 22 nm OOO Alpha 21264 core running an *x264* application from the PARSEC benchmark suite [4], which results in power parameters $\gamma = 3$, $\alpha = 0.27\frac{\text{W}}{\text{GHz}^3}$, $\beta = 0.52\frac{\text{W}}{\text{GHz}}$, and $\kappa = 0.5\,\text{W}$

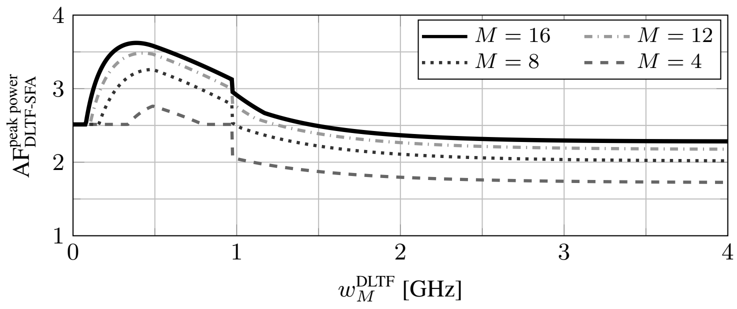


Figure 8.16: Example of the approximation factor for DLTF-SVA in terms of peak power consumption, for different values of M, when $\gamma = 3$, $\alpha = 0.27 \frac{\mathrm{W}}{\mathrm{GHz}^3}$, $\beta = 0.52 \frac{\mathrm{W}}{\mathrm{GHz}}$, and $\kappa = 0.5\,\mathrm{W}$ (i.e., for 22 nm OOO Alpha 21264 cores, as detailed in Chapter 3.3).

(as detailed in Chapter 3.3). Basically, this implies merging Figure 8.12 and Figure 8.15 into a single figure, specifically, into Figure 8.17, as well as merging Figure 8.13 and Figure 8.16 into another figure, specifically, into Figure 8.18.

With respect to peak power reduction, Figure 8.18 shows that DLTF-SVA will always outperform DLTF-SFA. This is an expected result, since under SVA the cores are executed at slower DVFS levels than under SFA in order to reduce the peak power consumption. In regards to energy minimization, Figure 8.17 shows that in the worst cases, DLTF-SFA can outperform DLTF-SVA when we assume *negligible overhead for sleeping*, while DLTF-SVA can outperform DLTF-SFA when we assume *non-negligible overhead for sleeping* (i.e., for the practical scenario in real systems). In practical terms, this means that if DLTF-SFA manages to sleep efficiently, it will save more energy than DLTF-SVA at the cost of having a higher peak power consumption. However, in case DLTF-SFA fails to sleep efficiently (i.e., if the cores mostly remain idle because there is not enough time for them to enter the sleep mode and return to execution mode), then DLTF-SVA will save more energy than DLTF-SFA in the worst-cases.

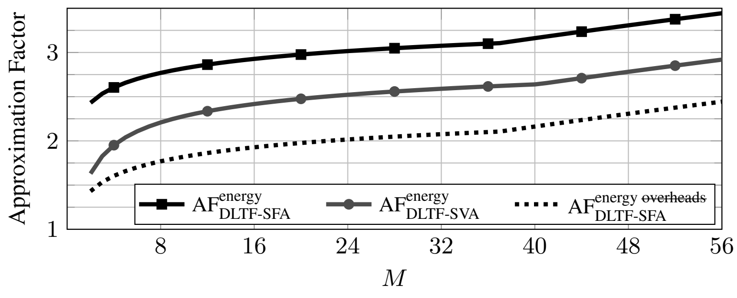


Figure 8.17: Comparison of the approximation factors for DLTF-SFA and DLTF-SVA in terms of energy consumption when $\gamma = 3$, $\alpha = 0.27 \frac{\mathrm{W}}{\mathrm{GHz}^3}$, $\beta = 0.52 \frac{\mathrm{W}}{\mathrm{GHz}}$, and $\kappa = 0.5\,\mathrm{W}$ (i.e., for 22 nm OOO Alpha 21264 cores, as detailed in Chapter 3.3).

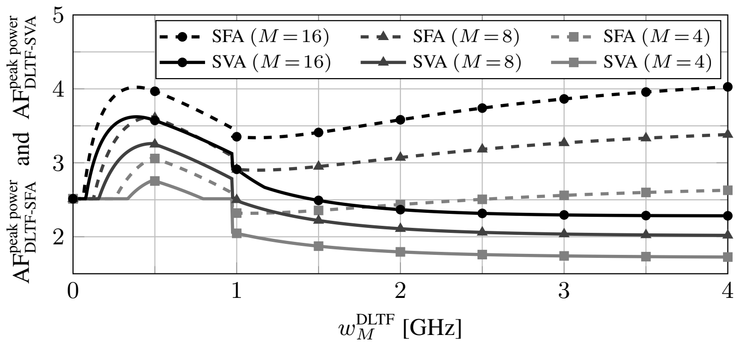


Figure 8.18: Comparison of the approximation factors for DLTF-SFA and DLTF-SVA in terms of peak power consumption, for different values of M, when $\gamma = 3$, $\alpha = 0.27 \frac{\mathrm{W}}{\mathrm{GHz}^3}$, $\beta = 0.52 \frac{\mathrm{W}}{\mathrm{GHz}}$, and $\kappa = 0.5\,\mathrm{W}$ (i.e., for 22 nm OOO Alpha 21264 cores, as detailed in Chapter 3.3).

8.6 Discrete Voltage and Frequency Pairs

This section extends the previous analysis to consider practical systems with discrete DVFS levels, by considering discrete voltage and frequency pairs $\left\{F^{\text{type}}_{q,0}, F^{\text{type}}_{q,1}, F^{\text{type}}_{q,2}, \ldots, F^{\text{type}}_{q,\hat{F}^{\text{type}}_q}\right\}$ for core type q, as defined in

Chapter 3.2. Since in this entire chapter we focus on homogeneous systems and thus there is only one possible type of core q, for simplicity of presentation, we ignore parameter q such that the available discrete voltage and frequency pairs are denoted as $\left\{F_0, F_1, F_2, \ldots, F_{\hat{F}}\right\}$, where there are in total $\hat{F}$ available frequencies for execution, frequency F_1 is the slowest available frequency such that $F_1 = F_{\text{min}}$, $F_{\hat{F}}$ is the highest available frequency such that $F_{\hat{F}} = F_{\text{max}}$, and an inactive core (i.e., idle or in a low-power mode) is said to be set at frequency $F_0 = 0$. Given the convexity of $P_{\text{core}}(f)$, and the convexity of $\frac{P_{\text{core}}(f)}{f}$ for any $f \geq f_{\text{crit}}$, the lower bounds that consider continuous frequencies, i.e., $E_{\downarrow}^{*}$ and $\hat{P}_{\downarrow}^{*}$, are also lower bounds for the optimal energy and peak power consumption when considering discrete frequencies.

Particularly for DLTF-SFA, in order to meet the timing constraints of all tasks, the discrete execution frequency for SFA, denoted as F_i, will simply be chosen among all the available frequencies such that $F_{i-1} < w_M^{\text{DLTF}} \leq F_i$. Therefore, the peak power consumption of DLTF-SFA by running at frequency F_i is equal to that of DLTF-SFA at frequency w_M^{DLTF} multiplied with $\frac{P_{\text{core}}(F_i)}{P_{\text{core}}\left(w_M^{\text{DLTF}}\right)}$. Similarly, the energy consumption of DLTF-SFA for execution by running at frequency F_i is equal to that of DLTF-SFA at frequency w_M^{DLTF} multiplied with $\frac{P_{\text{core}}(F_i) \cdot w_M^{\text{DLTF}}}{P_{\text{core}}\left(w_M^{\text{DLTF}}\right) \cdot F_i}$. For a given architecture, due to the convexity of $P_{\text{core}}(f)$ and of $\frac{P_{\text{core}}(f)}{f}$ for any $f \geq f_{\text{crit}}$, the values of the ratios $\frac{P_{\text{core}}(F_i)}{P_{\text{core}}\left(w_M^{\text{DLTF}}\right)}$ and $\frac{P_{\text{core}}(F_i) \cdot w_M^{\text{DLTF}}}{P_{\text{core}}\left(w_M^{\text{DLTF}}\right) \cdot F_i}$ will be larger as w_M^{DLTF} comes closer to frequency F_{i-1}, i.e., the ratios become larger when w_M^{DLTF} is farther away from F_i. Furthermore, assuming that frequency F_h is lowest available frequency higher than f_{crit}, i.e., it holds that $F_{h-1} < f_{\text{crit}} \leq F_h$, given that SFA will attempt to run at frequency f_{crit} in case that $w_M^{\text{DLTF}} \leq f_{\text{crit}}$, the values of the ratios when $w_M^{\text{DLTF}} \leq f_{\text{crit}}$ are $\frac{P_{\text{core}}(F_h)}{P_{\text{core}}(f_{\text{crit}})}$ and $\frac{P_{\text{core}}(F_h) \cdot f_{\text{crit}}}{P_{\text{core}}(f_{\text{crit}}) \cdot F_h}$. In this way, the approximation factors of DLTF-SFA for discrete frequencies are equal to $\rho_{\text{SFA}}^{\text{energy}} \cdot \text{AF}_{\text{DLTF-SFA}}^{\text{energy}}$ and $\rho_{\text{SFA}}^{\text{peak power}} \cdot \text{AF}_{\text{DLTF-SFA}}^{\text{peak power}}$, where the values of $\rho_{\text{SFA}}^{\text{energy}}$ and $\rho_{\text{SFA}}^{\text{peak power}}$ depend on the hardware parameters and the available frequencies, and they are computed as shown in Equation (8.31) and Equation (8.32). For example, for a system with $\gamma = 3$, $\alpha = 0.27 \frac{\text{W}}{\text{GHz}^3}$, $\beta = 0.52 \frac{\text{W}}{\text{GHz}}$, $\kappa = 0.5\,\text{W}$, and available frequencies $\{0.1\,\text{GHz}, 0.2\,\text{GHz}, \ldots, 4.0\,\text{GHz}\}$, the value of $\rho_{\text{SFA}}^{\text{energy}}$ is equal to 1.054, while the value of $\rho_{\text{SFA}}^{\text{peak power}}$ is equal to 1.113.

$$\rho_{\text{SFA}}^{\text{energy}} = \max\left\{\frac{P_{\text{core}}(F_h) \cdot f_{\text{crit}}}{P_{\text{core}}(f_{\text{crit}}) \cdot F_h}, \max_{h<i\leq\hat{F}} \frac{P_{\text{core}}(F_i) \cdot F_{i-1}}{P_{\text{core}}(F_{i-1}) \cdot F_i}\right\} \tag{8.31}$$

$$\rho_{\text{SFA}}^{\text{peak power}} = \max\left\{\frac{P_{\text{core}}(F_h)}{P_{\text{core}}(f_{\text{crit}})}, \max_{h<i\leq\hat{F}} \frac{P_{\text{core}}(F_i)}{P_{\text{core}}(F_{i-1})}\right\} \tag{8.32}$$

Similarly, for SVA, when the system has discrete DVFS levels, the voltage of the cluster is set according to F_m (i.e., to the minimum voltage at which frequency F_m can be stably achieved) such that $F_{m-1} < w_M^{\text{DLTF}} \leq F_m$, while the frequency of core i is set to F_j such that $F_{j-1} < w_i^{\text{DLTF}} \leq F_j$. Given that now we consider that the cores execute at frequencies slightly higher than their cycle utilization, it no longer holds that all cores are always busy (even if all the tasks require their worst-case execution times in order to finish every task instance), an thus some cores will be kept idle during a short time. Therefore, by keeping the cores idle when they finish all the workload in their ready queues (i.e., clock-gated, which is the worst case, as cores are not put into a low-power mode even if the idle time is longer than the break-even time), the worst-case energy consumption and peak power consumption ratio for SVA when we consider discrete DVFS levels against the continuous cases, defined as ρ_{SVA}, can be expressed as seen in Equation (8.33).

$$\rho_{\text{SVA}} = \max\left\{\frac{\alpha \cdot {F_m}^{\gamma-1} \cdot \sum_{i=1}^{M} w_i^{\text{DLTF}} + M^{\neq 0}\left(\beta \cdot F_m + \kappa\right)}{\alpha \cdot {w_M^{\text{DLTF}}}^{\gamma-1} \cdot \sum_{i=1}^{M} w_i^{\text{DLTF}} + M^{\neq 0}\left(\beta \cdot w_M^{\text{DLTF}} + \kappa\right)}\right\} \tag{8.33}$$

There are two extreme cases for ρ_{SVA}: (1) the total cycle utilization is really small, such that the leakage and independent energy/power consumptions dominate; or (2) the total cycle utilization is high, for which the highest total cycle utilization is $M^{\neq 0} \cdot w_M^{\text{DLTF}}$. Moreover, the worst case for ρ_{SVA} happens when $w_M^{\text{DLTF}} \to F_{m-1}$ and $w_i^{\text{DLTF}} \to F_{j-1}$ for all $i = 1, 2, \ldots, M-1$, as these are the cases in which the cycle utilization of each core is further away from the next feasible frequency. Therefore, we have that ρ_{SVA} can be expressed as seen in Equation (8.34).

$$\rho_{\text{SVA}} = \max_{1<i\leq\hat{F}}\left\{\max\left\{\frac{\beta \cdot F_i + \kappa}{\beta \cdot F_{i-1} + \kappa}, \frac{\alpha \cdot {F_i}^{\gamma-1} \cdot F_{i-1} + \beta \cdot F_i + \kappa}{\alpha \cdot {F_{i-1}}^{\gamma} + \beta \cdot F_{i-1} + \kappa}\right\}\right\} \tag{8.34}$$

Finally, the approximation factors become $\rho_{\text{SVA}} \cdot \text{AF}^{\text{energy}}_{\text{DLTF-SVA}}$ and $\rho_{\text{SVA}} \cdot \text{AF}^{\text{peak power}}_{\text{DLTF-SVA}}$, where the value of ρ_{SVA} depends on the hardware parameters and the available frequencies. For example, for a system with $\gamma = 3$, $\alpha = 0.27 \frac{\text{W}}{\text{GHz}^3}$, $\beta = 0.52 \frac{\text{W}}{\text{GHz}}$, $\kappa = 0.5\,\text{W}$, and available frequencies $\{0.1\,\text{GHz}, 0.2\,\text{GHz}, \ldots, 4.0\,\text{GHz}\}$, the value of ρ_{SVA} is no more than 1.096.

8.7 Experimental Evaluations

This section presents experimental evaluations conducted with gem5 [5] and McPAT [57]. We compare the power and energy efficiency of DLTF-SFA and DLTF-SVA against the peak power and energy lower bounds, as well as against each other.

8.7.1 Setup

For our experimental evaluations, we use the simulation framework described in Chapter 4 in high-level mode. We run our simulations for a single cluster, for which we consider two different cases for the number of cores in the cluster, specifically, 8 cores and 16 cores. We consider 22 nm OOO Alpha 21264 cores (as described in Chapter 4.2.1), with available frequencies $\{0.1\,\text{GHz}, 0.2\,\text{GHz}, \ldots, 4.0\,\text{GHz}\}$.

For benchmarks, we consider two representative applications from the PARSEC benchmark suite [4] described in Chapter 4.3, specifically, *x264* and *bodytrack*, executed as a single independent thread. Hence, a task is an instance of one of these two applications, and by selecting different deadlines/periods we can consider different cycle utilizations for each task. Moreover, we test $5 \cdot 10^6$ different combinations of tasks, with a random number of tasks, random periods and random cycle utilizations, as well as tasks that result in cycle utilization distributions according to the analyzed worst cases.

For each experiment, the tasks are partitioning with DLTF, and they are scheduled in each individual core according to EDF. Under SVA, the execution frequency of each core is chosen as the closest available frequency that is higher than or equal to the required cycle utilization in the core. Under SFA, the execution frequency for all cores in the cluster is chosen as the closest available frequency that is higher than or equal to the required cycle utilization of the highest loaded core in the cluster. The voltage of the cluster is chosen as the minimum voltage for stable execution in regards to the core running at the highest frequency inside the cluster. As discussed in Section 8.6, since cores execute at frequencies higher than their cycle utilization, cores will be idle during short time intervals under SVA, and during longer time intervals for SFA. We assume that the time overhead of a core for entering the sleep mode and returning to execution mode is 100 ms. Under SVA, when a core has no more workload to execute on its ready queue, the core can be simply kept idle, i.e., clock-gated (note that there is no restriction to further combine SVA with a DPM technique, and we only assume that cores are kept idle in order to account for the worst cases). Under SFA, if the idle time in a core is less than 100 ms, then the core is kept idle (i.e., clock-gated, and thus consuming idle power); and if it is larger than 100 ms, then the core is put to sleep (taking it 100 ms to enter the sleep mode and returning to execution mode), consuming idle power only during 100 ms.

In order to evaluate our previous analysis in Section 8.3 and Section 8.4, we also need to compare against the peak power and energy lower bounds, for which the power values from McPAT can be modeled by power parameters $\gamma = 3$, $\alpha = 0.27 \frac{\text{W}}{\text{GHz}^3}$, $\beta = 0.52 \frac{\text{W}}{\text{GHz}}$, and $\kappa = 0.5\,\text{W}$ (as detailed in Chapter 3.3). Furthermore, when computing the lower bounds, we use the Newton-Raphson method [7] to solve Equation (8.16) with 200 iterations, since this is possible for concrete cases and it reduces the pessimism of our experimental evaluations.

8.7.2 Results

Figure 8.19 and Figure 8.20 present the results for energy minimization and peak power reduction, respectively, for both DLTF-SFA and DLTF-SVA against lower bounds $E^*_{\downarrow}$ and $\hat{P}^*_{\downarrow}$, for $M = 8$ and $M = 16$. The horizontal axis represents the cycle utilization of the highest loaded core after partitioning tasks with DLTF (i.e., w_M^{DLTF}). Among all the tested cases for each frequency, we show the maximum experimental ratio between DLTF-SFA or DLTF-SVA and the corresponding lower bound. The *saw-tooth* shapes observed in the figures occur due to the regrouping done by DLTF, which reduces the resulting number of active cores with cycle utilizations larger than 0 whenever possible (i.e., $M^{\neq 0}$).

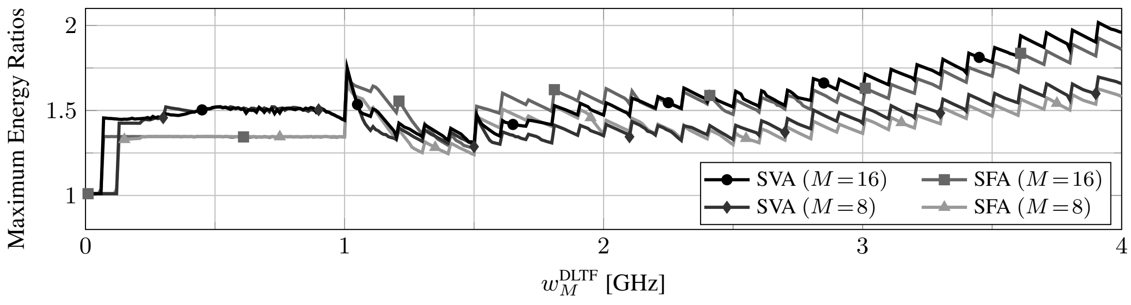


Figure 8.19: Experimental results of energy minimization for DLTF-SFA and DLTF-SVA, against the energy lower bound.

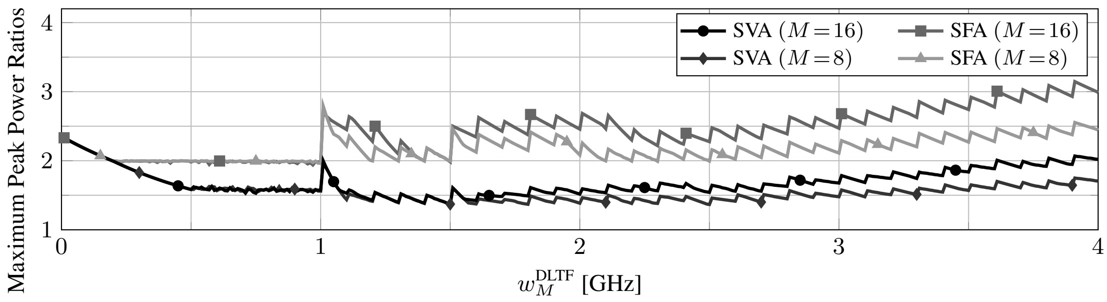


Figure 8.20: Experimental results of peak power reduction for DLTF-SFA and DLTF-SVA, against the peak power lower bound.

Furthermore, Figure 8.21 and Figure 8.22 directly compare DLTF-SFA to DLTF-SVA. Both figures show the maximum and average experimental ratios between DLTF-SFA and DLTF-SVA, and vice-versa. In Figure 8.19, for the same number of cores, we can observe that DLTF-SFA generally behaves better than DLTF-SVA for the worst cases in regards to energy minimization. However, there are several cases in which DLTF-SVA can be more efficient. As explained in Section 8.5, this depends on how efficiently DLTF-SFA manages to bring cores into sleep mode. For the cases in which DLTF-SFA fails to sleep efficiently, DLTF-SVA will save more energy. This can also be observed in Figure 8.21. Contrarily, Figure 8.20 and Figure 8.22 show that DLTF-SVA always consumes less peak power than DLTF-SFA, both in average and for the worst cases.

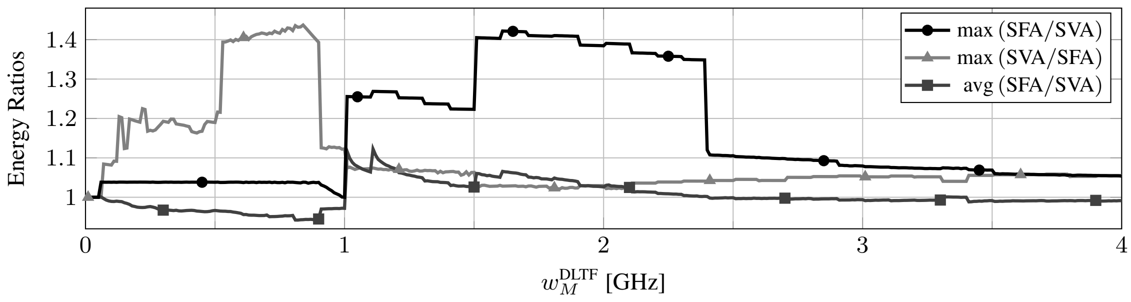


Figure 8.21: Experimental results of energy minimization comparing maximum and average ratios for DLTF-SFA against DLTF-SVA, for an island with $M = 16$ cores.

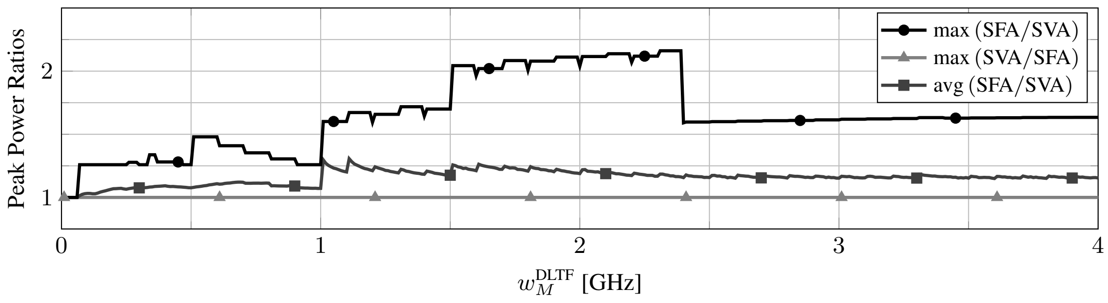


Figure 8.22: Experimental results of peak power reduction comparing maximum and average ratios for DLTF-SFA against DLTF-SVA, for an island with $M=16$ cores.

8.8 Summary

For performance-constrained applications or real-time tasks that are already assigned to a specific cluster (or for systems with a global supply voltage), in this chapter we have presented the polynomial-time Double Largest Task First (DLTF) strategy for partitioning tasks to cores based on load balancing and idle energy reduction, and the linear-time Single Frequency Approximation (SFA) and Single Voltage Approximation (SVA) schemes for deciding the DVFS schedule for execution. Most importantly, we have provided comprehensive theoretical analysis for the worst-case behavior (in terms of energy and peak power efficiency) of combining DLTF with either SFA or SVA, denoted as DLTF-SFA and DLTF-SVA. The analysis and the experimental evaluations show that the efficiency for energy minimization of DLTF-SVA outperforms that of DLTF-SFA when the latter fails to efficiently bring cores to low-power modes. However, when DLTF-SFA manages to sleep efficiently, DLTF-SVA results in higher energy consumptions. In regards to peak power reduction, DLTF-SVA always consumes less peak power than DLTF-SFA. This occurs because the leakage and independent power consumption are equivalent for both DLTF-SFA and DLTF-SVA in most cases (the peak power consumption of DLTF-SFA is larger when we have low cycle utilizations), but the dynamic power consumption of DLTF-SFA can be much larger than that of DLTF-SVA, mainly due to running cores at higher frequencies. Therefore, DLTF-SVA is much more efficient than DLTF-SFA for peak power reduction in all cases, both average and corner cases. Because of this reason, DLTF-SVA can potentially satisfy the power budget under consideration for many more cases than DLTF-SFA can, also potentially resulting in lower temperatures throughout the chip. Furthermore, DLTF-SVA manages to accomplish this without unnecessary sacrifices in terms of energy consumption.

Finally, assuming architectures like the Exynos 5 Octa (5422) processor [92] (based on ARM's big.LITTLE architecture [21]), in which all cores in a cluster share a common voltage and common frequency, SFA is used as the basic DVFS scheduling policy for the task-to-core assignment techniques for systems with multiple clusters/voltage islands, presented in Chapter 9 and Chapter 10. Moreover, Chapter 9 also extends the theoretical analysis for SFA presented in this chapter in order to consider systems with multiple clusters/voltage islands.

Chapter 9

Energy-Efficient Task-to-core Assignment for Homogeneous Clustered Manycores

9.1 Overview

This chapter focuses on energy-efficient task-to-core assignment/mapping for homogeneous multicore systems clustered in multiple voltage islands, where due to the restriction that all the cores in a cluster have to operate at the same DVFS levels at any point in time, different assignments will result in different energy consumptions. In order for the tasks to meet their timing constraints, a particular task set (i.e., a group of tasks assigned together on one core) has to be executed at certain DVFS levels. When several task sets are mapped onto the same cluster, one task set might dominate the required DVFS levels and thus significantly increase the energy consumption of the other task sets in the cluster. Furthermore, mapping tasks to a cluster enforces the activation of the cluster, which consumes its own energy (aside from the energy consumption of the cores). As a result, in a clustered architecture with multiple voltage islands where the objective is to minimize the overall energy consumption, the assignment/mapping of task sets to clusters and the corresponding cluster activations have to be done carefully, and adopting existing system-level power management schemes that do not consider the characteristics of such architectures may not work as well as expected.

Therefore, by using the simple Single Frequency Approximation (SFA) scheme (presented in detail in Chapter 8) to decide the DVFS levels on individual clusters, in this chapter we explore how to map given task sets onto cores belonging to different clusters in order to minimize the overall energy consumption [72]. Specifically, we present some simple and intuitive polynomial-time heuristics to assign given task sets onto cores in different clusters, and we analyze the approximation factor of *any mapping heuristic* against the optimal solution that results in the minimum energy consumption. Furthermore, also for given task sets, we develop a dynamic programming algorithm, called Dynamic Voltage Island Assignment (DYVIA), that derives optimal mapping solutions for minimizing the energy consumption when using SFA in individual clusters. Our DYVIA algorithm has polynomial-time complexity when either the number of clusters or the number of cores in each cluster are constant. Moreover, we also provide analysis for comparing our DYVIA algorithm against the optimal mapping solution with the ideal DVFS schedule for every cluster. We experimentally evaluate the execution time and energy efficiency of some mapping heuristics and our dynamic programming solution on Intel's Single Chip Cloud computer (SCC) [36]. Even though in order to derive optimal solutions our DYVIA algorithm has higher time complexity than the evaluated heuristics, the evaluations show that the execution time of DYVIA is just a few milliseconds for practical settings that consider up to 6 clusters and 8 cores per cluster, e.g., SCC. Finally, we conduct simulations for hypothetical platforms with several combinations for the number of clusters and the number of cores per cluster, as well as richer DVFS and DPM features than SCC, and different policies for the partitioning of tasks into task sets.

9.1.1 Problem Definition

In this chapter, we focus on a homogeneous multicore system with M cores clustered in multiple voltage islands, with a total of V clusters defined as $\{I_1, I_2, \ldots, I_V\}$, where all clusters have an equal number of cores per cluster K (i.e., according to the notation in Chapter 3.2, this means that in this chapter it holds that $K = M_k^{\text{cluster}} = M_{k-1}^{\text{cluster}}$ for all $k = 2, \ldots, V$), such that $M = K \cdot V$. We consider that any cluster consumes negligible power when in the *inactive* state, and that it consumes η amount of power when in the *active* state (since there is no voltage regulator with 100% efficiency). After the assignment of task sets to cores, we consider a cluster to be *inactive* if all the task sets assigned to the cluster have zero cycle utilization (i.e., no core in the cluster is assigned any workload), and we consider it to be *active* otherwise. Without loss of generality, the clusters are increasingly ordered with respect to their DVFS levels, such that I_1 is the cluster with the lowest DVFS levels and I_V is the cluster with the highest DVFS levels.

There are R periodic performance-constrained/real-time tasks that have to be executed on the chip, which we assume to be partitioned according to some task partitioning strategy (e.g., LTF or DLTF, described in Chapter 8) into $M^\star$ sets of tasks (or tasks sets), where each task set can hold one or more tasks, such that all tasks belonging to the same task set will be mapped and executed on a single core by using preemption. Clearly, since there are M cores in the system, in order to provide a feasible mapping, it should hold that $M^\star \leq M$, as otherwise there would be more task sets to map than cores on the chip. Without loss of generality and for simplicity of presentation, after the task partitioning stage, we create $M - M^\star$ dummy (empty) task sets with zero cycle utilization, resulting in task sets $\{\mathbf{S}_1, \mathbf{S}_2, \ldots, \mathbf{S}_M\}$, with corresponding cycle utilizations $w_1 \leq w_2 \leq \cdots \leq w_M$, where (compared to the notation from Chapter 3.1) we have omitted parameter q as in this chapter we focus on homogeneous systems. In case a task leaves the system or a new task arrives, then a new task partition should be obtained, and the algorithms later presented in Section 9.2 and Section 9.4 should be re-executed.

Some additional notations are necessary in order to identify the task sets assigned to each cluster. We define set $\mathbf{L}_j = \{\ell_{j,1}, \ell_{j,2}, \ldots, \ell_{j,K}\}$ as the indexes of the task sets assigned to cluster I_j such that $\ell_{j,1} < \ell_{j,2} < \cdots < \ell_{j,K}$ for all $j = 1, 2, \ldots, V$. Moreover, for every $\ell_{j,i}$ it should hold that $1 \leq \ell_{j,i} \leq M$, and that $\ell_{j,i}$ is unique for all j, i. Given that the task sets are ordered with respect to their cycle utilizations, it holds that $w_{\ell_{j,1}} \leq w_{\ell_{j,2}} \leq \cdots \leq w_{\ell_{j,K}}$. For example, in case that $K = 3$ and task sets $\mathbf{S}_5$, $\mathbf{S}_8$, and $\mathbf{S}_9$ are assigned to cluster I_4, then $\mathbf{L}_4$ is set to $\{5, 8, 9\}$, i.e., $\ell_{4,1} = 5$, $\ell_{4,2} = 8$, and $\ell_{4,3} = 9$.

After the task set assignment is finished and all task sets have been assigned onto cores in different clusters, a DVFS policy has to be adopted in order to decide the DVFS levels for the individual clusters. Here, we consider the simple and intuitive SFA scheme (presented in Chapter 8), where all the tasks assigned to cluster I_j are executed at single frequency f_j during the entire hyper-period, such that $w_{\ell_{j,K}} \leq f_j$ so all the tasks assigned to cluster I_j meet their timing constraints. Particularly, cluster I_j will be executed at frequency $f_j = \max\left\{f_{\text{crit}}, w_{\ell_{j,K}}\right\}$ during the entire hyper-period. As done everywhere else in this dissertation, the supply voltage of cluster I_j is set to the lowest available value at which all cores in the cluster can stably execute at frequency f_j.

We consider a general power consumption model for individual cores in which the average power consumption of a core executing a certain task at frequency f is defined as $P_{\text{core}}(f)$. Namely, compared to the notation presented in Chapter 3.3 in which the power consumed on a core of type q while executing task τ_n at frequency index j was denoted as $P_q^{\tau_n}\left(F_{q,j}^{\text{type}}\right)$, here we omit parameter q since we focus on homogeneous systems, and we assume that all tasks consume the same average power when executing at the same frequency. For the algorithms and properties derived in Sections 9.2, 9.3, and 9.4, we assume continuous frequencies in the range of $[F_{\text{min}}, F_{\text{max}}]$. However, all algorithms still work for systems with discrete frequencies, as shown in the experiments of Section 9.5 and the simulations of Section 9.6.

Similar to Chapter 8, when a core finishes executing all the workload available in its ready queue, we consider that it enters a low-power mode (e.g., sleep, power-gated, etc.) that consumes $\kappa^{\text{sleep}} \geq 0$ power. In this way, when a cluster is active, the minimum power consumption of any core in the cluster is κ^{sleep}. This implies that an active cluster not only consumes η power for being active, but there is also an offset of $K \cdot \kappa^{\text{sleep}}$ power consumed by the cores in the cluster. Therefore, given that we can transfer this power consumption $K \cdot \kappa^{\text{sleep}}$ to the value of power consumption η, without loss of generality, we can set η to $\eta + K \cdot \kappa^{\text{sleep}}$, and set $P_{\text{core}}(f)$ to $P_{\text{core}}(f) - \kappa^{\text{sleep}}$, such that we can disregard the effect of the power consumption of a core in a low-power mode, because it is already considered inside the new value of η. For simplicity of presentation, the overheads of entering/leaving a low-power mode (i.e., the *overheads for sleeping*) are implicitly considered negligible.

In order to analyze the quality of the proposed algorithms in terms of energy minimization, we assume that

$P_{\text{core}}(f)$ is a convex and increasing function with respect to f, and we assume that $\frac{P_{\text{core}}(f)}{f}$ is a non-decreasing function with respect to f for any $f \geq f_{\text{crit}}$. *All the theorems presented in this chapter are only based on these assumptions for $P_{core}(f)$ and $\frac{P_{core}(f)}{f}$, and therefore apply for many power models, making the derived algorithms and obtained results very general. Specific power functions are only used in this chapter when numerical results are required.* Furthermore, as already discussed in Chapter 3.3 and Chapter 3.4, these assumptions about $P_{\text{core}}(f)$ and $\frac{P_{\text{core}}(f)}{f}$ comply with most of the power models for CMOS processors adopted in the literature, e.g, [3,12,14,15,108,109], where the most widely used power consumption function, already shown in Equation (3.3) in Chapter 3.3, is $P_{\text{core}}(f) = \alpha \cdot f^{\gamma} + \beta \cdot f + \kappa$ (with $\alpha > 0$, $\gamma > 1$, $\beta \geq 0$, and $\kappa \geq 0$) and $f_{\text{crit}} = \sqrt[\gamma]{\frac{\kappa}{(\gamma-1)\alpha}}$.

For the above models, a core belonging to cluster I_j where we execute task set $\mathbf{S}_i$ will have an energy consumption during a hyper-period of $D \cdot P_{\text{core}}(f_j) \cdot \frac{w_i}{f_j}$. Therefore, considering all K cores inside the cluster and the power consumption η for keeping the cluster active, the energy consumption of cluster I_j during a hyper-period is computed as shown in Equation (9.1).

$$E_j = \begin{cases} 0 & \text{if } \sum_{i=1}^{K} w_{\ell_{j,i}} = 0 \\ D\left(\eta + \frac{P_{\text{core}}(f_j)}{f_j} \sum\limits_{i=1}^{K} w_{\ell_{j,i}}\right) & \text{otherwise} \end{cases} \tag{9.1}$$

For a fixed hardware platform with V clusters and K cores per cluster (i.e., V and K are constants, e.g., Intel's SCC), the *multiple voltage islands assignment problem* focuses on assigning/mapping periodic real-time task (or performance-constrained applications) onto cores and clusters, such that the overall energy consumption among all clusters is minimized, i.e., such that $\sum_{j=1}^{V} E_j$ is minimized according to Equation (9.1), by using SFA on individual clusters, and by assuming a given task partition $\{\mathbf{S}_1, \mathbf{S}_2, \ldots, \mathbf{S}_M\}$ as an input (i.e., the tasks are already partitioned into task sets using some task partitioning strategy, e.g., LTF of DLTF).

Finally, in regards to considering a given task partition as an input, for Sections 9.2, 9.3 and 9.4, we will consider the mapping of $M^{\star}$ already partitioned task sets together with the corresponding $M - M^{\star}$ dummy task sets, i.e., $\{\mathbf{S}_1, \mathbf{S}_2, \ldots, \mathbf{S}_M\}$. Nevertheless, in order to show the effects of task partitioning combined with the task sets assignment algorithms presented in Section 9.2 and Section 9.4, we conduct simulations for different task partitioning policies of LTF in Section 9.6 (i.e., we consider several values for $M^{\star}$ and derive several task partitions with LTF). As mentioned above, it should hold that $M^{\star} \leq M$, as otherwise there would be more task sets to map than cores available on the chip. However, depending on the influence of the static/leakage power in $P_{\text{core}}(f)$ and on the value of η, for some cases, using less than M cores for task partitioning can potentially result in energy savings by shutting down cores and clusters.

9.2 Simple Heuristic Algorithms

This section presents two *simple* and *very intuitive* algorithms to solve the *multiple voltage islands assignment problem* stated in Section 9.1.1, and it also describes an algorithm based on extremal optimization extended from [65]. These three algorithms have low time complexity; however, they derive non-optimal solutions. Furthermore, when using SFA in individual clusters, this section also provides theoretical analysis for the approximation factor of *any* task partition mapping heuristic against the optimal assignment.

9.2.1 Description of Simple Heuristic Algorithms

Consecutive Cores Heuristic (CCH)

One simple and very intuitive heuristic algorithm, which we call Consecutive Cores Heuristic (CCH), consist in assigning the task sets onto cores inside clusters consecutively with respect to their cycle utilization. Namely, we assign to cluster I_j all the task sets whose indexes are inside set $\mathbf{L}_j$, such that $\mathbf{L}_j = \{(j-1)K+1, (j-1)K+2, \ldots, (j-1)K+K\}$, and we do this for all clusters I_j, i.e., for all $j =$

$1, 2, \ldots, V$, resulting in linear time complexity $O(K \cdot V)$. In other words, considering that every cluster has K cores and that task sets $\{\mathbf{S}_1, \mathbf{S}_2, \ldots, \mathbf{S}_M\}$ are ordered according to their cycle utilizations $w_1 \leq w_2 \leq \cdots \leq w_M$, we assign the first K task sets to cluster I_1, the second K task sets to cluster I_2, and so on. For example, if $V = 4$ and $K = 3$, CCH will assign task sets $\{\mathbf{S}_1, \mathbf{S}_2, \mathbf{S}_3\}$ to cluster I_1, task sets $\{\mathbf{S}_4, \mathbf{S}_5, \mathbf{S}_6\}$ to cluster I_2, task sets $\{\mathbf{S}_7, \mathbf{S}_8, \mathbf{S}_9\}$ to cluster I_3, and task sets $\{\mathbf{S}_{10}, \mathbf{S}_{11}, \mathbf{S}_{12}\}$ to cluster I_4, as illustrated in the example in Figure 9.1.

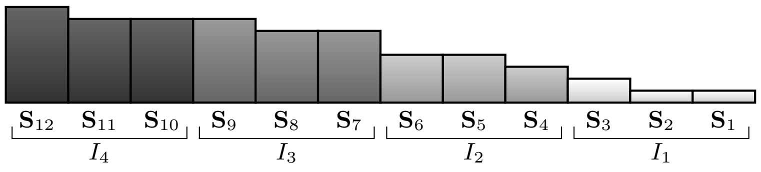


Figure 9.1: Example of CCH for $V = 4$ and $K = 3$. The high of the bars represents the cycle utilization of the task sets, and the colors represent their cluster assignment.

Balanced Utilization Heuristic (BUH)

Another simple heuristic algorithm, which we call Balanced Utilization Heuristic (BUH), consist in assigning K consecutive task sets onto cores in cluster I_j, such that the difference between the lowest and highest utilization task sets in the cluster is minimized. Namely, if there are still more than K task sets to be assigned, BUH finds index h such that $w_{h+K-1} - w_h$ is minimized. Then, these K task sets are mapped onto one cluster. Following, we can find a feasible mapping by removing these task sets from the problem instance, relabeling the remaining task sets, and repeating the process until there are no unassigned task sets left. Given that finding index h requires $O(M + K)$ time complexity and the number of iterations is at most V, the overall time complexity of BUH is $O(K \cdot V^2)$. An example illustrating this algorithm is presented in Figure 9.2, where task sets $\{\mathbf{S}_9, \mathbf{S}_{10}, \mathbf{S}_{11}\}$ are first assigned to cluster I_3, then task sets $\{\mathbf{S}_1, \mathbf{S}_2, \mathbf{S}_3\}$ are assigned to cluster I_1, then task sets $\{\mathbf{S}_4, \mathbf{S}_5, \mathbf{S}_6\}$ are assigned to cluster I_2, and finally task sets $\{\mathbf{S}_7, \mathbf{S}_8, \mathbf{S}_{12}\}$ are assigned to cluster I_4.

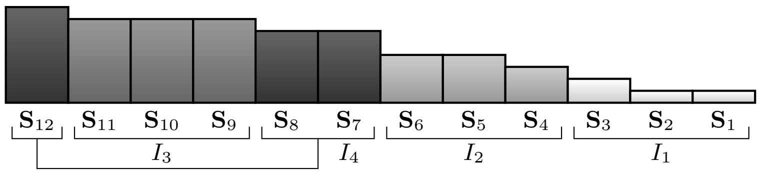


Figure 9.2: Example of BUH for $V = 4$ and $K = 3$. The high of the bars represents the cycle utilization of the task sets, and the colors represent their cluster assignment.

Extremal Optimization Heuristic (EOH)

This algorithm, called EOH and extended from [65], performs a random search in order to improve the overall energy consumption, starting from an arbitrary initial solution, e.g., the assignment derived by CCH. The algorithm is based on swapping, by selecting two task sets to swap in every iteration: an unfavorable task set and a replacement task set. By considering two fitness functions and a power-law distribution, EOH uses information about the system costs for selecting the task sets to be swapped. It is expected that such an approach results into a fast progress towards the final and improved solution. In addition, EOH accepts new solutions unconditionally, therefore potentially making the algorithm easier to tune and avoiding a local minimum. The randomness process is repeated with a predefined number of iterations until there is no apparent improvement. Among all the evaluated mappings, EOH returns the mapping that results in the lowest overall energy consumption. *Note that EOH is not a contribution of this chapter. We present it here for completeness, since we later use it for comparison in Sections 9.5 and 9.6.*

The main difference between the EOH implementation considered in this chapter and that of the algorithm presented in [65], is that here we map one task set to one core, instead of just one task per core. Moreover, in our EOH implementation we consider negligible energy consumption for the communication between tasks, such that our fitness functions are simplified. *Although EOH may help to improve the mapping solutions, in the worst cases, the quality of the solution remains as the initial solution.*

9.2.2 Approximation Factor for Simple Heuristics

This section presents the detailed proof of the approximation factor for *any* task partition mapping heuristic that uses $\left\lceil \frac{M^\star}{K} \right\rceil$ clusters with non-empty task sets and SFA to decide the DVFS levels on individual clusters, against the optimal assignment that also uses SFA on individual clusters. We denote such an approximation factor as $\mathrm{AF}_{\text{ASG=ANY}}^{\text{DVFS=SFA}}$. Namely, although some heuristics might perform well for the general cases, for a given heuristic there will exist at least one worst-case task partition for which the specific heuristic behaves poorly, particularly, resulting in the approximation factor presented below.

We start by denoting $E_{j\,\text{ASG=ANY}}^{\text{DVFS=SFA}}$ as the energy consumption of cluster I_j that uses *any* task partition mapping algorithm (hence the ASG=ANY) and that uses SFA to decide the DVFS levels of individual clusters (hence the DVFS=SFA). For example, when using heuristic CCH, the value of $E_{j\,\text{ASG=ANY}}^{\text{DVFS=SFA}}$ represents the energy consumption on cluster I_j when mapping the task sets with heuristic CCH and using SFA to decide the DVFS levels on each cluster. Given that the same holds when considering *any* other heuristic, we do not constraint the analysis using a more specific notation, e.g., $E_{j\,\text{ASG=CCH}}^{\text{DVFS=SFA}}$.

From the definition of set $\mathbf{L}_j = \{\ell_{j,1}, \ell_{j,2}, \ldots, \ell_{j,K}\}$ in Section 9.1.1, task set $\mathbf{S}_{\ell_{j,i}}$ is assigned onto the i-th core of cluster I_j. Hence, we denote $E_{j\,\text{ASG=SFA}}^{\text{DVFS=SFA}}$ as the energy consumption of task sets $\mathbf{S}_{\ell_{j,i}}$ for all $i = 1, 2, \ldots, K$ when using SFA to decide the DVFS levels on individual clusters, and when using the optimal task set assignment solution under SFA (hence, the ASG=SFA). Furthermore, let $E_{j\,\text{ASG=ANY}}^{\text{DVFS=per-core}}$ be the energy consumption of task sets $\mathbf{S}_{\ell_{j,i}}$ for all $i = 1, 2, \ldots, K$ when having per-core DVFS. Given that having per-core DVFS is the optimal solution from a DVFS perspective where the task set assignment would play no role, then clearly $E_{j\,\text{ASG=ANY}}^{\text{DVFS=per-core}}$ is the lower bound for the energy consumption. Therefore, the approximation factor $\mathrm{AF}_{\text{ASG=ANY}}^{\text{DVFS=SFA}}$ can be computed as

$$\mathrm{AF}_{\text{ASG=ANY}}^{\text{DVFS=SFA}} = \frac{\sum_{j=1}^{V} E_{j\,\text{ASG=ANY}}^{\text{DVFS=SFA}}}{\sum_{j=1}^{V} E_{j\,\text{ASG=SFA}}^{\text{DVFS=SFA}}} \leq \frac{\sum_{j=1}^{V} E_{j\,\text{ASG=ANY}}^{\text{DVFS=SFA}}}{\sum_{j=1}^{V} E_{j\,\text{ASG=ANY}}^{\text{DVFS=per-core}}} \leq \max_{j=1,2,\ldots,V} \left\{ \frac{E_{j\,\text{ASG=ANY}}^{\text{DVFS=SFA}}}{E_{j\,\text{ASG=ANY}}^{\text{DVFS=per-core}}} \right\}.$$

Throughout the proof, we implicitly assume that η is equal to 0, as the optimal solution also requires to use at least $\left\lceil \frac{M^\star}{K} \right\rceil$ clusters. Moreover, given that η is a constant that appears both in the numerator and denominator, the worst-case for the approximation factor occurs when η is equal to 0. The energy consumption function for $E_{j\,\text{ASG=ANY}}^{\text{DVFS=SFA}}$ was already presented in Equation (9.1), and the energy consumption function for $E_{j\,\text{ASG=ANY}}^{\text{DVFS=per-core}}$ (assuming $\eta = 0$) is shown in Equation (9.2), such that $f_{j,i} = \max\left\{f_{\text{crit}}, w_{\ell_{j,i}}\right\}$.

$$E_{j\,\text{ASG=ANY}}^{\text{DVFS=per-core}} = D \sum_{i=1}^{K} \frac{P_{\text{core}}\left(f_{j,i}\right)}{f_{j,i}} \cdot w_{\ell_{j,i}} \tag{9.2}$$

Therefore, from Equation (9.1) and Equation (9.2), the approximation factor can be rewritten as

$$\mathrm{AF}_{\text{ASG=ANY}}^{\text{DVFS=SFA}} \leq \max_{j=1,2,\ldots,V} \left\{ \frac{\frac{P_{\text{core}}(f_{j,K})}{f_{j,K}} \sum_{i=1}^{K} w_{\ell_{j,i}}}{\sum_{i=1}^{K} \frac{P_{\text{core}}(f_{j,i})}{f_{j,i}} \cdot w_{\ell_{j,i}}} \right\} \leq \max_{j=1,2,\ldots,V} \left\{ \frac{1 + \sum_{i=1}^{K-1} \frac{w_{\ell_{j,i}}}{w_{\ell_{j,K}}}}{1 + \sum_{i=1}^{K-1} \frac{f_{j,K}}{P_{\text{core}}(f_{j,K})} \cdot \frac{P_{\text{core}}(f_{j,i})}{f_{j,i}} \cdot \frac{w_{\ell_{j,i}}}{w_{\ell_{j,K}}}} \right\}.$$

Regardless of the absolute value of the cycle utilizations and the cluster index j, the worst case for the above relation only depends on the ratios of task sets $\frac{w_{\ell_{j,i}}}{w_{\ell_{j,K}}}$ for every cluster. The maximal value is found when $\sum_{i=1}^{K-1} \frac{w_{\ell_{j,i}}}{w_{\ell_{j,K}}}$ remains constant and when $\sum_{i=1}^{K-1} \frac{f_{j,K}}{P_{\text{core}}(f_{j,K})} \cdot \frac{P_{\text{core}}(f_{j,i})}{f_{j,i}} \cdot \frac{w_{\ell_{j,i}}}{w_{\ell_{j,K}}}$ is minimized. Given that the latter is a convex and increasing function with respect to $\frac{w_{\ell_{j,i}}}{w_{\ell_{j,K}}}$, we know that it is minimized when $w_{\ell_{j,i}} = \frac{1}{K-1} \sum_{i=1}^{K-1} w_{\ell_{j,i}}$ for all $j = 1, 2, \ldots, V$. In other words, the maximum value of $\mathrm{AF}_{\text{ASG=ANY}}^{\text{DVFS=SFA}}$ occurs when the $K-1$ least loaded task sets in every cluster have the same cycle utilization. Therefore, by defining $x = \frac{1}{K-1} \sum_{i=1}^{K-1} \frac{w_{\ell_{j,i}}}{w_{\ell_{j,K}}}$ as the average cycle utilization of the $K-1$ least loaded task sets, the approximation factor is computed as shown in Equation (9.3), where $0 \leq x \leq 1$ and $f_x = \max\left\{f_{\text{crit}}, x \cdot w_{\ell_{j,K}}\right\}$.

$$\mathrm{AF}_{\text{ASG=ANY}}^{\text{DVFS=SFA}} \leq \max_{0 \leq x \leq 1} \frac{1 + (K-1)\,x}{1 + (K-1) \frac{f_{j,K}}{P_{\text{core}}(f_{j,K})} \cdot \frac{P_{\text{core}}(f_x)}{f_x} \cdot x} \tag{9.3}$$

The specific value of x that maximizes Equation (9.3) will depend on the power consumption model adopted for the cores.

9.2.3 Numerical Examples for the Approximation Factor of Simple Heuristics

Given that Equation (9.3) presents the approximation factor for *any* task partition mapping heuristic when using SFA to decide the DVFS levels on individual clusters, it also represents the approximation factor for using CCH and BUH for mapping task sets to cores and clusters under SFA. In order to illustrate the worst-case behavior derived in Equation (9.3), we provide numerical results based on a specific power consumption function, particularly, $P_{\text{core}}(f) = \alpha \cdot f^3$. For such a case, Equation (9.3) can be rewritten as shown in Equation (9.4).

$$\text{AF}^{\text{DVFS=SFA}}_{\text{ASG=ANY}} \leq \max_{0 \leq x \leq 1} \frac{1 + (K-1)\,x}{1 + (K-1)\,x^3} \qquad \text{if } P_{\text{core}}(f) = \alpha \cdot f^3 \tag{9.4}$$

Following, to find the value of x that results in the worst case, denoted as x_{max}, we set to zero the first-order derivative of Equation (9.4) with respect to x. From this procedure we obtain that $2\,(K-1)\,x_{\text{max}}^3 + 3 \cdot x_{\text{max}}^2 - 1 = 0$, and we can simply compute the value of x_{max} for a given value of K. Figure 9.3 presents the resulting approximation factor in Equation (9.4), for $x = x_{\text{max}}$. The approximation factor is not bounded with respect to the value of K, and can therefore be very large when K is large.

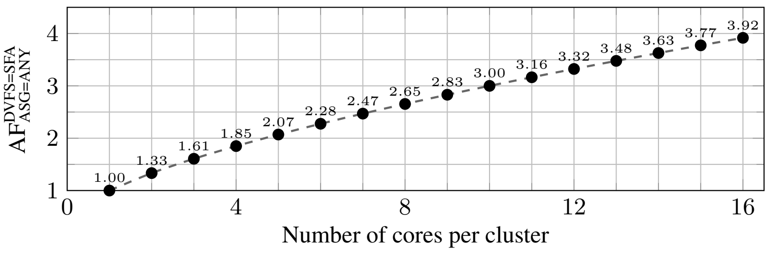


Figure 9.3: Approximation factor for *any* task partition mapping heuristic that uses $\left\lceil \frac{M^\star}{K} \right\rceil$ clusters with non-empty task sets, and SFA to decide the DVFS levels of individual clusters, when $P_{\text{core}}(f) = \alpha \cdot f^3$ with $\alpha > 0$.

The *tightness* of our analysis can be proven by a concrete example built according to the above conditions. In other words, for given values of K, V, and w_M, in case $P_{\text{core}}(f) = \alpha \cdot f^3$, the worst case occurs when we have $(K-1)\,V$ task sets with cycle utilization equal to zero and $K-1$ task sets with cycle utilization $x_{\text{max}} \cdot w_M$, i.e., the worst case occurs when $w_1 = w_2 = \cdots = w_{M-K-1} = w_{M-K} = 0$ and when $w_{M-K+1} = w_{M-K+2} = \cdots = w_{M-2} = w_{M-1} = x_{\text{max}} \cdot w_M$. For example, consider a system with $V = 8$ clusters, $K = 8$ cores per cluster, $\eta = 0$ cluster power consumption, $P_{\text{core}}(f) = 2\frac{\text{Watts}}{\text{GHz}^3} \cdot f^3$ power consumption per core, and a hyper-period among all tasks of $D = 1\,\text{s}$. After the task partitioning stage, the task sets to be assigned to the clusters have cycle utilizations $w_1 = w_2 = \cdots = w_{55} = w_{56} = 0$, $w_{57} = w_{58} = \cdots = w_{62} = w_{63} = 3.544 \cdot 10^8 \frac{\text{cycles}}{\text{second}}$, and $w_{64} = 10^9 \frac{\text{cycles}}{\text{second}}$. Both CCH and BUH will assign task sets $\mathbf{S}_{57}, \mathbf{S}_{58}, \ldots, \mathbf{S}_{64}$ to cluster I_8, which according to Equation (9.1) results in an energy consumption under SFA of 6.96 J. The optimal solution however, will assign task set $\mathbf{S}_{57}$ to cluster I_1, task set $\mathbf{S}_{58}$ to cluster I_2, task set $\mathbf{S}_{59}$ to cluster I_3, …, and task set $\mathbf{S}_{64}$ to cluster I_8, which results in an energy consumption of 2.62 J (*which is equivalent to the energy consumption for having a per-core DVFS platform in this example*). The ratio between these two energy consumptions is 2.65, which corresponds to the result of Equation (9.4) when $K = 8$ (as also seen in Figure 9.3), since for such a case we have that $x_{\text{max}} = 0.3544$.

The above example can be easily extended for any value of K, such that $V \geq K$. For instance, consider a system with $V = 16$ clusters, $K = 16$ cores per cluster, and the same $\eta = 0$, $P_{\text{core}}(f) = 2\frac{\text{Watts}}{\text{GHz}^3} \cdot f^3$, and $D = 1\,\text{s}$ as in the previous example. By solving $2\,(K-1)\,x_{\text{max}}^3 + 3 \cdot x_{\text{max}}^2 - 1 = 0$ when $K = 16$, we know that for this value of K, Equation (9.4) is maximized when $x = x_{\text{max}} = 0.2917$. In other words, if after the task partitioning stage, the task set with the highest cycle utilization is, e.g., $w_{256} = 10^9$, then the worst case occurs when we have $K-1$ task sets with cycle utilization $2.917 \cdot 10^8 \frac{\text{cycles}}{\text{second}}$, and $(K-1)\,V$ task sets with cycle utilization zero, i.e., when $w_1 = w_2 = \cdots = w_{239} = w_{240} = 0$, $w_{241} = w_{242} = \cdots = w_{254} = w_{255} = 2.917 \cdot 10^8 \frac{\text{cycles}}{\text{second}}$, and $w_{256} = 10^9 \frac{\text{cycles}}{\text{second}}$. For this second example, both CCH and BUH will assign task sets $\mathbf{S}_{241}, \mathbf{S}_{242}, \ldots, \mathbf{S}_{256}$ to cluster I_{16}, which according to Equation (9.1), will result in an energy consumption under SFA of 10.75 J. However, the optimal solution will choose to assign one non-empty task set to every cluster, i.e., task set $\mathbf{S}_{241}$ to cluster I_1, task set $\mathbf{S}_{242}$ to cluster I_2, …, and task set $\mathbf{S}_{256}$ to cluster

I_{16}, resulting in an energy consumption of $2.74\,\text{J}$ (*which again is equivalent to the energy consumption for having a per-core DVFS platform in this example*). The ratio between these two energy consumptions is 3.92, which corresponds to the result of Equation (9.4) when $K = 16$ (as also seen in Figure 9.3), since for such a case we have that $x_{\text{max}} = 0.2917$. Similarly, we can extend these examples for any other value of K, such that $V \geq K$, thus proving the *tightness* of our analysis for CCH and BUH for all K.

9.3 Assignment Properties

This section presents some assignment properties that are later used in Section 9.4. We have two cases: (1) when the task sets with the highest cycle utilization in each cluster are given, such that the DVFS levels of the clusters under SFA are known, and (2) when they are not given, i.e., all cases need to be considered.

9.3.1 Given Highest Cycle Utilization Task Sets Assigned to Every Cluster

For simplicity of presentation, we first define set $\mathbf{Y} = \{y_1, y_2, \ldots, y_V\}$ containing the indexes of the task sets with the highest cycle utilizations mapped to every cluster, such that task set $\mathbf{S}_{y_j}$ will have the highest cycle utilization inside cluster I_j. In other words, the set of indexes for which it holds that $y_j = \ell_{j,K}$ for all clusters $j = 1, 2, \ldots, V$. Given that the clusters are ordered with respect to their DVFS levels, it holds that $y_1 < y_2 < \cdots < y_V$. Clearly, task set $\mathbf{S}_M$ will always be the *highest cycle utilization task set* in the *highest DVFS cluster* I_V, or in other words, it always holds that $y_V = M$. Furthermore, as there are M cores in the chip and M task sets, such that one task set is mapped to every core, and given that the task sets are ordered according to their cycle utilizations, the *highest cycle utilization task set* in cluster I_j cannot be a task set with a cycle utilization less than $w_{j \cdot K}$, as this would contradict the definition of set $\mathbf{Y}$. Therefore, by definition, we know that $y_j \geq j \cdot K$ for all clusters $j = 1, 2, \ldots, V$, as otherwise there would be no feasible assignment to satisfy the definition of set $\mathbf{Y}$. This statement is illustrated in the example in Figure 9.4.

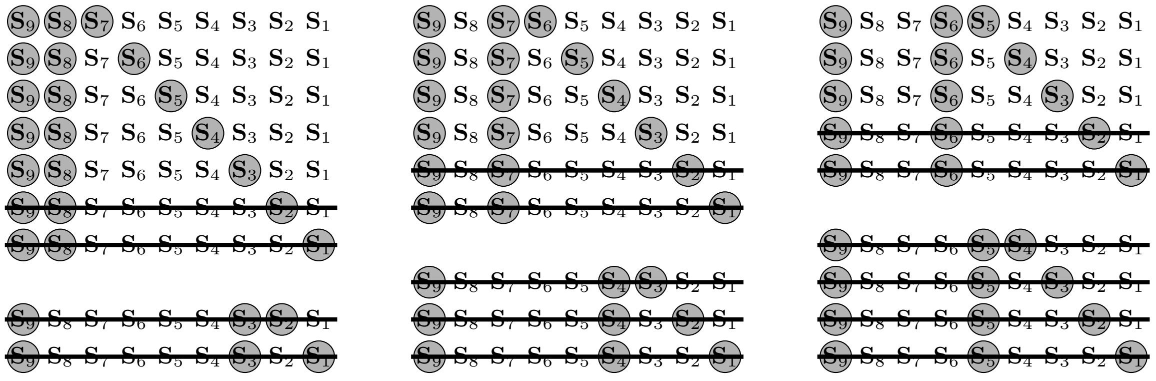
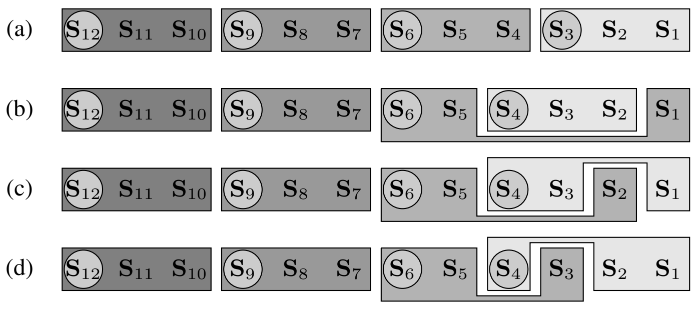


Figure 9.4: Examples of different sets $\mathbf{Y}$, for a chip with $V = 3$ and $K = 3$. The task sets with highest cycle utilizations in each cluster (i.e., the task sets whose indexes are inside set $\mathbf{Y}$) are circled and colored. Unfeasible combinations that contradict the definition of $\mathbf{Y}$ are crossed out. For example, if $y_1 < 3$, we could not assign 3 task sets to cluster I_1 and still have that w_{y_1} is the highest cycle utilization inside cluster I_1. A similar situation occurs for y_2 and y_3. Cases in which $y_3 < M$, all unfeasible, are omitted from the figure.

From the above argument, in order to satisfy the definition of set $\mathbf{Y}$, the possible combinations of task sets assignments to consider can be reduced. Nevertheless, there will still exist many feasible combinations, among which there will be one or more combinations that result in the minimum energy consumption under SFA. For example, for $V = 4$ and $K = 3$, Figure 9.5a shows the only possible assignment combination for the case in which $\mathbf{Y} = \{3, 6, 9, 12\}$. Contrarily, there are three possible assignment combinations for the case in which $\mathbf{Y} = \{4, 6, 9, 12\}$, as seen in Figures 9.5b, 9.5c, and 9.5d. With this in mind, for a given set $\mathbf{Y}$, Theorem 6 provides an important property for assigning task sets to clusters in order to minimize the energy consumption. For example, for the case in which $\mathbf{Y} = \{4, 6, 9, 12\}$, Theorem 6 proves that the combination in Figure 9.5b will always minimize the energy consumption, and thus the combinations in Figures 9.5c and 9.5d do not need to be considered.



(a)
(b)
(c)
(d)
$\mathbf{S}_{12}$ $\mathbf{S}_{11}$ $\mathbf{S}_{10}$ $\mathbf{S}_9$ $\mathbf{S}_8$ $\mathbf{S}_7$ $\mathbf{S}_6$ $\mathbf{S}_5$ $\mathbf{S}_4$ $\mathbf{S}_3$ $\mathbf{S}_2$ $\mathbf{S}_1$

Figure 9.5: Examples of possible task set assignments, for a chip with $V = 4$ and $K = 3$. Figure (a) shows the only possible assignment combination when $\mathbf{Y} = \{3, 6, 9, 12\}$. Figures (b), (c), and (d) show the three possible assignments when $\mathbf{Y} = \{4, 6, 9, 12\}$, for which Theorem 6 proves that combination (b) minimizes the energy consumption.

Theorem 6 *Suppose that $\frac{P_{core}(f)}{f}$ is a non-decreasing function with respect to f, for any $f \geq f_{crit}$. Consider a given set $\mathbf{Y}$, where the highest cycle utilizations of clusters I_i and I_h are w_{y_i} and w_{y_h}, respectively, and it holds that $w_{y_i} \leq w_{y_h}$ when $i < h$. Given that the highest cycle utilizations of the clusters are known, then the DVFS levels of both cluster under SFA are also known, and for this case it holds that $f_i = \max\{f_{crit}, w_{y_i}\} \leq f_h = \max\{f_{crit}, w_{y_h}\}$. Furthermore, consider task sets $\mathbf{S}_j$ and $\mathbf{S}_k$ such that $w_j \leq w_k \leq w_{y_i} \leq w_{y_h}$, where task set $\mathbf{S}_j$ is assigned to cluster I_i and task set $\mathbf{S}_k$ is assigned to cluster I_h. From these assumptions, under SFA and for the given set $\mathbf{Y}$, swapping the assignment such that task set $\mathbf{S}_j$ is assigned to cluster I_h and task set $\mathbf{S}_k$ is assigned to cluster I_i consumes no more energy than the original assignment.*

Proof. The overall energy consumption before swapping the tasks can be computed as

$$E^{\text{before}}_{\text{swapping}} = \cdots + D \cdot P_{\text{core}}(f_h) \cdot \frac{w_k}{f_h} + \cdots + D \cdot P_{\text{core}}(f_i) \cdot \frac{w_j}{f_i} + \cdots,$$

while the overall energy consumption after swapping the task sets can be computed as

$$E^{\text{after}}_{\text{swapping}} = \cdots + D \cdot P_{\text{core}}(f_h) \cdot \frac{w_j}{f_h} + \cdots + D \cdot P_{\text{core}}(f_i) \cdot \frac{w_k}{f_i} + \cdots.$$

Therefore, the difference in the overall energy consumption after and before swapping can be computed as

$$\begin{aligned} E^{\text{after}}_{\text{swapping}} - E^{\text{before}}_{\text{swapping}} &= D\left[P_{\text{core}}(f_h)\frac{w_j}{f_h} + P_{\text{core}}(f_i)\frac{w_k}{f_i} - P_{\text{core}}(f_h)\frac{w_k}{f_h} - P_{\text{core}}(f_i)\frac{w_j}{f_i}\right] \\ &= D\left[\frac{P_{\text{core}}(f_h)}{f_h} - \frac{P_{\text{core}}(f_i)}{f_i}\right](w_j - w_k). \end{aligned}$$

Given that $\frac{P_{\text{core}}(f)}{f}$ is an increasing function with respect to f, it holds that $\frac{P_{\text{core}}(f_h)}{f_h} - \frac{P_{\text{core}}(f_i)}{f_i} \geq 0$. Moreover, since it also holds that $w_j - w_k \leq 0$, we have that

$$D\left[\frac{P_{\text{core}}(f_h)}{f_h} - \frac{P_{\text{core}}(f_i)}{f_i}\right](w_j - w_k) \leq 0 \qquad \Rightarrow \qquad E^{\text{after}}_{\text{swapping}} \leq E^{\text{before}}_{\text{swapping}}$$

and thus the theorem is proven. □

For example, by translating the statement in Theorem 6 to the example in Figure 9.5, the task set assignment before the swapping could refer to the combination in Figure 9.5c (i.e., task set $\mathbf{S}_1$ assigned to cluster I_1 and task set $\mathbf{S}_2$ assigned to cluster I_2), while the task set assignment after the swapping could refer to the combination in Figure 9.5b (i.e., task set $\mathbf{S}_1$ assigned to cluster I_2 and task set $\mathbf{S}_2$ assigned to cluster I_1). Therefore, according to Theorem 6, the combination in Figure 9.5b consumes no more energy than the combination in Figure 9.5c. The same applies when comparing the assignment of task sets $\mathbf{S}_1$ and $\mathbf{S}_3$ in Figure 9.5b, to the assignment of task sets $\mathbf{S}_1$ and $\mathbf{S}_3$ in Figure 9.5d, where according to Theorem 6, the combination in Figure 9.5b consumes no more energy than the combination in Figure 9.5d.

Algorithm 8 Optimal Greedy Algorithm for a Given Set $\mathbf{Y}$

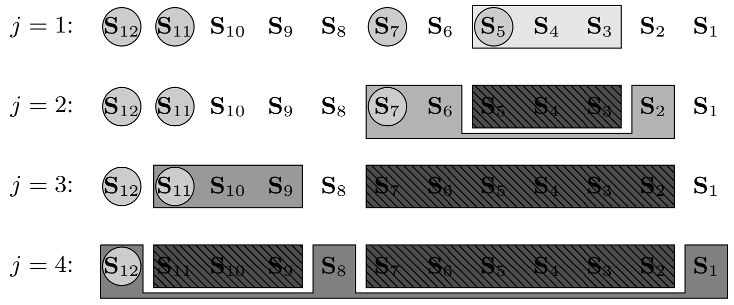
```
Input: A given set $\mathbf{Y} = \{y_1, y_2, \dots, y_V\}$;
Output: The task set assignment with the minimal energy consumption based on the given set $\mathbf{Y}$;
1: for all $j = 1, 2, \dots, V$ (i.e., for all clusters) do
2:     $\mathbf{L}_j \leftarrow \mathbf{S}_{y_j}$; {Assign task set $\mathbf{S}_{y_j}$ to to cluster $I_j$}
3:     By iterating through the unassigned task sets decreasingly with respect to their cycle utilizations, assign
       the $K - 1$ task sets whose cycle utilization is less than or equal to $w_{y_j}$ to cluster $I_j$ (by filling set $\mathbf{L}_j$);
4:     Remove assigned task sets and the cores in set $\mathbf{L}_j$ from the problem;
5: end for
6: return $\mathbf{L}_1, \mathbf{L}_2, \dots, \mathbf{L}_V$;
```

Based on Theorem 6, when $\frac{P_{\text{core}}(f)}{f}$ is a non-decreasing function with respect to f for any $f \geq f_{\text{crit}}$, and when the *highest cycle utilization task sets* on every cluster are already selected (i.e., when set $\mathbf{Y} = \{y_1, y_2, \dots, y_V\}$ is given), we can optimally solve the *multiple voltage islands assignment* problem under SFA by implementing a greedy algorithm. The pseudo-code of such a greedy solution is presented in Algorithm 8. Algorithm 8 starts by assigning the $K - 1$ task sets with highest cycle utilizations that do not exceed w_{y_1} onto cores inside cluster I_1. Given that now all the K cores inside cluster I_1 have task sets assigned to them, we can remove the cluster, the cores, and the assigned task sets from the input instance of the problem, thus creating a new sub-problem with one less element inside set $\mathbf{Y}$. This process is repeated for clusters $I_2, \dots, I_{V-1}$ until only $K - 1$ task sets remain, which are then assigned onto cores inside cluster I_V. For a given set $\mathbf{Y}$, the overall time complexity of such a greedy algorithm is $O(M \cdot V)$. Finally, from Theorem 6 and Algorithm 8, we have the following corollary that summarizes the assignment property derived in this section.

Corollary 1 *For a given set $\mathbf{Y}$, when using SFA to decide the DVFS levels on individual clusters, in order to assign task sets to cluster I_j, the optimal assignment solution assigns $K - 1$* adjacent and consecutive *task sets with the* highest cycle utilizations *whose values are less than or equal to the corresponding cycle utilization w_{y_j}.*

Example

Consider a system with $V = 4$ and $K = 3$, and a given set $\mathbf{Y} = \{5, 7, 11, 12\}$. Under SFA, the optimal assignment of the task sets onto cores belonging to different clusters is shown in Figure 9.6, derived through Algorithm 8, based on Theorem 6 and Corollary 1. According to Algorithm 8, we first focus on cluster I_1 (i.e., $j = 1$ in the figure), and assign $K - 1$ task sets with cycle utilizations less than or equal to w_{y_1} onto cores in cluster I_1. In this example task set $\mathbf{S}_5$ has the highest cycle utilization in cluster I_1 (i.e., $y_1 = 5$), and therefore task sets $\mathbf{S}_4$ and $\mathbf{S}_3$ are also assigned onto cores belonging to I_1. Furthermore, task sets $\mathbf{S}_3$, $\mathbf{S}_4$, and $\mathbf{S}_5$ are now removed from the sub-problem (colored in dark-gray and crossed out in the figure when $j > 1$). In order to solve the new sub-problem, we focus on cluster I_2 (i.e., $j = 2$ in the figure). Given that task set $\mathbf{S}_7$ has the highest cycle utilization on cluster I_2, task sets $\mathbf{S}_6$ and $\mathbf{S}_2$ are assigned onto cores belonging to I_2. These task sets are removed from the sub-problem (colored in dark-gray and crossed out in the figure $j > 2$) and the process is repeated for $j = 3$ and $j = 4$ until all task sets are mapped.



j = 1:
S12 S11 S10 S9 S8 S7 S6 S5 S4 S3 S2 S1
j = 2:
S12 S11 S10 S9 S8 S7 S6 S5 S4 S3 S2 S1
j = 3:
S12 S11 S10 S9 S8 S7 S6 S5 S4 S3 S2 S1
j = 4:
S12 S11 S10 S9 S8 S7 S6 S5 S4 S3 S2 S1

Figure 9.6: Example of the process executed by Algorithm 8, for $V = 4$ and $K = 3$, when $\mathbf{Y} = \{5, 7, 11, 12\}$. The task sets with highest cycle utilizations in each cluster are circled and colored. Every value of j represents the assignment of task sets to cluster I_j, i.e., one iteration inside the algorithm.

9.3.2 All Possible Highest Cycle Utilization Task Sets

In case the highest cycle utilization task sets inside each cluster are *not* given, then one possibility to derive the optimal solution is a brute-force approach that tries every possible combination for set $\mathbf{Y}$ and selects the one that would result in the minimum energy consumption. Given that the number of combinations for binomial coefficient $\binom{a}{b}$ is upper bounded by $\left(\frac{e \cdot a}{b}\right)^b$, where e is Euler's number, therefore, the total number of combinations to be considered for such a brute-force approach is at most $\binom{K \cdot V - K}{V-1} \leq (e \cdot K)^{V-1}$. Hence, an algorithm that tries every possible combination for set $\mathbf{Y}$ and applies Algorithm 8 for each possibility will have total time complexity $O\left(e^{V-1} \cdot K^V \cdot V^2\right)$. This time complexity is polynomial when V is a constant, but it is still too high and does not solve the problem in an efficient manner.

9.4 Dynamic Programming Solution

This section details an efficient dynamic programming algorithm to solve the *multiple voltage islands assignment problem*, called the Dynamic Voltage Island Assignment (DYVIA) algorithm.

9.4.1 Description of the DYVIA Algorithm

Our DYVIA algorithm is based on a property described in Theorem 7 and illustrated in Figure 9.7, which comes as a direct result of Theorem 6 and Corollary 1. For simplicity of presentation, we define set $\mathbf{L} = \{\ell_0, \ell_1, \ell_2, \ldots, \ell_K\}$ (overloading in this chapter symbol $\mathbf{L}$ defined in Chapter 3.5) as a set containing the indexes of the task sets assigned onto cores in a *general* cluster (as opposed to set $\mathbf{L}_j$, defined for a particular cluster I_j), such that $\ell_1 < \ell_2 < \cdots < \ell_K$, with ℓ_0 auxiliary and less than ℓ_1. Naturally, similar to the definition of $\mathbf{L}_j$, it will hold that $w_{\ell_1} \leq w_{\ell_2} \leq \cdots \leq w_{\ell_K}$. Furthermore, we define set $\Lambda(i,j)$ as a set that contains all possible $\mathbf{L}$ sets that satisfy Theorem 7. Namely, for a given (i,j), set $\Lambda(i,j)$ stores all the *potentially optimal combinations*, such that $\ell_0 = i - 1$, $\ell_K = j$, and $\ell_h = \ell_{h-1} + 1 + n \cdot K$ for all $h = 1, 2, \ldots, K-1$ with $n \in \mathbb{N}^0$.

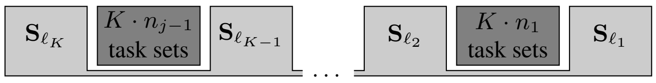


Figure 9.7: Adjacent remaining task sets groups, with $n_i \in \mathbb{N}^0$.

Theorem 7 *When assigning task sets with indexes $\mathbf{L} = \{\ell_1, \ell_2, \ldots, \ell_K\}$ onto cores in cluster I_j, in the optimal solution that uses SFA to decide the DVFS levels on individual clusters, all the task sets assigned onto cores in clusters $I_1, I_2, \ldots, I_{j-1}$ will form up to $j-1$ groups of* remaining adjacent task sets *and every group will have a number of task sets that is a multiple of K.*

Proof. From Theorem 6 and Corollary 1, it is quite clear that Theorem 7 holds for any given $\mathbf{Y}$ set, including the optimal one. □

Based on the property described in Theorem 7, our DYVIA algorithm can reduce the number of combinations to evaluate. Moreover, as any other dynamic programming algorithm, it relies on the optimality of small sub-problems whose results are saved on a table, thus reducing the total time complexity. Specifically, our *dynamic programming function* is defined as *DYVIA* (i,j), where i is the index of the first task set to be considered in this sub-problem, and j is the index of the last task set to be considered in this sub-problem. Namely, function *DYVIA* (i,j) returns the minimum energy consumption for the assignment of task sets $\mathbf{S}_i, \mathbf{S}_{i+1}, \ldots, \mathbf{S}_{j-1}, \mathbf{S}_j$ onto cores, using a number of clusters equal to $v = \frac{j-i+1}{K}$ (from Corollary 1, $j - i + 1$ will always be an integer multiple of K). Particularly, function *DYVIA* (i,j) only focuses on the *highest DVFS level cluster* in the sub-problem and on the task sets $\mathbf{S}_{\ell_1}, \mathbf{S}_{\ell_2}, \ldots, \mathbf{S}_{\ell_K}$ that are to be assigned onto cores in this cluster. In order to derive the optimal assignment of task sets onto cores that resulted in this minimal energy consumption after the dynamic programming algorithm finished, a standard backtracking technique may be used, e.g., by building an additional backtracking table in which entry *DYVIA*$^{\text{back-}}_{\text{tracking}}$ (i,j) contains the task sets indexes $\{\ell_1, \ell_2, \ldots, \ell_K\}$ that resulted in the minimum energy consumption for sub-problem *DYVIA* (i,j). Among task sets $\mathbf{S}_i, \mathbf{S}_{i+1}, \ldots, \mathbf{S}_{j-1}, \mathbf{S}_j$ to be mapped in sub-problem *DYVIA* (i,j), those not mapped to the *highest DVFS level cluster* in this sub-problem are dealt with in other (smaller) sub-problems.

From the definition, just as for any other cluster, the *highest DVFS level cluster* on each *DYVIA* (i, j) sub-problem will hold K cores. Given that $w_i \leq w_{i+1} \leq \cdots \leq w_{j-1} \leq w_j$, it is easy to see that task set $\mathbf{S}_j$ is always assigned onto a core in this *highest DVFS level cluster* and that, according to SFA, its cycle utilization defines the DVFS levels of the *highest DVFS level cluster*. Therefore, function *DYVIA* (i, j) has to decide which other $K - 1$ task sets (among $\mathbf{S}_i, \mathbf{S}_{i+1}, \ldots, \mathbf{S}_{j-2}, \mathbf{S}_{j-1}$) should be assigned to the *highest DVFS level cluster* in this sub-problem. To accomplish this goal, we consider all the *potentially optimal combinations* that satisfy Theorem 7, i.e., we consider all $\mathbf{L}$ sets inside $\Lambda(i, j)$.

For each possible assignment combination evaluated in a sub-problem, the energy consumption of the *highest DVFS level cluster* is computed through function $E(\mathbf{L})$ as shown in Equation (9.5), which is similar to Equation (9.1), but considering set $\mathbf{L}$ instead of set $\mathbf{L}_j$. Therefore, the total energy consumption for each assignment combination evaluated in a sub-problem is computed as the summation of $E(\mathbf{L})$ and the minimum energy consumption of each one of the smaller sub-problems for every group of remaining adjacent task sets, as stated in Theorem 7. Once the total energy consumption for every combination is obtained, the minimum one is chosen as the result of the sub-problem under consideration.

$$E(\mathbf{L}) = \begin{cases} 0 & \text{if } \sum_{i=1}^{K} w_{\ell_i} = 0 \\ D\left(\eta + \frac{P_{\text{core}}(\max\{f_{\text{crit}}, w_{\ell_K}\})}{\max\{f_{\text{crit}}, w_{\ell_K}\}} \sum\limits_{i=1}^{K} w_{\ell_i}\right) & \text{otherwise} \end{cases} \tag{9.5}$$

The initial conditions for building the dynamic programming table are defined in Equation (9.6), which builds the dynamic programming table for all sub-problems with K consecutive task sets, i.e., $j - i + 1 = K$. The recursive dynamic programming function is defined in Equation (9.7). The pseudo-code for a bottom-up implementation of our DYVIA algorithm is presented in Algorithm 9.

$$\textit{DYVIA}(i, j) = \begin{cases} 0 & \text{if } i = 0, \text{ or if } \sum_{h=i}^{j} w_h = 0, \\ & \text{or if } 0 \leq j < i + K - 1 \leq M \\ D\left(\eta + \frac{P_{\text{core}}(\max\{f_{\text{crit}}, w_j\})}{\max\{f_{\text{crit}}, w_j\}} \sum\limits_{h=i}^{j} w_h\right) & \text{otherwise} \end{cases} \tag{9.6}$$

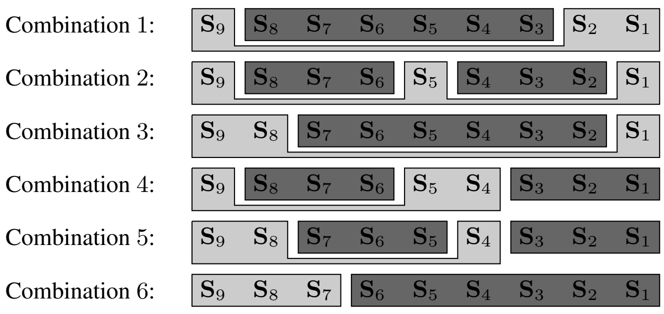
$$\textit{DYVIA}(i, j) = \min_{\forall \mathbf{L} \in \Lambda(i,j)} \left\{ E(\mathbf{L}) + \sum_{n=1}^{K} \textit{DYVIA}(\ell_{n-1} + 1, \ell_n - 1) \right\} \tag{9.7}$$

Example

Following, we consider a system with $V = 3$ and $K = 3$ (i.e., $M = 9$), and present an example to conceptually illustrate how algorithm DYVIA works. The solution to the problem, i.e., the optimal energy consumption under SFA, is found in entry *DYVIA* $(1, M)$ of our dynamic programming table, i.e., *DYVIA* $(1, 9)$ in this example. In order to build entry *DYVIA* $(1, 9)$ according to Equation (9.7), the algorithm evaluates the energy consumption of assigning task sets according to all sets $\mathbf{L} \in \Lambda(1, 9)$, i.e., it checks all *potentially optimal combinations* satisfying Theorem 7, as shown in Figure 9.8. As stated in Theorem 7, given that $K = 3$, every sub-problem contains 3 or 6 task sets (not more, as there are 9 task sets in the main problem). For each evaluated combination, i.e., for each $\mathbf{L} \in \Lambda(1, 9)$, function $E(\mathbf{L})$ computes the energy consumption for cluster I_3, and the algorithm refers to the entries built previously in order to obtain the lowest energy consumption under SFA for the resulting sub-problems. Sub-problems *DYVIA* $(1, 3)$, *DYVIA* $(2, 4)$, *DYVIA* $(3, 5)$, *DYVIA* $(4, 6)$, *DYVIA* $(5, 7)$, and *DYVIA* $(6, 8)$ are built as initial conditions according to Equation (9.6), as done in lines 1-3 in Algorithm 9. Sub-problems *DYVIA* $(1, 6)$, *DYVIA* $(2, 7)$, and *DYVIA* $(3, 8)$ are solved according to Equation (9.7), as done in lines 4-10 in Algorithm 9. Finally, the main problem *DYVIA* $(1, 9)$ is also solved according to Equation (9.7), as done in lines 11-13 in Algorithm 9.

Algorithm 9 Bottom-up implementation of our Dynamic Voltage Island Assignment (DYVIA) algorithm

```
Input: Number of clusters $V$, number of cores per cluster $K$, and task sets $\{\mathbf{S}_1, \mathbf{S}_2, \ldots, \mathbf{S}_M\}$;
Output: The overall minimum energy consumption under SFA, and backtracking table $\mathit{DYVIA}^{\text{back-}}_{\text{tracking}}$;
    {First, initialize the dynamic programming table for all sub-problems with $K$ consecutive task sets}
 1: for all $h = 1, 2, \ldots, K(V-1)$ do
 2:     $\mathit{DYVIA}(h, h+K-1) \leftarrow$ Result from Equation (9.6) for $i = h$ and $j = h+K-1$;
 3: end for
    {Secondly, fill the rest of the dynamic programming table in a bottom-up manner}
 4: for all $k = 2, 3, \ldots, V-1$ do
 5:     for all $h = 1, 2, \ldots, K(V-k)$ do
 6:         Obtain all potentially optimal $\mathbf{L}$ sets for sub-problem $\mathit{DYVIA}(h, h+k \cdot K-1)$, i.e., obtain set $\Lambda(h, h+k \cdot K-1)$ to use inside Equation (9.7);
 7:         $\mathit{DYVIA}(h, h+k \cdot K-1) \leftarrow$ Result from Equation (9.7) for $i = h$ and $j = h+k \cdot K-1$;
 8:         $\mathit{DYVIA}^{\text{back-}}_{\text{tracking}}(h, h+k \cdot K-1) \leftarrow$ Set $\mathbf{L}$ that resulted in the minimum energy consumption when computing Equation (9.7) in the previous step for $i = h$ and $j = h+k \cdot K-1$;
 9:     end for
10: end for
    {Lastly, compute the final solution for the dynamic programming algorithm}
11: Obtain all potentially optimal $\mathbf{L}$ sets for sub-problem $\mathit{DYVIA}(1, M)$, i.e., obtain set $\Lambda(1, M)$;
12: $\mathit{DYVIA}(1, M) \leftarrow$ Result from Equation (9.7) for $i = 1$ and $j = M$;
13: $\mathit{DYVIA}^{\text{back-}}_{\text{tracking}}(1, M) \leftarrow$ Set $\mathbf{L}$ that resulted in the minimum energy consumption when computing Equation (9.7) in the previous step for $i = 1$ and $j = M$;
14: return Minimum energy consumption under SFA $\mathit{DYVIA}(1, M)$, and backtracking table $\mathit{DYVIA}^{\text{back-}}_{\text{tracking}}$;
```



Combination 1: S9 S8 S7 S6 S5 S4 S3 S2 S1
Combination 2: S9 S8 S7 S6 S5 S4 S3 S2 S1
Combination 3: S9 S8 S7 S6 S5 S4 S3 S2 S1
Combination 4: S9 S8 S7 S6 S5 S4 S3 S2 S1
Combination 5: S9 S8 S7 S6 S5 S4 S3 S2 S1
Combination 6: S9 S8 S7 S6 S5 S4 S3 S2 S1

Figure 9.8: Example of algorithm DYVIA for building *DYVIA* $(1, 9)$, with $V = 3$ and $K = 3$. Each combination corresponds to a set $\mathbf{L} \in \Lambda(1, 9)$ to assign in cluster I_3, for which the algorithm evaluates the resulting energy, returning the minimum among all tested cases. The task sets assigned to I_3 in a combination are boxed in light-gray and the resulting sub-problems are colored in dark-gray.

According to the example of Figure 9.8 for a system with $V = 3$ and $K = 3$, Figure 9.9 illustrates the *potentially optimal combinations* that satisfy Theorem 7 for sub-problems *DYVIA* $(1, 6)$, *DYVIA* $(2, 7)$, and *DYVIA* $(3, 8)$, which are solved according to Equation (9.7).

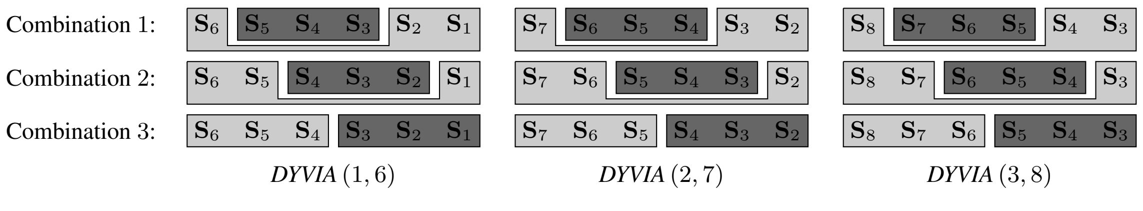


Figure 9.9: Example of algorithm DYVIA for sub-problems *DYVIA* $(1, 6)$, *DYVIA* $(2, 7)$, and *DYVIA* $(3, 8)$, with $V = 3$ and $K = 3$. Each combination corresponds to a set $\mathbf{L}$ inside $\Lambda(1, 6)$, $\Lambda(2, 7)$, or $\Lambda(3, 8)$ to assign in I_2, for which DYVIA evaluates the resulting energy, returning the minimum among all tested cases. Task sets assigned to I_2 in a combination are boxed in light-gray and the resulting sub-problems are colored in dark-gray.

For the backtracking procedure, DYVIA simply stores the indexes of the combination that results in the minimum energy consumption for each sub-problem inside backtracking table $\mathit{DYVIA}^{\text{back-}}_{\text{tracking}}$. For example, in case that *Combination 3* derives the best result for *DYVIA* $(1,9)$ in Figure 9.8, then DYVIA would store $\{1,8,9\}$ in entry $\mathit{DYVIA}^{\text{back-}}_{\text{tracking}}\,(1,9)$. Once the algorithm finishes, in order to know the final assignment, we look inside $\mathit{DYVIA}^{\text{back-}}_{\text{tracking}}\,(1,9)$ to retrieve the indexes of the task sets that should be assigned to I_3. Then, we continue looking inside the backtracking table for the groups of unassigned consecutive task sets, which in this case would mean to look inside $\mathit{DYVIA}^{\text{back-}}_{\text{tracking}}\,(2,7)$ to retrieve the indexes of the task sets that should be assigned to I_2. This process is repeated until all task sets are assigned to their respective clusters. For another example, in case *Combination 4* would have derived the best result for *DYVIA* $(1,9)$ in Figure 9.8, then DYVIA would store $\{4,5,9\}$ in entry $\mathit{DYVIA}^{\text{back-}}_{\text{tracking}}\,(1,9)$. For the retrieving process, after retrieving the indexes from $\mathit{DYVIA}^{\text{back-}}_{\text{tracking}}\,(1,9)$, we would need to look inside $\mathit{DYVIA}^{\text{back-}}_{\text{tracking}}\,(1,3)$ and $\mathit{DYVIA}^{\text{back-}}_{\text{tracking}}\,(6,8)$.

Due to Theorem 7, for this example with $V = 3$ and $K = 3$, DYVIA only checks among 6 *potentially optimal combinations* when computing *DYVIA* $(1,9)$ through Equation (9.7), instead of the 28 *brute-force* combinations for checking all possible assignments (already considering that $\mathbf{S}_9$ is always assigned to I_3).

9.4.2 Complexity Analysis for the DYVIA Algorithm

In this section we analyze the total time complexity for building the dynamic programming table in Algorithm 9. In order to compute *DYVIA* $(1,M)$, all the possible combinations that satisfy Theorem 7 need to be considered. This can be considered equivalent to choosing $V-1$ groups of K task sets from a set of $\frac{(V-1)K}{K} + K - 1$ task sets, which results in a total of $\binom{K+V-2}{V-1}$ combinations that have to be evaluated. Similarly, the total number of combinations that have to be evaluated for the resulting $i \cdot K$ sub-problems with $(V-i)\,K$ task sets is $\binom{K+V-2-i}{V-1-i}$, for all $i = 1, 2, \ldots, V-1$. Therefore, the total number of combinations that needs to be evaluated when building the dynamic programming table, denoted as $\text{DYVIA}_{\text{\#combinations}}$, is

$$\text{DYVIA}_{\text{\#combinations}} = \binom{K+V-2}{V-1} + K\sum_{i=1}^{V-1} i \binom{K+V-2-i}{V-1-i}$$

or

$$\text{DYVIA}_{\text{\#combinations}} = \binom{K+V-2}{K-1} + K\sum_{i=1}^{V-1} (V-i)\binom{K-2+i}{K-1}.$$

By applying the Hockey-Stick Identity and Pascal's rule [46], these last expressions can be rephrased as

$$\text{DYVIA}_{\text{\#combinations}} = \binom{K+V-2}{K-1} + K\binom{K+V-1}{K+1}$$

or

$$\text{DYVIA}_{\text{\#combinations}} = \binom{K+V-2}{V-1} + K\binom{K+V-1}{V-2}.$$

Furthermore, given that the number of combinations for binomial coefficient $\binom{a}{b}$ is upper bounded by $\left(\frac{e\cdot a}{b}\right)^b$, where e is Euler's number, we have that

$$\text{DYVIA}_{\text{\#combinations}} \leq \left(\frac{e\,(K+V-2)}{K-1}\right)^{K-1} + K\left(\frac{e\,(K+V-1)}{K+1}\right)^{K+1}$$

or

$$\text{DYVIA}_{\text{\#combinations}} \leq \left(\frac{e\,(K+V-2)}{V-1}\right)^{V-1} + K\left(\frac{e\,(K+V-1)}{V-2}\right)^{V-2}.$$

Finally, the total time complexity of DYVIA is $\min\left\{O\left(\frac{(e\cdot V)^{K+1}}{K^K}\right), O\left(\left(\frac{e\cdot K}{V}\right)^{V-1}\right)\right\}$, which is polynomial when either V or K are constant (which is the case for a fixed architecture, e.g., Intel's SCC), and is more efficient than the greedy algorithm presented in Section 9.3.2.

9.4.3 Optimal Task Set Assignment Under SFA vs. Optimal DVFS

Although DYVIA derives the optimal task-set-to-cluster assignment with respect to energy consumption when using SFA to decide the DVFS levels on individual clusters, there may exist different assignments that result in a lower energy consumption when each individual cluster is free to use any DVFS algorithm, e.g., the accelerating DVFS schedule based on the *deep sleeping property* (every core is put to sleep after executing all the workload in its ready queue) presented by Yang *et al.* [109] (discussed and illustrated in Chapter 2.2.2). Therefore, in this section we analyze the approximation factor of DYVIA under SFA, against the optimal task set assignment under an optimal DVFS algorithm.

For such a purpose, suppose that $\mathrm{AF}_{\mathrm{SFA}}^{\mathrm{energy}}$ is the approximation factor (by definition, the worst case) of SFA against the optimal DVFS schedule for the task sets assigned on *a single cluster*. Moreover, we define different total energy consumptions as $E_{\mathrm{ASG=DYVIA}}^{\mathrm{DVFS=SFA}}$, $E_{\mathrm{ASG=DYVIA}}^{\mathrm{DVFS=optimal}}$, $E_{\mathrm{ASG=optimal\ DVFS}}^{\mathrm{DVFS=SFA}}$, and $E_{\mathrm{ASG=optimal\ DVFS}}^{\mathrm{DVFS=optimal}}$, where DVFS=SFA implies that SFA is used to decide the DVFS levels on individual clusters, DVFS=optimal implies that an optimal DVFS algorithm is used to decide the DVFS levels on individual clusters, ASG=DYVIA implies that task sets are assigned to clusters using DYVIA, and ASG=optimal DVFS implies that task sets are assigned to clusters using an algorithm which is optimal when used in combination with an optimal DVFS algorithm to decide the DVFS levels on individual clusters. From these definitions, the optimal energy consumption is clearly $E_{\mathrm{ASG=optimal\ DVFS}}^{\mathrm{DVFS=optimal}}$. When using DYVIA to assign task sets to clusters, it holds that $E_{\mathrm{ASG=DYVIA}}^{\mathrm{DVFS=optimal}} \leq E_{\mathrm{ASG=DYVIA}}^{\mathrm{DVFS=SFA}}$, from the definition that the optimal DVFS strategy is adopted. Additionally, give that DYVIA is optimal if individual clusters use SFA and any other task set assignment solution will consume the same or more energy, it also holds that $E_{\mathrm{ASG=DYVIA}}^{\mathrm{DVFS=SFA}} \leq E_{\mathrm{ASG=optimal\ DVFS}}^{\mathrm{DVFS=SFA}}$. Furthermore, according to the definition of the approximation factor for the energy consumption of SFA in a single cluster, it holds that $E_{\mathrm{ASG=optimal\ DVFS}}^{\mathrm{DVFS=SFA}} \leq \mathrm{AF}_{\mathrm{SFA}}^{\mathrm{energy}} \cdot E_{\mathrm{ASG=optimal\ DVFS}}^{\mathrm{DVFS=optimal}}$. Finally, combining these three inequalities, the approximation factor of the total energy consumption for DYVIA under SFA, against the optimal task set assignment under an optimal DVFS algorithm, is presented in Equation (9.8).

$$
\begin{aligned}
E_{\mathrm{ASG=optimal\ DVFS}}^{\mathrm{DVFS=optimal}} \leq E_{\mathrm{ASG=DYVIA}}^{\mathrm{DVFS=optimal}} \leq E_{\mathrm{ASG=DYVIA}}^{\mathrm{DVFS=SFA}} \leq E_{\mathrm{ASG=optimal\ DVFS}}^{\mathrm{DVFS=SFA}} \leq \mathrm{AF}_{\mathrm{SFA}}^{\mathrm{energy}} \cdot E_{\mathrm{ASG=optimal\ DVFS}}^{\mathrm{DVFS=optimal}} \\
\Rightarrow \quad E_{\mathrm{ASG=DYVIA}}^{\mathrm{DVFS=SFA}} \leq \mathrm{AF}_{\mathrm{SFA}}^{\mathrm{energy}} \cdot E_{\mathrm{ASG=optimal\ DVFS}}^{\mathrm{DVFS=optimal}}
\end{aligned}
\tag{9.8}
$$

Therefore, for energy minimization, the approximation factor of DYVIA under SFA, against the optimal task set assignment under an optimal DVFS algorithm, is the same as the approximation factor of SFA for energy minimization on a single cluster, which we have already studied in detail in Chapter 8, with the corresponding results presented in Theorem 1 and Theorem 2, and illustrated in Figure 8.12. In this way, Equation (9.8) extends the theoretical analysis presented in Chapter 8 in order to consider systems with multiple clusters/voltage islands.

9.5 Experimental Evaluations on SCC

This section presents experimental evaluations conducted on Intel's Single Chip Cloud computer (SCC). We compare the resulting energy consumption and the required execution time of the heuristic algorithms presented in Section 9.2, i.e., CCH, BUH, and EOH (with 200 iterations for seeking improvement from the current solution, using CCH as initial solution), against our dynamic programming algorithm, i.e., DYVIA.

9.5.1 Setup

The experiments are conducted on Intel's Single Chip Cloud computer (SCC) [36], a research platform that integrates 48 cores on a single chip, where the individual IA (P54C) cores can run at frequencies between 100 MHz and 800 MHz. In Intel's SCC, the cores are clustered into groups of eight cores sharing a common voltage, while the frequency can be selected every two cores (i.e., a tile), such that there can be up to four different frequencies inside every cluster of eight cores sharing a voltage. The voltages of the clusters and the frequencies of the cores are managed through the Voltage Regulator Controller (VRC), which also provides users with the ability to read the instantaneous current consumption and voltage on every cluster, from which we can infer the total power consumed in every cluster. A figure illustrating a block-diagram of SCC is presented in Figure 9.10. Intel's SCC runs a single-core Linux (kernel version 3.1.4) on every core.

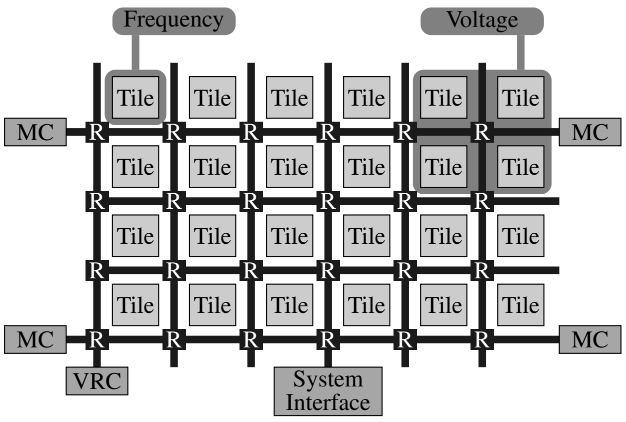
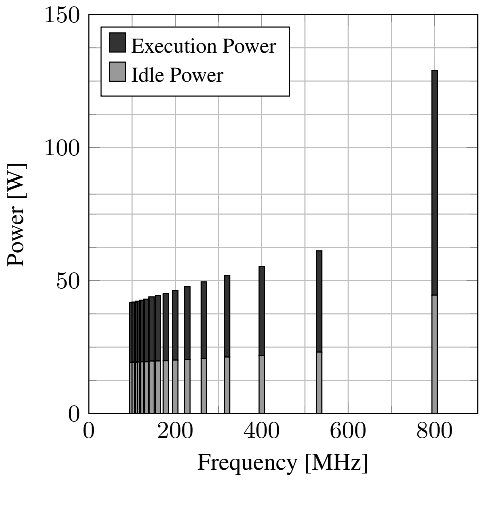


Figure 9.10: Block-diagram of Intel's SCC architecture [36], with two IA (P54C) cores per tile, one router (R) associated with each tile, four memory controllers (MC) for off-die (but on-board) DDR3 memory, and a Voltage Regulator Controller (VRC) to set the voltages of the clusters and the frequencies of the cores.

Given that algorithms EOH and DYVIA internally compute the expected energy consumption of different task sets assignments in order to compare their energy efficiency, they both need a power consumption model/profile of the system. For such a purpose, we conduct experimental measurements on SCC to obtain such power consumption profile, and the results are presented in Figure 9.11. The figure shows all the available execution frequencies, their corresponding minimum voltages for stable execution[1], together with the measured idle power[2] and execution power consumptions for all 48 cores. Some error is present in this power profile. Part of it is due to the resolution of the voltage and current meters inside SCC (around 0.3 W resolution), but mostly because our SCC platform has one *faulted* core, and it is not possible for us to estimate the power consumption of this core for every frequency.



Frequency	Min. Voltage	Idle Power	Execution Power
100 MHz	0.8 V	19.237213 W	22.441350 W
106 MHz	0.8 V	19.402374 W	22.538383 W
114 MHz	0.8 V	19.402374 W	22.867801 W
123 MHz	0.8 V	19.468439 W	23.165997 W
133 MHz	0.8 V	19.468439 W	23.560321 W
145 MHz	0.8 V	19.763666 W	24.119624 W
160 MHz	0.8 V	19.794633 W	24.584324 W
178 MHz	0.8 V	19.864827 W	25.348004 W
200 MHz	0.8 V	20.129086 W	26.168786 W
228 MHz	0.8 V	20.391280 W	27.293623 W
266 MHz	0.8 V	20.756701 W	28.771025 W
320 MHz	0.8 V	21.280769 W	30.674611 W
400 MHz	0.8 V	21.811671 W	33.449853 W
533 MHz	0.8 V	23.132058 W	38.052265 W
800 MHz	1.1 V	44.549986 W	84.395704 W

Figure 9.11: Experimental power consumption profile for Intel's SCC.

Moreover, Figure 9.11 shows that if a core configured to execute at a high frequency is idle, it would consume more energy than an idle core that is configured to execute at a low frequency. However, given that it is not possible to shut down cores or clusters on the SCC, the 19.237 W of idle power consumption for always having all clusters active and all cores idle at 100 MHz is an offset that no algorithm can improve. Hence, for this power consumption profile, when algorithm EOH or DYVIA internally compute the expected

[1]In *The SCC Programmer's Guide* v1.0, the minimum voltage for stable execution bellow 460 MHz is said to be 0.7 V. However, in the RCCE manycore message passing library v2.0 provided with SCC, this was changed to 0.8 V due to stability problems.

[2]The changes in the idle power for different frequencies, seen in Figure 9.11, are due to background processes of the operating system.

energy consumption of different task sets assignments, we consider the values of f_{crit} and η both to be zero in Equations (9.1), (9.6) and (9.7). Furthermore, in the reported results of our experimental evaluations, this $19.237\,\mathrm{W}$ offset is subtracted from the measurements to fairly compare the *effective* performance of the different evaluated algorithms.

With regards to the tasks, we consider two different *single-threaded* benchmarks: (1) a Fast Fourier Transform (FFT) digital filter, and (2) an edge detection algorithm for images. Every instance of a benchmark represents one task, and every task is periodically executed in the assigned core. For each task instance of the FFT benchmark, the input for the FFT filter consist of a discrete signal composed of 10^5 samples, at a sampling rate of $100\,\mathrm{kHz}$, of two sine waves added together (specifically, $5\,\mathrm{kHz}$ and $12\,\mathrm{kHz}$) plus different random noise for every run. For the tasks using the edge detection algorithm, the input is a 640×480 pixels *.bmp* image, randomly chosen for every period among a database of 2500 pictures. Namely, for all tasks using one type of benchmark, the input is randomly chosen for every period; however, the size of the input remains constant. This is intentionally done in order to allow us to obtain a lower-bound for the worst-case execution cycles of both benchmarks through experimental measurements on the SCC. Particularly, we execute both benchmarks 10^3 times at every available DVFS level on SCC, and we use the highest measured value for each benchmark as the lower-bound for the worst-case execution cycles for each frequency (a summary of these experimental measurements is presented in Figure 9.12). In this way, we are able to account for the effects of cache misses and memory access latencies, whose access time is not scaled when the frequency of a core changes. Namely, for a given benchmark type, this implies that the worst-case execution cycles of all task instances of this benchmark type are equivalent for two tasks that are executing at the same frequency, but that is different for two tasks that are executing at different frequencies. In order to obtain different cycle utilizations for different tasks, the period in which the benchmarks are executed is set according to the desired cycle utilization by considering the measured worst-case execution cycles.

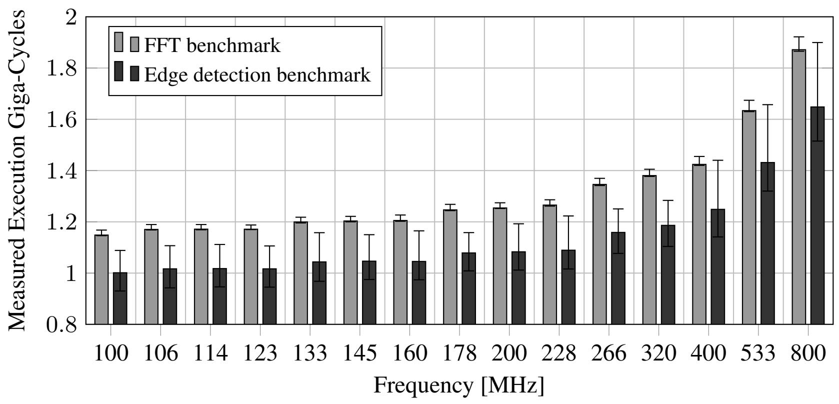


Figure 9.12: Measured experimental execution cycles of the benchmarks used for the evaluations on SCC. Every bar represents the average execution cycles for the given frequency (among 10^3 runs). The error bars represent the minimum and maximum (worst-case) measured execution cycles.

Given that our SCC platform has one *faulted* core, there are only 47 available cores in the platform for our experiments. In order to evaluate the energy efficiency of the evaluated algorithms for different task sets, 12 arbitrary cases are considered. Each one of the 12 cases consists of 200 tasks with different cycle utilizations, specifically, 100 tasks of each benchmark type. The tasks are partitioned using LTF with $M^{\star} = 47$ (all the available cores on *our* SCC). Since the benchmarks are *single-threaded* applications, no communication between cores is needed after the task sets are assigned onto cores. We have integrated all the task set mapping algorithms as a software written in C++ that runs on a single core. Each algorithm decides the task set assignment, configures the voltage of the clusters and the frequencies of the cores according to SFA, and finally executes the benchmarks on the corresponding cores. Given that the purpose of this experiment is to evaluate the energy efficiency of the resulting task set assignment, we measure the total energy consumed by all clusters in the SCC chip during $100\,\mathrm{s}$ after triggering the execution of the benchmarks.

9.5.2 Measurement Errors

The experimental energy consumption measurements later reported in Section 9.5.3 are obtained through the integration of power measurements. Namely, the power consumption of all cores is measured (sampled) every 1 ms, the total measured power is multiplied by the time elapsed since the previous measurement, and this energy consumption value is then added to the total measured energy consumption. Furthermore, for a given task set assignment, we can easily compute the *expected energy consumption* through Equation (9.1), by considering the experimental lower-bound for the worst-case execution cycles of each task (as shown in Figure 9.12), and the same power profile used by EOH and DYVIA to estimate the energy consumption (i.e., the power profile based on experimental measurements presented Figure 9.11). Naturally, for a given task set assignment, we can expect that there will exist some difference between the measured energy consumption and the expected energy consumption. These differences are mainly due to the resolution of the voltage and current meters inside SCC, the presence of one *faulted* core in our SCC platform, the actual execution cycles taken for a task instance to finish (which could be different from the expected worst-case execution cycles), and the intrinsic integration error from the 1 ms resolution in which we measure the total power when computing the consumed energy.

Particularly, the voltage and current meters inside SCC have a resolution of 0.3 W for every power measurement. As already mentioned in Section 9.5.1, this resolution will result in some error when obtaining the power profile in Figure 9.11, as well as in errors on the energy measurements later reported in Section 9.5.3. The *faulted* core results in additional error for all measurements, given that it adds noise to the power measurements, and this noise cannot be easily filtered out. Moreover, a difference between the actual execution cycles of the tasks and our experimental lower-bound for the worst-case execution cycles (for which the expected energy consumptions are computed) will have a clear impact in the energy consumption values. This occurs because energy is the integration of power through time, and changes in the execution time result in changes in the final energy consumption. Finally, as shown in the abstract example in Figure 9.13, integration error is intrinsic to any digital energy consumption measurement in which the energy consumption is indirectly computed by measuring power and time. Namely, energy is the integration of power through time, where power and time are continuous magnitudes. Nevertheless, both power and time can only be measured and quantified by discrete values, with a given resolution for each case. For example, in our experimental setup, we are unable to measure time at intervals shorter than 1 ms.

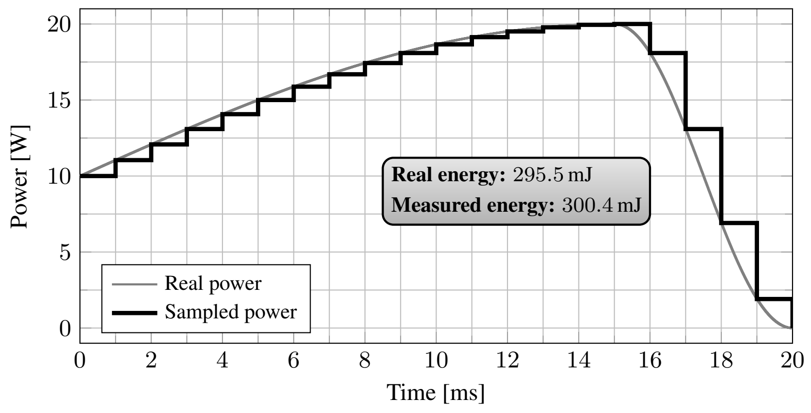


Figure 9.13: Abstract example of the integration error intrinsic to digital energy measurements indirectly computed by measuring power and time. The changes in power occur every 10 µs; however, the measurement sample period is 1 ms. In this example, the resulting measured energy is 1.66% higher than the actually consumed energy.

9.5.3 Results

For each one of the 12 cases of different benchmark utilizations, Table 9.1 presents the experimental measurements of the total energy consumption on SCC, specifically, the average values among 10 consecutive executions during 100 s each, as well as the associated *expected energy consumption* values (defined in Section 9.5.2). The experimental energy consumption values shown in Table 9.1 are obtained through the integration of power measurements, as explained in Section 9.5.2.

Naturally, given that DYVIA results in the optimal task set assignment when using SFA on individual clusters, the *expected energy consumption* of DYVIA is always the lowest among all algorithms. However, for configurations where the heuristics provide good results, the experimental measurements may show values where DYVIA has a higher energy consumption, nevertheless, this effect is only due to the measurement

Table 9.1: Experimental measurement results of the total energy consumption on SCC. For each one of the 12 cases of different benchmark utilizations and for every evaluated algorithm, the table shows the average energy consumption among 10 consecutive executions lasting 100 s each, as well as the associated *expected energy consumption* values.

12 cases	Expected Energy Consumption [J]				Measured Energy Consumption [J]			
	CCH	BUH	EOH	DYVIA	CCH	BUH	EOH	DYVIA
1	1066.3	1066.3	1066.3	1004.5	1072.4	1066.3	1067.4	996.1
2	1085.6	1085.6	1085.6	1024.8	1050.3	1073.1	1072.9	1009.3
3	2144.8	2572.3	2144.8	2045.1	2068.9	2425.9	2057.3	1913.4
4	1070.9	1070.9	1070.9	1013.6	993.9	1029.8	990.5	961.9
5	773.4	773.1	773.4	766.4	787.4	798.5	771.9	811.8
6	2181.9	2620.0	2181.9	2093.0	2090.6	2373.6	2085.8	1981.5
7	952.5	931.2	931.6	895.6	943.1	919.0	933.5	882.9
8	950.8	940.1	920.8	894.3	941.3	916.9	922.5	888.6
9	2120.4	2120.4	2026.0	2013.8	2091.8	2074.5	1896.5	1974.7
10	2081.9	2557.3	2081.9	2006.1	2001.8	2450.4	1999.4	1820.0
11	854.0	854.0	797.6	777.5	744.6	751.5	713.4	728.6
12	890.4	890.4	798.0	797.9	904.7	883.1	810.3	800.8

errors discussed in Section 9.5.2. Specifically, the measured energy consumption values shown in Table 9.1 show a 3.62% error (in average) with respect to the *expected energy consumption* values.

Furthermore, for each one of the 12 cases presented Table 9.1 (expected and measured), we compute the ratios between the energy consumption of the heuristics and the energy consumption of DYVIA, and present the summarized results in Table 9.2. In this way, Table 9.2 presents the minimum, average, and maximum ratios (among the 12 cases) between the energy consumption of each heuristic and the energy consumption of DYVIA, where we can observe that the measured energy ratios are quite similar to the expected energy ratios.

Table 9.2: Summary of the experimental energy ratios between the (expected and measured) energy consumption of each heuristic and the energy consumption of DYVIA.

Algorithm	Expected Energy Ratio			Measured Energy Ratio		
	Min.	Avg.	Max.	Min.	Avg.	Max.
CCH	1.0092	1.0591	1.1159	0.9700	1.0579	1.1297
BUH	1.0088	1.1107	1.2748	0.9837	1.1048	1.3464
EOH	1.0002	1.0348	1.0615	0.9509	1.0324	1.0986

Finally, we conduct a separate experiment to measure the required average execution time of each task set assignment algorithm on SCC running at 533 MHz. All four algorithms need to configure the voltages of the clusters, configure the frequencies of the cores, and assign the task sets onto cores. Therefore, we only measure the execution time of the mapping decision process. Every algorithm is executed 10^4 times, resulting in an average execution time of 0.03 ms for CCH, 0.34 ms for BUH, 220.52 ms for EOH, and 225.78 ms for DYVIA.

9.6 Additional Experimental Evaluations

This section presents simulations for hypothetical platforms with different number of clusters and different number of cores per cluster, such that we can analyze the energy efficiency and execution time of CCH, BUH, EOH, and DYVIA in a more general way.

9.6.1 Setup

In order to observe the effects of the number of clusters and the number of cores per cluster on the energy efficiency and on the execution time of the evaluated algorithms, in this section, the simulations are conducted considering twelve hypothetical platforms with different V and K values, specifically, the platforms resulting from the combinations of $V = \{2, 4, 6\}$ and $K = \{2, 4, 6, 8\}$. The simulations are conducted on a single core of SCC, where again the task set mapping algorithms are integrated as a software written in C++ that runs on a single core. The main difference between the simulations conducted in this section and the experiments conducted in Section 9.5 is that here, after each algorithm decides the assignment of the task sets to cores and clusters, instead of actually executing the benchmarks in the corresponding cores and measuring the consumed energy, the energy consumption is simply computed through Equation (9.1) by considering a power consumption profile, i.e., we compute the *expected energy consumption* as described in Section 9.5.2.

Given that we would also like to test the algorithms in systems with richer DVFS and DPM features than SCC, rather than using the power profile from Figure 9.11, we consider a power consumption profile derived from the experimental results in [30] (a research paper of a customized version of SCC with additional DVFS and DPM capabilities). Figure 12 (Frequency vs. voltage) and Figure 13 (Measured power vs. voltage) from [30] have already been summarized and presented in Figure 3.4a and Figure 3.4b in Chapter 3.3. Furthermore, Chapter 3.3 also approximated the results in Figure 3.4a using a quadratic function, and this quadratic function was used to relate the power values from Figure 3.4b with different frequency settings. In this way, we derived a power consumption profile which was presented in Figure 3.4c. For easy reference, the results from Figure 3.4c are summarized here in Table 9.3 (for all 48 cores running together at the same DVFS levels). Note that, instead of using our simulation framework based on gem5 and McPAT as described in Chapter 4, in this section we use the power profile from Table 9.3 in order to have results that are more related to the practical results for SCC already presented in Section 9.5.

Frequency	Execution Power	Energy for executing 10^8 computer cycles simultaneously in all cores
242.7 MHz	25.38 W	10.46 J
464.5 MHz	37.26 W	8.02 J
686.7 MHz	50.76 W	7.39 J
851.6 MHz	70.73 W	8.31 J
936.6 MHz	91.25 W	9.74 J
1016.9 MHz	110.15 W	10.83 J
1077.8 MHz	125.27 W	11.62 J
1177.0 MHz	161.99 W	13.76 J
1267.0 MHz	201.40 W	15.90 J

Table 9.3: Power consumption profile derived from the measurements presented in [30] for a customized version of SCC with richer DVFS and DPM features. Given that for a constant number of executed cycle, the minimum energy consumption is found at frequency 686.7 MHz, this frequency is the *critical frequency* for this power profile.

For our simulations, we consider the frequencies shown in Table 9.3 as the available execution frequencies for the cores, and divide the total execution power by 48 to obtain the power consumed by every individual core. Furthermore, Table 9.3 also shows the total energy consumption for executing 10^8 computer cycles (simultaneously in all 48 cores) at each corresponding frequency. Given that for a constant number of executed cycles, the minimum energy consumption is found at frequency 686.7 MHz, this frequency is the *critical frequency* for such a power profile. Therefore, when considering this power profile, SFA will never execute at a frequency under 686.7 MHz, because even though doing so might still meet the timing constraints of all tasks in a cluster, it would consume unnecessary energy. Finally, since the work in [30] makes no reference to the power consumed by a cluster for been active, we assume that $\eta = 0$.

In regards to DPM, we consider that the cores can be put to a low-power mode when they have no workload to execute in their ready queues. As discussed in Section 9.1.1, we assume that when a core is in such a low-power mode it consumes κ^{sleep} power. Similar to the experimental measurements from Section 9.5, the power consumption for having all the cores in the low-power mode (i.e., $\kappa^{\text{sleep}} \cdot M$) is an offset that no algorithm can improve. Therefore, in the simulations we set $\kappa^{\text{sleep}} = 0$, to fairly compare the *effective* energy efficiency of the different algorithms.

For every hardware configuration (i.e., for every combination of V and K), we consider 100 different random cases of synthetic tasks. For each case, the total amount of tasks, denoted as R, is randomly chosen between M and $10 \cdot M$, and the the cycle utilizations and periods of the tasks are also randomly chosen. The tasks are partitioned using the LTF strategy described in Chapter 8. We consider two different policies to choose the number of task sets in which to partition task with LTF (i.e., the value of $M^{\star}$). For the first policy, we partition the R tasks into $M^{\star} = M$ task sets using LTF, such that all the available cores are used. For the second policy, we partition the R tasks into $M^{\star} = 1, 2, \ldots, M$ task sets using LTF, then we execute the corresponding task set assignment algorithm for each different resulting task partition (considering $M - M^{\star}$ dummy task sets), and finally choose the task partition that results in the lowest overall energy consumption for the corresponding algorithm. Clearly, the second policy incurs in higher time complexity, but can potentially save more energy.

9.6.2 Results

First, we conduct the same experiment as done in Section 9.5.3 to measure the required average execution time of each task set assignment algorithm on SCC running at 533 MHz, for the twelve hypothetical platforms with different V and K values. Similar to Section 9.5.3, we only measure the execution time of the mapping decision process, and we take the average among 10^4 executions for each algorithm, as shown in Figure 9.14. As expected, heuristics CCH and BUH complete their executions very quickly due to their low complexity. As for EOH and DYVIA, the average execution time increases when the number of K or the number of V increases. However, DYVIA proves to be much faster than EOH in all the evaluated cases, except when $V = 6$ and $K = 8$ (i.e., the case of SCC), where EOH and DYVIA have similar execution times.

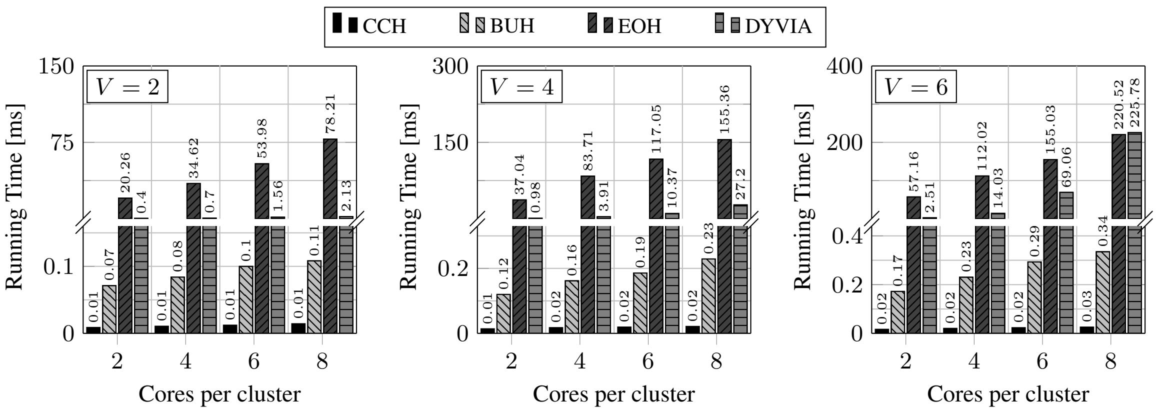


Figure 9.14: Experimental results on SCC (running at 533 MHz) of the average execution time of the evaluated task set mapping algorithms, for twelve hypothetical platforms with different V and K values.

Secondly, Figure 9.15 presents the simulation results for energy efficiency when all the four algorithms consider only one task partition obtained with LTF by using all the available cores (i.e., when $M^\star = M$). In Figure 9.15, for every different hypothetical platform, the simulation results are shown as ratios between the energy consumption of a specific heuristic and the energy consumption of DYVIA. Given that we consider 100 different cases (in regards to the tasks) for each platform, the maximum and minimum energy consumption ratios (among the 100 cases) are shown by vertical lines, while the bars represent the average energy consumption ratios.

As expected, since DYVIA derives the optimal task set assignment when using SFA on individual clusters and thus the heuristics never consume less energy than DYVIA, the simulation results in Figure 9.15 show that the minimum energy consumption ratio is never lower than 1. In the average cases, we observe that all heuristics actually behave rather well, resulting in an average energy consumption ratio for all heuristics of at most 1.0295. In regards to the maximum energy consumption ratios, Figure 9.15 shows that the maximum ratios increase as the value of K increases. This is an expected effect, given that lower K values imply that the considered platform is closer to the ideal per-core DVFS architecture, and simple heuristics have higher chances of providing reasonable solutions. Particularly, the maximum ratio can go up to 1.6507 for CCH and BUH, and up to 1.1351 for EOH, which implies that even though the heuristics can behave well for the average cases, there exist some corner cases for which DYVIA can save considerable amounts of energy.

Among all the evaluated cases, considering different number of cores for the task partitioning stage and then choosing the partition that derives the lowest energy consumption resulted in insignificant improvements, even at the cost of higher time complexity. Therefore, we omit the figures where all algorithms iterate through $M^\star = 1, 2, \ldots, M$ possible partitions, as they have no visible difference with Figure 9.15.

9.7 Summary

When using SFA to decide the DVFS levels on individual clusters, this chapter has presented several algorithms to solve the *multiple voltage island assignment* problem. Particularly, we have presented two simple

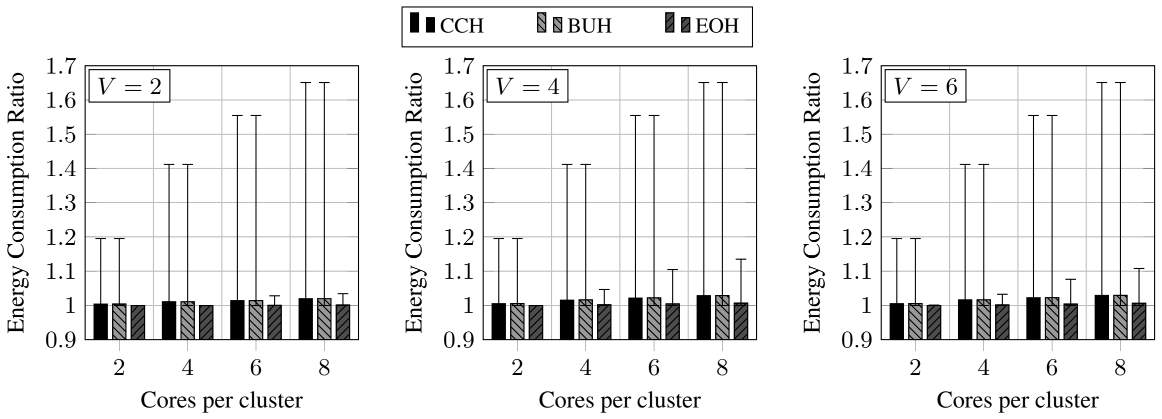


Figure 9.15: Simulation results for energy efficiency for twelve hypothetical platforms with different V and K values, where the range of the energy consumption ratios of a configuration (normalized to the energy consumption of DYVIA) is shown by the vertical line and the bar represents the average energy consumption ratio (among the 100 evaluated cases). All four algorithms partition tasks with LTF by using all the available cores (i.e., $M^{\star} = M$ for each configuration).

and intuitive mapping heuristics, and we have also theoretically analyzed the worst-case efficiency of these heuristics for energy minimization. Furthermore, we developed the DYVIA algorithm based on dynamic programming, which derives optimal task set mapping solutions in terms of energy minimization for any task partition. Based on the approximation factor of SFA for single clusters (presented in Chapter 8), we have theoretically analyzed the approximation factor of mapping task sets with DYVIA when using SFA on individual clusters, which results in the same approximation factor as that of SFA for single clusters.

We have evaluated the energy efficiency and average execution time of the heuristic algorithms and DYVIA on Intel's SCC, for which DYVIA derives solutions resulting in the minimum energy consumption among all the evaluated cases, by taking on average no longer than 225 ms to execute. Nevertheless, due to the limited available supply voltages for stable executions in SCC (as shown in Figure 9.11), and the lack of DPM capabilities to put cores in low-power modes, in the experiments the heuristics consume only up to 1.35 times more energy than DYVIA.

Additionally, we conduct further simulations for twelve hypothetical platforms with different combinations for the number of clusters and the number of cores per cluster, as well as richer DVFS and DPM features than the standard SCC. The simulation results show that for such platforms, the heuristics behave well for the average cases. However, for the worst cases, the heuristics can consume up to 1.65 times more energy than DYVIA, and this value increases with the number of cores per cluster.

In this chapter we have also proven that the total time complexity of DYVIA is exponential with respect to the number of clusters or the number of cores per cluster, and polynomial if either value is a constant (which is generally the case for commercial manycore systems). Given that DYVIA is optimal under SFA, it is the best choice to map task sets to cores and clusters, as long as its execution time is within tolerable limits. For systems with a large number of cores per cluster (for which DYVIA might require a long execution time), the simple heuristic CCH can be adopted, as it provides reasonable solutions in terms of energy consumption for the general cases with extremely low complexity.

Finally, Chapter 10 removes some of the limiting assumptions made in this chapter. Namely, Chapter 10 focuses on the same problem as this chapter, but it considers heterogeneous systems, assuming different average power consumptions for different types of tasks, and considering the raw set of tasks as an input (instead of already partitioned tasks, i.e., given tasks sets, as an input, as done in this chapter). Nevertheless, adding this additional complexity to the problem denies us the possibility of deriving optimal assignment properties (like those presented in this chapter, which are the basis for our DYVIA algorithm), and therefore Chapter 10 focuses on deriving a reasonable heuristic algorithm, rather than an optimal solution like DYVIA.

Chapter 10

Energy-Efficient Task-to-core Assignment for Heterogeneous Clustered Manycores

10.1 Overview

As discussed in Chapter 1.2.1, heterogeneous multicore and manycore systems are a promising alternative for power and energy efficiency over their homogeneous counterparts, as an application's thread/task may witness large improvements in computational performance and/or power when mapped to an appropriate type of core. Given that the cores inside a cluster have to share a common voltage at any given time point, having a system in which a cluster is composed of different types of cores is not power/energy efficient for most cases, as different types of cores may use different voltages even when executing at the same frequency. Therefore, throughout this dissertation (and in this chapter) we focus on architectures in which the cores inside a cluster are homogeneous, but different clusters can have different types and numbers of cores, i.e., heterogeneous clusters. An example of such a platform is the Exynos 5 Octa (5422) processor based on ARM's big.LITTLE architecture [21], with a simple block diagram already shown in Figure 1.6 in Chapter 1.2.1.

Because of the power and performance heterogeneity of the clusters, the power consumption and execution time of a thread/task changes not only with the DVFS settings, but also according to the task-to-cluster assignment. Moreover, given that different tasks execute different types of instructions and have different behaviors in regards to memory accesses, executing different tasks on a given core and given DVFS levels might anyway result in different average power consumptions, as already seen in Figure 1.3 in Chapter 1.1.2. Hence, task partitioning, task-to-core mapping, DVFS, and DPM play a major role in energy minimization. For periodic performance-constrained applications or real-time tasks, obtaining a task partition that results in the minimum energy consumption (i.e., the optimal task partition for energy minimization) is known to be an $\mathcal{NP}$-hard problem [3, 18], meaning that efficient but lightweight solutions are needed. Nevertheless, the state-of-the-art solutions [18, 24, 53, 65] remain inefficient in terms of energy minimization, as they fail to properly handle the existence of heterogeneous clusters.

In this chapter, we present *efficient* and *lightweight* algorithms [78] focusing on overall energy minimization for periodic real-time tasks (or performance-constrained applications) running on clustered multicore/manycore systems with heterogeneous voltage/frequency islands, in which the cores in a cluster are homogeneous and share the same voltage and frequency, but different clusters may have different types and numbers of cores and can be executed at different voltages and frequencies at any point in time. Furthermore, unlike most state-of-the-art [18,24,53,65] and our DYVIA algorithm presented in Chapter 9, in this chapter we consider that different tasks may consume different average power values, even when running on the same core and at the same DVFS levels.

The proposed techniques consist on the coordinated selection of the DVFS levels on individual clusters (particularly, using the SFA scheme described and analyzed in Chapter 8), together with a task partitioning strategy that considers the energy consumption of every task executing on different clusters and at different DVFS levels, as well as the impact of the frequency and the underlying core architecture to the resulting execution time. Every task is then mapped to the most energy-efficient cluster for the selected DVFS levels, and to a core inside the cluster such that the workloads of the cores in a cluster are balanced and all tasks meet their deadlines. In particular, for periodic real-time tasks in heterogeneous clusters:

- For the special case in which the execution cycles of all tasks scale by constant factors between different types of cores, and with the simplifying assumption that tasks consume equal power when running on the same core and at the same DVFS levels, when we have a given task partition and given DVFS levels for the clusters as inputs, we derive an optimal polynomial-time task set assignment algorithm for energy minimization under SFA.
- For the same special case as in the previous point and based on observations from its solution, we derive an efficient task partitioning and DVFS level selection algorithm for energy minimization when the task partition and DVFS levels of the clusters are not given.
- Finally, we extend the previous algorithms to consider the general case in which there is no relation between the execution cycles of a task executing in different clusters, and different tasks consume *different* average power even when running on the same core and at the same DVFS levels.

10.1.1 Motivational Example

For simplicity of presentation, consider a system with two types of cores in 22 nm technology: a high performance/power out-of-order (OOO) Alpha 21264 core, and a low performance/power *simple in-order* Alpha 21264 core (both detailed in Chapter 4). Moreover, consider an *x264* application from the PARSEC benchmark suite (as detailed in Chapter 4.3), for which every application instance runs in a single thread and needs to be computed under 6 s in this example. Intuitively, one could simply assume that in order to reduce the average power consumption and to save energy, the application should be mapped to the low power core while satisfying its timing constraint; however, this is not always the case. According to Figure 1.1 in Chapter 1.1.1 and Figure 1.3 in Chapter 1.1.2, we can observe that in order for the application to meet its deadline, it should run at least at $0.6\,\text{GHz}$ when assigned to an OOO Alpha core, finishing in $5.46\,\text{s}$ and resulting in an average power consumption of $0.31\,\text{W}$. Similarly, according to Figure 1.1 and Figure 1.3, the application should run at least at $4.0\,\text{GHz}$ if assigned to a *simple* Alpha core, finishing in $5.72\,\text{s}$ and resulting in an average power consumption of $1.27\,\text{W}$. From Figure 1.3, the corresponding energy consumption values are $1.67\,\text{J}$ for the OOO Alpha core, and $7.27\,\text{J}$ for the *simple* Alpha core. In this way, to just meet its deadline, this *x264* application consumes less energy when assigned to an OOO Alpha core than when assigned to a *simple* Alpha core, which is not necessarily intuitive. This occurs because the OOO Alpha cores perform better than the *simple* Alpha cores and thus require slower frequencies to compute the same workload in the same time, which in this case considerably reduces the average power consumption of the application.

Therefore, in order to minimize the energy of performance-constrained applications or real-time tasks, the system should take into account the power consumption of different cores running at different DVFS levels, as well as the impact of the frequency and the underlying core architecture on the resulting execution time.

Moreover, since the solutions for homogeneous clusters [24, 53, 65, 72, 107] will not necessarily derive good solutions for the general case, and since the state-of-the-art solutions for heterogeneous clusters [18,63] are not yet in a mature state, obtaining an energy-efficient algorithm for periodic real-time tasks on clustered heterogeneous multicore system with multiple voltage/frequency islands remains an open problem.

10.1.2 Problem Definition

As described in Chapter 3.2, we focus on a heterogeneous multicore/manycore system clustered in multiple voltage islands in which there are Q types of cores, V clusters, at least one cluster for each type of core (i.e., $Q \leq V$), every cluster k is composed of M_k^{cluster} cores, there are in total M cores in the chip among all types of cores (i.e., $M = \sum_{k=1}^{V} M_k^{\text{cluster}}$), the cores in cluster k are denoted as $C_{k,1}, C_{k,2}, \ldots, C_{k,M_k^{\text{cluster}}}$, and we identify the type of core of each cluster through indexes $Q_1^{\square}, Q_2^{\square}, \ldots, Q_V^{\square}$ (i.e., if cluster k is composed of cores of type q then we have that $Q_k^{\square} = q$). Moreover, with respect to DVFS we assume that all the cores inside a cluster share a common voltage and frequency, and that every core of type q has $\hat{F}_q^{\text{type}}$ available frequencies, denoted as $\left\{F_{q,1}^{\text{type}}, F_{q,2}^{\text{type}}, \ldots, F_{q,\hat{F}_q^{\text{type}}}^{\text{type}}\right\}$, such that $F_{q,\hat{F}_q^{\text{type}}}^{\text{type}}$ is the maximum frequency for a core of type q. As done everywhere else in this dissertation, for running the cores in a cluster at a desired frequency, the voltage of the cluster is implicitly set to the minimum value that supports stable execution.

As described in Chapter 3.3, the average power consumption for executing a task on a certain cores depends on the specific task under execution, on the type of core in which the task is executed, and on chosen DVFS levels. Particularly, the average power consumed on a core of type q while executing one instance of task τ_n at frequency index j (such that $0 \leq j \leq \hat{F}_q^{\text{type}}$), is denoted as $P_q^{\tau_n}\left(F_{q,j}^{\text{type}}\right)$. For example, Figure 1.3 in Chapter 1.1.2 shows average power consumption values of five applications from the PARSEC benchmark suite [4], where it can be clearly observed that different applications have different average power consumptions, even when running on the same type of core and at the same DVFS levels.

Similar to Chapter 8 and Chapter 9, when a core finishes executing all the workload available in its ready queue, it has to wait until more workload arrives. During this waiting interval, the core is not executing any task, and therefore it can enter a low-power mode (e.g., sleep, power-gated, etc.) that consumes $P_{\text{inact}_q}^{\text{type}} \geq 0$ power for a core of type q. *Note that not even the optimal solutions for energy energy minimization can optimize for $P_{\mathit{inact}_q}^{\mathit{type}}$, as this is a constant power offset for every core that is always present, which results in a constant energy consumption per time unit for the entire chip.* Given that we can transfer the power consumption $P_{\text{inact}_q}^{\text{type}}$ to some other part of the system (e.g., to the system for being active), without loss of generality and for simplicity of presentation, we can set $P_q^{\tau_n}\left(F_{q,j}^{\text{type}}\right)$ to $P_q^{\tau_n}\left(F_{q,j}^{\text{type}}\right) - P_{\text{inact}_q}^{\text{type}}$, such that we can disregard the effect of the power consumption of a core in a low-power mode. *In this way, since not even the optimal solution can optimize for $P_{\mathit{inact}_q}^{\mathit{type}}$, we only focus on the* effective optimization region *and we avoid possible masking problems in systems with large $P_{\mathit{inact}_q}^{\mathit{type}}$ values.*

Task τ_n releases an infinite number of task instances with period (and relative deadline) d_n, and every instance has worst-case execution cycles $e_{q,n}$ when being executed on a core of type q. Therefore, as described in Chapter 3.4, given that τ_n is executed $\frac{D}{d_n}$ times during one hyper-period, when executing on a core of type q running at frequency $F_{q,j}^{\text{type}}$ (such that $0 \leq j \leq \hat{F}_q^{\text{type}}$), the total energy consumed by τ_n during one hyper-period is denoted as $E_q^{\tau_n}\left(F_{q,j}^{\text{type}}\right)$, computed as already shown in Equation (3.4). Furthermore, Chapter 3.4 also discussed the existence of a *critical frequency* for every task and core type, which for task τ_n running on a core of type q is denoted as $f_{\text{crit}_q}^{\tau_n}$. Namely, for a given type of core, executing a task below its critical frequency is not energy efficient, and it should therefore be avoided when the optimization goal is minimizing energy, even if it reduces the power consumption and meets the timing constraints of the task.

$$E_q^{\tau_n}\left(F_{q,j}^{\text{type}}\right) = \frac{D}{d_n} \cdot P_q^{\tau_n}\left(F_{q,j}^{\text{type}}\right) \cdot \frac{e_{q,n}}{F_{q,j}^{\text{type}}} \tag{3.4 revisited}$$

After the tasks are partitioned and assigned to the clusters, a DVFS policy has to be adopted in order to decide the DVFS levels for the individual clusters. For such a purpose, just as done in Chapter 9, here we consider the simple and intuitive Single Frequency Approximation (SFA) scheme (presented in Chapter 8). Furthermore, the objective of this chapter is to minimize the overall energy consumption while guaranteeing that all instances of all tasks meet their deadlines (i.e., guaranteeing that all tasks meet their minimum cycle utilizations). Therefore, considering Equation (3.4) and by using SFA to decide the DVFS levels on individual clusters, the total energy consumption on cluster k during the hyper-period, defined as E_k^{cluster}, is expressed as

$$E_k^{\text{cluster}} = \frac{D}{F_{Q_k^{\square},h_k}^{\text{type}}} \cdot \sum_{i=1}^{M_k^{\text{cluster}}} \left[\sum_{\forall \tau_n \in C_{k,i}} P_{Q_k^{\square}}^{\tau_n}\left(F_{Q_k^{\square},h_k}^{\text{type}}\right) \cdot \frac{e_{Q_k^{\square},n}}{d_n} \right] \tag{10.1}$$

such that

$$F_{Q_k^{\square},h_k-1}^{\text{type}} \leq \sum_{\forall \tau_n \in C_{k,i}} \frac{e_{Q_k^{\square},n}}{d_n} \leq F_{Q_k^{\square},h_k}^{\text{type}} \qquad \begin{array}{l}\text{for all } k = 1, 2, \ldots, V \text{ and all } i = 1, 2, \ldots, M_k^{\text{cluster}} \\ \text{with } 1 \leq h_k \leq \hat{F}_{Q_k^{\square}}^{\text{type}}\end{array}$$

where indexes $h_1, h_2, \ldots, h_V$ represent the frequency indexes selected for all clusters, and where we define $F_{Q_k^{\square},0}^{\text{type}} = 0$ for all $k = 1, 2, \ldots, V$ for notational purposes. Namely, since cluster k is composed of cores of type $Q_k^{\square}$, there are $\hat{F}_{Q_k^{\square}}^{\text{type}}$ available frequencies in cluster k (reason why it should hold that $1 \leq h_k \leq \hat{F}_{Q_k^{\square}}^{\text{type}}$), the selected frequency for cluster k is $F^{\text{type}}_{Q^{\square}_k, h_k}$, and the value of $e_{Q^{\square}_k, n}$ represents the worst-case execution cycles of τ_n when being executed on cluster k. Moreover, in Equation (10.1), $\forall \tau_n \in C_{k,i}$ means "for all tasks τ_n assigned to core $C_{k,i}$", such that $\sum_{\forall \tau_n \in C_{k,i}} \frac{e_{Q^{\square}_k, n}}{d_n}$ is the total cycle utilization of core $C_{k,i}$, which should be less than or equal to the selected cluster frequency $F^{\text{type}}_{Q^{\square}_k, h_k}$ in order to guarantee that all tasks in core $C_{k,i}$ meet their timing constraints. Finally, our overall objective translates to minimizing $\sum_{k=1}^{V} E_k^{\text{cluster}}$ according to Equation (10.1).

10.2 Assignment of Given Task Sets

This section presents an assignment property when using SFA on individual clusters, which is a generalization from the property derived in Theorem 6 and Corollary 1 in Chapter 9.3.1 (the property from Chapter 9.3.1 is for the special case of homogeneous systems with the same number of cores in all clusters, e.g., Intel's SCC [36]). Furthermore, in this section we consider a few simplifying assumptions, specifically, we assume that the DVFS levels on every cluster are fixed, that the tasks are already partitioned into task sets, that the execution cycles of all tasks scale by constant factors between different types of cores, and that all tasks consume equal power when running on the same core type and at the same DVFS levels. Although these are clear limitations, the assignment property derived in this section is later used as a motivation for formulating the more general algorithms in Section 10.3 and Section 10.4.

The simplifying assumptions considered in this section lead to a new and easier notation. Particularly, having a constant scaling factor for the execution cycles in a type of core, defined as χ_q for core type q, we have that the worst-case execution cycles of τ_n executing on core type q can be computed as $e_{q,n} = \chi_q \cdot e_{y,n}$ for all core types $q = 1, 2, \ldots, Q$ and for all tasks $n = 1, 2, \ldots, R$, where $e_{y,n}$ are the worst-case execution cycles of τ_n executing on an arbitrary reference type of core y, e.g., the lowest-power cores. In other words, there is a constant ratio between the worst-case execution cycles of *any* task executed on core type q and any task executed on reference core type y, and this relation holds for all core types q by considering different scaling factors for each type of core, i.e., different χ_q for all $q = 1, 2, \ldots, Q$. Therefore, the cycle utilization of task set $\mathbf{S}_g$ running on core type q, normally computed as $w_{q,g} = \sum_{\tau_n \in \mathbf{S}_g} \frac{e_{q,n}}{d_n}$, can also be computed as $w_{q,g} = \chi_q \cdot w_{y,g}$, where the cycle utilization of task set $\mathbf{S}_g$ running on reference core type y is computed as $w_{y,g} = \sum_{\tau_n \in \mathbf{S}_g} \frac{e_{y,n}}{d_n}$. Furthermore, since in this section we are also assuming that all tasks consume *equal* power when running on the same type of core and at the same DVFS levels, we have that for core type q running at frequency index j it holds that $P_q^{\tau_1}\left(F^{\text{type}}_{q,j}\right) = P_q^{\tau_2}\left(F^{\text{type}}_{q,j}\right) = \cdots = P_q^{\tau_R}\left(F^{\text{type}}_{q,j}\right)$ and ${f_{\text{crit}}}_q^{\tau_1} = {f_{\text{crit}}}_q^{\tau_2} \cdots = {f_{\text{crit}}}_q^{\tau_R}$, such that we can simply refer to this power consumption as $P_q\left(F^{\text{type}}_{q,j}\right)$ and to this critical frequency as ${f_{\text{crit}}}_q$, i.e., ignore τ_n from this notation. Hence, from this new notation and Equation (3.4), the energy consumption of task τ_n and the energy consumption of task set $\mathbf{S}_g$ during a hyper-period running on core type q at frequency $F^{\text{type}}_{q,j}$ considering the simplifying assumptions on this section, defined as $\tilde{E}_q^{\tau_n}\left(F^{\text{type}}_{q,j}\right)$ and $\tilde{E}_q^{\mathbf{S}_g}\left(F^{\text{type}}_{q,j}\right)$, respectively, are expressed as

$$\tilde{E}_q^{\tau_n}\left(F^{\text{type}}_{q,j}\right) = D \cdot P_q\left(F^{\text{type}}_{q,j}\right) \cdot \frac{\chi_j}{F^{\text{type}}_{q,j}} \cdot \frac{e_{y,n}}{d_n} \tag{10.2}$$

and

$$\tilde{E}_q^{\mathbf{S}_g}\left(F^{\text{type}}_{q,j}\right) = D \cdot P_q\left(F^{\text{type}}_{q,j}\right) \cdot \frac{\chi_j}{F^{\text{type}}_{q,j}} \cdot w_{y,g} \tag{10.3}$$

where we denote $P_q\left(F^{\text{type}}_{q,j}\right) \frac{\chi_j}{F^{\text{type}}_{q,j}}$ as the *energy factor* of core type q running at frequency $F^{\text{type}}_{q,j}$.

According to the definition of the *energy factor*, when task set $\mathbf{S}_g$ can be mapped on two different types of cores running at different frequencies, mapping it to the core type with the smallest energy factor for the given frequencies will save more energy. This occurs because the energy factor not only considers the power consumption of the core type at the given DVFS levels, but it also considers the impact of the frequency and the underlying architecture to the execution time.

With this new notation, we can present Lemma 18, which provides an important property for assigning task sets to types of cores when the task sets are given and the DVFS levels of the clusters are known, as illustrated in Figure 10.1. Following, based on Lemma 18, Theorem 8 derives an optimal task set assignment when the DVFS levels of every type of core are known. Finally, Observation 1 summarizes the most important concept from Lemma 18 and Theorem 8, which although it might seem intuitive, it is an observation based on the formal mathematical proofs in Lemma 18 and Theorem 8.

Lemma 18 *Consider two clusters with different types of cores, specifically, a cluster with cores of type j running at frequency F^{type}_{j,z_j} (i.e., at frequency step z_j) and a cluster with cores of type g running at frequency F^{type}_{g,z_g} (i.e., at frequency step z_g). Moreover, consider task sets $\mathbf{S}_q$ and $\mathbf{S}_\ell$, such that task set $\mathbf{S}_q$ has a smaller cycle utilization than task set $\mathbf{S}_\ell$ (i.e., such that $w_{y,q} \leq w_{y,\ell}$), and both task sets can be mapped either to the cluster with cores of type j or to the cluster with cores of type g while satisfying their timing constraints (i.e., it holds that $w_{j,q} \leq w_{j,\ell} \leq F^{type}_{j,z_j}$ and $w_{g,q} \leq w_{g,\ell} \leq F^{type}_{g,z_g}$). Under SFA for the given frequencies, assigning task set $\mathbf{S}_\ell$ to the cluster with cores of type j and task set $\mathbf{S}_q$ to the cluster with cores of type g consumes (1) more or equal energy than the opposite assignment if the energy factor of the cores of type g is lower than the energy factor of the cores of type j for the given DVFS levels, i.e., if $P_j\left(F^{type}_{j,z_j}\right) \cdot \frac{\chi_j}{F^{type}_{j,z_j}} \geq P_g\left(F^{type}_{g,z_g}\right) \cdot \frac{\chi_g}{F^{type}_{g,z_g}}$, and it consumes (2) less or equal energy than the opposite assignment if the energy factor of the cores of type j is lower than the energy factor of the cores of type g for the given DVFS levels, i.e., if $P_j\left(F^{type}_{j,z_j}\right) \cdot \frac{\chi_j}{F^{type}_{j,z_j}} \leq P_g\left(F^{type}_{g,z_g}\right) \cdot \frac{\chi_g}{F^{type}_{g,z_g}}$.*

Proof. The energy for assigning task set $\mathbf{S}_\ell$ to the cluster with cores of type j and assigning task set $\mathbf{S}_q$ to the cluster with cores of type g can be computed as

$$E^{\text{original}}_{\text{assignment}} = \cdots + D \cdot P_j\left(F^{\text{type}}_{j,z_j}\right) \cdot \frac{\chi_j \cdot w_{y,\ell}}{F^{\text{type}}_{j,z_j}} + \cdots + D \cdot P_g\left(F^{\text{type}}_{g,z_g}\right) \cdot \frac{\chi_g \cdot w_{y,q}}{F^{\text{type}}_{g,z_g}} + \cdots,$$

while the energy for assigning task set $\mathbf{S}_q$ to the cluster with cores of type j and assigning task set $\mathbf{S}_\ell$ to the cluster with cores of type g can be computed as

$$E^{\text{opposite}}_{\text{assignment}} = \cdots + D \cdot P_j\left(F^{\text{type}}_{j,z_j}\right) \cdot \frac{\chi_j \cdot w_{y,q}}{F^{\text{type}}_{j,z_j}} + \cdots + D \cdot P_g\left(F^{\text{type}}_{g,z_g}\right) \cdot \frac{\chi_g \cdot w_{y,\ell}}{F^{\text{type}}_{g,z_g}} + \cdots.$$

Therefore, the difference between $E^{\text{original}}_{\text{assignment}}$ and $E^{\text{opposite}}_{\text{assignment}}$ is computed as

$$E^{\text{original}}_{\text{assignment}} - E^{\text{opposite}}_{\text{assignment}} = D\left[P_j\left(F^{\text{type}}_{j,z_j}\right) \cdot \frac{\chi_j}{F^{\text{type}}_{j,z_j}} - P_g\left(F^{\text{type}}_{g,z_g}\right) \cdot \frac{\chi_g}{F^{\text{type}}_{g,z_g}}\right](w_{y,\ell} - w_{y,q}).$$

Given that $w_{y,\ell} - w_{y,q} \geq 0$, in case that $P_j\left(F^{\text{type}}_{j,z_j}\right) \cdot \frac{\chi_j}{F^{\text{type}}_{j,z_j}} \geq P_g\left(F^{\text{type}}_{g,z_g}\right) \cdot \frac{\chi_g}{F^{\text{type}}_{g,z_g}}$, it holds that $E^{\text{original}}_{\text{assignment}} \geq E^{\text{opposite}}_{\text{assignment}}$. Similarly, in case that $P_j\left(F^{\text{type}}_{j,z_j}\right) \cdot \frac{\chi_j}{F^{\text{type}}_{j,z_j}} \leq P_g\left(F^{\text{type}}_{g,z_g}\right) \cdot \frac{\chi_g}{F^{\text{type}}_{g,z_g}}$, it holds that $E^{\text{original}}_{\text{assignment}} \leq E^{\text{opposite}}_{\text{assignment}}$. Thus, the lemma is proven. □

Theorem 8 *Suppose that, for the given DVFS levels, the types of cores can be re-ordered according to their energy factors such that $P_1\left(F^{type}_{1,z_1}\right) \cdot \frac{\chi_1}{F^{type}_{1,z_1}} \leq P_2\left(F^{type}_{2,z_2}\right) \cdot \frac{\chi_2}{F^{type}_{2,z_2}} \leq \cdots \leq P_Q\left(F^{type}_{Q,z_Q}\right) \cdot \frac{\chi_Q}{F^{type}_{Q,z_Q}}$, where z_j is the frequency index of the DVFS levels for all cores of type j for all $j = 1, 2, \ldots, Q$. An optimal solution under SFA will start with the cores of type $j = 1$, assigning to every cluster k with cores of type $j = 1$ the $M^{cluster}_k$ task sets that have the highest cycle utilizations (when running on core type $j = 1$) that are less than or equal to F^{type}_{1,z_1}, then removing the assigned task sets from the problem, and finally repeating the process with the remaining task sets for core types $j = 2, 3, \ldots, Q$.*

Proof. The proof comes directly from Lemma 18, where assigning task set $\mathbf{S}_\ell$ to a cluster with cores of type j and assigning task set $\mathbf{S}_q$ to a cluster with cores of type g is the energy-efficient alternative in case

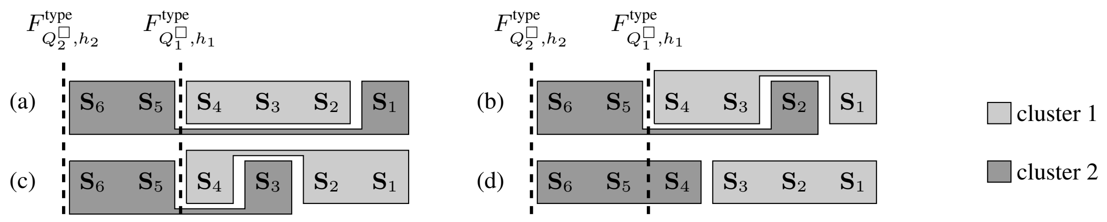

$F^{\text{type}}_{Q_2^\square,h_2}$
$F^{\text{type}}_{Q_1^\square,h_1}$
(a)
$\mathbf{S}_6$ $\mathbf{S}_5$ $\mathbf{S}_4$ $\mathbf{S}_3$ $\mathbf{S}_2$ $\mathbf{S}_1$
(b)
$\mathbf{S}_6$ $\mathbf{S}_5$ $\mathbf{S}_4$ $\mathbf{S}_3$ $\mathbf{S}_2$ $\mathbf{S}_1$
(c)
$\mathbf{S}_6$ $\mathbf{S}_5$ $\mathbf{S}_4$ $\mathbf{S}_3$ $\mathbf{S}_2$ $\mathbf{S}_1$
(d)
$\mathbf{S}_6$ $\mathbf{S}_5$ $\mathbf{S}_4$ $\mathbf{S}_3$ $\mathbf{S}_2$ $\mathbf{S}_1$
cluster 1
cluster 2

Figure 10.1: Examples of four possible task set assignments, for 2 clusters with 3 cores per cluster, in which cluster 1 runs at frequency $F^{\text{type}}_{Q_1^\square,h_1}$ and cluster 2 runs at frequency $F^{\text{type}}_{Q_2^\square,h_2}$. The task sets are increasingly ordered according to their cycle utilizations when running on reference core type y. Moreover, it holds that $w_{1,1} \leq w_{1,2} \leq w_{1,3} \leq w_{1,4} \leq F^{\text{type}}_{Q_1^\square,h_1} \leq w_{1,5} \leq w_{1,6}$ and $w_{2,1} \leq w_{2,2} \leq w_{2,3} \leq w_{2,4} \leq w_{2,5} \leq w_{2,6} \leq F^{\text{type}}_{Q_2^\square,h_2}$, such that task sets $\mathbf{S}_1$, $\mathbf{S}_2$, $\mathbf{S}_3$, and $\mathbf{S}_4$ can be mapped to either cluster, but task sets $\mathbf{S}_5$ and $\mathbf{S}_6$ can only be mapped to cluster 2. According to Lemma 18, in case that cluster 1 has a lower energy factor than cluster 2 for the given DVFS levels, then option (c) saves more energy than option (d), option (b) saves more energy than option (c), and option (a) saves more energy than option (b), such that assignment (a) is the more energy-efficient option. Similarly, in case that cluster 2 has a lower energy factor than cluster 1 for the given DVFS levels, then option (b) saves more energy than option (a), option (c) saves more energy than option (b), and option (d) saves more energy than option (c), such that assignment (d) is the more energy-efficient option.

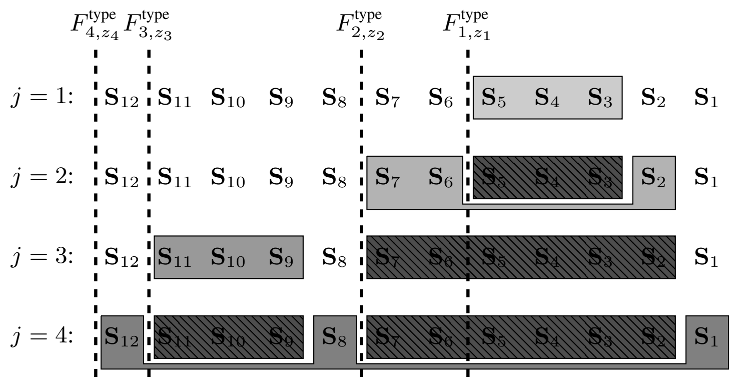
that $w_{y,q} \leq w_{y,\ell}$ and $P_j\left(F^{\text{type}}_{j,z_j}\right) \cdot \frac{\chi_j}{F^{\text{type}}_{j,z_j}} \leq P_g\left(F^{\text{type}}_{g,z_g}\right) \cdot \frac{\chi_g}{F^{\text{type}}_{g,z_g}}$. Therefore, when the DVFS levels of the types of cores are known, assigning the highest cycle utilization task sets to the clusters with cores of type j that has the lowest energy factor $P_j\left(F^{\text{type}}_{j,z_j}\right) \cdot \frac{\chi_j}{F^{\text{type}}_{j,z_j}}$ while meeting the timing constraints, consumes less or equal energy than any other assignment. Thus, the theorem is proven. □

Observation 1 *An optimal energy minimization solution for assigning given task sets to different types of cores with known DVFS levels, will assign the highest cycle utilization task sets to the types of cores that have the lowest energy factors.*

Example

Following, we present an example of the process of an algorithm based on Theorem 8 and Observation 1, for a system with $Q = 4$ types of cores, $V = 4$ clusters (i.e., one cluster for every type of core), and 3 cores inside every cluster (i.e., $M_1^{\text{cluster}} = M_2^{\text{cluster}} = M_3^{\text{cluster}} = M_4^{\text{cluster}} = 3$). For the given DVFS levels for each type of core (i.e., F^{type}_{1,z_1}, F^{type}_{2,z_2}, F^{type}_{3,z_3}, and F^{type}_{4,z_4}), the core types are increasingly ordered according to their energy factors, such that core type 1 has a lower energy factor than core type 2, core type 2 has a lower energy factor than core type 3, and core type 3 has a lower energy factor than core type 4. Furthermore, for the given DVFS levels, it holds that $w_{1,1} \leq w_{1,2} \leq \cdots \leq w_{1,5} \leq F^{\text{type}}_{1,z_1} \leq w_{1,6} \leq \cdots \leq w_{1,12}$, that $w_{2,1} \leq w_{2,2} \leq \cdots \leq w_{2,7} \leq F^{\text{type}}_{2,z_2} \leq w_{2,8} \leq \cdots \leq w_{2,12}$, that $w_{3,1} \leq w_{3,2} \leq \cdots \leq w_{3,11} \leq F^{\text{type}}_{3,z_3} \leq w_{3,12}$, and that $w_{4,1} \leq w_{4,2} \leq \cdots \leq w_{4,12} \leq F^{\text{type}}_{4,z_4}$, such that task sets $\mathbf{S}_1, \mathbf{S}_2, \ldots, \mathbf{S}_5$ can be mapped to any cluster, task sets $\mathbf{S}_6, \mathbf{S}_7, \ldots, \mathbf{S}_{12}$ can be mapped to any cluster except cluster 1, task sets $\mathbf{S}_8, \mathbf{S}_9, \ldots, \mathbf{S}_{12}$ can only be mapped to cluster 3 or cluster 4, and task set $\mathbf{S}_{12}$ can only be mapped to cluster 4. For such a case, when using SFA to decide the DVFS levels on individual clusters, the optimal assignment of the task sets onto cores belonging to different clusters is shown in Figure 10.2, based on Theorem 8 and Observation 1. Namely, we first focus on the cores of type $j = 1$ with the lowest energy factor, and assign M_1^{cluster} task sets with cycle utilizations (when executed on a core of type $j = 1$) less than or equal to F^{type}_{1,z_1} onto cores in cluster 1. Then, in this example, task sets $\mathbf{S}_3$, $\mathbf{S}_4$, and $\mathbf{S}_5$ are assigned onto cores belonging to cluster 1. Furthermore, task sets $\mathbf{S}_3$, $\mathbf{S}_4$, and $\mathbf{S}_5$ are now removed from the problem (colored in dark-gray and crossed out in the figure when $j > 1$). Continuing, we focus on cluster 2 (i.e., $j = 2$ in the figure), which is the cluster with the second lowest energy factor core types, assigning M_2^{cluster} task sets with cycle utilizations (when executed on a core of type $j = 2$) less than or equal to F^{type}_{2,z_2}, specifically, task sets $\mathbf{S}_2$, $\mathbf{S}_6$, and $\mathbf{S}_7$, onto cores belonging to cluster

2. These task sets are removed from the problem (colored in dark-gray and crossed out in the figure $j > 2$) and the process is repeated for $j = 3$ and $j = 4$ until all task sets are mapped.



F^{type}_{4,z_4}
F^{type}_{3,z_3}
F^{type}_{2,z_2}
F^{type}_{1,z_1}
$j = 1$:
$j = 2$:
$j = 3$:
$j = 4$:
S_{12} S_{11} S_{10} S_9 S_8 S_7 S_6 S_5 S_4 S_3 S_2 S_1

Figure 10.2: Example of the process of an algorithm based on Theorem 8 and Observation 1, for a system with 4 types of cores and 3 cores per cluster. For the given DVFS levels, the clusters are increasingly ordered according to their energy factors. Every value of j represents the core types j in Theorem 8. The task sets assigned to cores of type j are colored in dark-gray and crossed out for other core types with larger energy factors.

10.3 Special Case: Tasks Consume Equal Average Power on a Given Core and Voltage/Frequency

This section presents a polynomial-time algorithm to solve our problem, considering the same simplifying assumptions as done in Section 10.2, but without considering that the tasks are already partitioned into task sets. First, in Section 10.3.1, assuming that the DVFS levels for every type of core is (preliminary) fixed, we derive a task partitioning algorithm motivated by Observation 1 and the Largest Task First (LTF) strategy (described in detail in Chapter 8). Secondly, in Section 10.3.2, we propose an algorithm that evaluates a finite number of DVFS configurations for all the clusters, and chooses the most energy-efficient DVFS configuration after applying the task partitioning strategy from Section 10.3.1. These algorithms are later extended in Section 10.4 in order to consider the more general case in which different tasks consume *different* power even when running on the same core and at the same DVFS levels, and there is no constant factor for scaling the execution cycles among different types of cores.

10.3.1 Special Task Partitioning for Fixed Frequencies

This section presents a polynomial-time task partitioning algorithm, which we call Fixed Frequency Island-Aware Largest Task First (FFI-LTF). The FFI-LTF algorithm is motivated by Observation 1 and the LTF strategy for homogeneous cores. Although Observation 1 refers to task sets instead of tasks, the same basic idea still applies when partitioning tasks. Namely, when the types of core have (preliminary) fixed DVFS levels, in order to minimize the overall energy consumption, *a good strategy is to attempt to fully utilize every cluster increasingly with regards to the energy factors of the corresponding types of cores*. The reason for this is because a high cycle utilization task consumes more power in a high energy factor core type than it does in a low energy factor core type. In this way, assigning high cycle utilization tasks to clusters with low energy factor core types saves energy. Contrarily, assigning low cycle utilization tasks to clusters with high energy factor core types (in case the clusters with low energy factor core types are already filled with high cycle utilization tasks) has a much smaller impact on the absolute energy consumption values.

Given that obtaining the optimal task partition of periodic real-time tasks for energy minimization is an $\mathcal{NP}$-hard problem [3, 18], we know that there is no polynomial-time algorithm that can perfectly fully utilize any given cluster unless $\mathcal{P} = \mathcal{NP}$. Hence, we apply a Worst-Fit-Decreasing heuristic like LTF to assign a task onto a core, but choosing the types of cores increasingly with respect to their energy factors. A pseudo-code for FFI-LTF is presented in Algorithm 10, where we assume that the (preliminary) fixed DVFS levels and the tasks (ordered according to their cycle utilizations when running on a reference core type y) are given as inputs. Particularly, FFI-LTF orders the types of cores increasingly according to their energy factors for the (preliminary) fixed DVFS levels (line 1). Then, starting from the task with the largest cycle utilization on core type y (i.e., for all $n = R, R-1, \ldots, 1$, in line 2), algorithm FFI-LTF attempts to assign every task τ_n to

Algorithm 10 Fixed Frequency Island-Aware Largest Task First (FFI-LTF)

```
Input: Tasks $\{\tau_1, \ldots, \tau_R\}$ ordered according to their cycle utilizations when running on reference core type
    $y$ such that $\frac{e_{y,1}}{d_1} \leq \cdots \leq \frac{e_{y,R}}{d_R}$, and given frequencies $\{F^{\text{type}}_{1,z_1}, F^{\text{type}}_{2,z_2}, \ldots F^{\text{type}}_{Q,z_Q}\}$ for every type of core;
Output: Task-to-core assignment of all tasks, and the adjusted DVFS levels of all clusters;
1: Sort the types of cores increasingly according to their energy factors, such that $P_1\left(F^{\text{type}}_{1,z_1}\right) \cdot \frac{\chi_1}{F^{\text{type}}_{1,z_1}} \leq$
   $P_2\left(F^{\text{type}}_{2,z_2}\right) \cdot \frac{\chi_2}{F^{\text{type}}_{2,z_2}} \leq \cdots \leq P_Q\left(F^{\text{type}}_{Q,z_Q}\right) \cdot \frac{\chi_Q}{F^{\text{type}}_{Q,z_Q}}$;
2: for all $n = R, R-1, \ldots, 1$ (i.e., for all tasks, decreasingly according to their cycle utilizations) do
3:   for all $q = 1, 2, \ldots, Q$ (i.e., for all core types, increasingly according to their energy factors) do
4:     Find core $i$ in cluster $k$ (i.e., $C_{k,i}$), with the smallest cycle utilization among all cores of type $q$. If
       two or more clusters with cores of type $q$ have a core with equal smallest cycle utilization, cluster $k$
       is chosen as the cluster with the smallest highest cycle utilization among all cores in the cluster;
5:     if task $\tau_n$ fits in core $C_{k,i}$ without exceeding $F^{\text{type}}_{q,z_q}$ (i.e., if $\tau_n$ can be feasibly mapped to $C_{k,i}$) then
6:       $C_{k,i} \leftarrow \tau_n$; {Assign task $\tau_n$ to core $C_{k,i}$}
7:       break for loop $q$; {After a task is assigned to a core, we continue with the next task}
8:     end if
9:   end for
10: end for
11: According to the final task partitioning, reduce the DVFS levels of individual clusters as much as possible
    while meeting the timing constraints, but not below the corresponding critical frequencies;
12: return Task-to-core assignment of all tasks, and the adjusted DVFS levels of all clusters;
```

the least loaded core of the cluster with the lowest energy factor core type, such that all tasks remain feasible (i.e., the total cycle utilization of every core is no larger than the execution frequency of its cluster). Namely, the first loop (between lines 2 and 10) iterates through all R tasks, decreasingly with respect to their cycle utilizations when running on core type y. The second loop (between lines 3 and 9) iterates through all types of cores, increasingly with respect to their energy factors (i.e., $q = 1, 2, \ldots, Q$ in line 3). Following (line 4), the algorithm iterates through all clusters with cores of type q, and selects indexes i and k such that core i in cluster k (i.e., $C_{k,i}$), has the smallest cycle utilization among all cores of type q. In case that task τ_n fits inside core $C_{k,i}$ without exceeding F^{type}_{q,z_q} (line 5), i.e., if we can assign τ_n to $C_{k,i}$ while satisfying the timing constraints of all tasks mapped to core $C_{k,i}$, then we assign task τ_n to core $C_{k,i}$ (line 6) and break the second loop iterating through all core types (line 7), i.e., after a task is assigned to a core we move to the next task. In case that task τ_n does not fit inside core $C_{k,i}$, then the algorithm tests the next core type q. Namely, when a task does not fit on the least loaded core of the cluster with the lowest energy factor core type, FFI-LTF attempts to assign the task on the least loaded core of a cluster with the second lowest energy factor core type, and so on. When two or more clusters with the same core type have a core with equal smallest cycle utilization, the task is assigned to the cluster with the smallest highest cycle utilization among all cores in the cluster (i.e., to the cluster which can potentially be executed at the slowest frequency).

In regards to time complexity, ordering the types of cores increasingly according to their energy factors for the (preliminary) fixed DVFS levels (line 1) has time complexity $O(Q \cdot \log Q)$. Furthermore, in case we do not use any special structure to keep the cores in all cluster ordered, for every task τ_n, the worst-case time complexity for finding core $C_{k,i}$ (in line 4) is $O(M)$, such that the total time complexity of FFI-LTF is $O(Q \cdot \log Q + R \cdot M)$. Contrarily, in case we use a binary heap to keep the cores in a cluster ordered according to their cycle utilizations, for every task τ_n, the time complexity for finding core $C_{k,i}$ (in line 4) is at most $O(V)$ for searching in all clusters, and the time complexity for updating the binary heap of cluster k when assigning task τ_n to core $C_{k,i}$ (in line 6) is $O\left(\log M^{\text{cluster}}_k\right)$. In order to consider the worst case, we define $M^{\text{cluster}}_{\max}$ as the maximum number of cores in any cluster, i.e., $M^{\text{cluster}}_{\max} = \max_{1 \leq k \leq V}\left\{M^{\text{cluster}}_k\right\}$, resulting in total time complexity $O\left(Q \cdot \log Q + R \cdot V + R \cdot \log M^{\text{cluster}}_{\max}\right)$ for algorithm FFI-LTF.

Before FFI-LTF finishes, whenever possible, the (preliminary) fixed DVFS levels can be further reduced individually for every cluster (line 11 in Algorithm 10) while still guaranteeing that all tasks meet their timing constraints. For example, consider a cluster with available frequencies $\{0.1\,\text{GHz}, 0.2\,\text{GHz}, \ldots, 2.0\,\text{GHz}\}$, with a (preliminary) selected frequency of $1.0\,\text{GHz}$. For such a cluster, in case that the critical frequency of the cluster is smaller than or equal to $0.8\,\mathrm{GHz}$ and if the maximum cycle utilization among all cores in the cluster after the task-to-core assignment is $0.75\,\mathrm{GHz}$, the frequency of the cluster could be further reduced to $0.8\,\mathrm{GHz}$. In this way, all the tasks assigned to the cluster can still meet their deadlines while the energy consumption of the cluster is reduced. This is the reason why throughout this section we state that before running FFI-LTF the DVFS levels of the different types of cores are *(preliminary) fixed*, instead of simply stating that they are *fixed*.

Algorithm FFI-LTF requires a preliminary DVFS level selection for the different types of cores in order to order the core types according to their energy factors, which depends on their DVFS levels. However, after all the tasks are assigned to cores, this preliminary DVFS levels can be adjusted to further reduce the overall energy consumption.

10.3.2 Potential DVFS Configurations for all Clusters in the Special Case

By using algorithm FFI-LTF described in Section 10.3.1, we can now map the tasks to cores such that the overall energy consumption is minimized when the preliminary DVFS levels of the core types and clusters are known. Nevertheless, different preliminary DVFS configurations for the core types/clusters may result in very different energy consumptions. In order to find out which is the most (or at least a reasonably good) energy-efficient DVFS configuration, more than one configuration alternative needs to be tested. A naive algorithm would, e.g., simply apply brute-force and test all possible DVFS configurations, resulting in exponential time complexity. Contrarily, in this section we present an algorithm, which we call Heterogeneous Island-Aware Largest Task First (HI-LTF), that only tests a limited number of DVFS configurations, executes FFI-LTF, and then chooses the configuration with the lowest energy consumption.

To illustrate how we can constraint the number of configurations to test, Figure 10.3 shows an abstract example of the energy factors of three different core types running at different DVFS levels. As an initial point, it is clear that we can disregard all the frequencies that are below the critical frequency of each type of core, since they would not be energy-efficient choices (dashed in Figure 10.3). Now, consider two different preliminary DVFS configurations for the different types of cores, specifically, configuration $[A2, B5, C4]$ and configuration $[A5, B5, C4]$. From Figure 10.3, we can observe that Core A is the most energy-efficient core (i.e., the core type with the lowest energy factor) under both configurations, and it will hence be the first choice for task assignment when executing FFI-LTF for both cases. However, given that frequency $A2$ is smaller than frequency $A5$ and thus less tasks can be feasibly mapped to Core A when running at frequency $A2$ than when running at frequency $A5$, after executing FFI-LTF for both configurations, Core A will generally have less tasks and a smaller cycle utilization under configuration $[A2, B5, C4]$ than it will under configuration $[A5, B5, C4]$, resulting in a higher energy consumption for Core A in the second case. Nevertheless, since the total number of tasks to be executed remains constant, more tasks will be assigned to Core B and Core C in configuration $[A2, B5, C4]$ than in configuration $[A5, B5, C4]$. Therefore, with respect to the overall energy consumption, configuration $[A2, B5, C4]$ is probably less energy efficient than configuration $[A5, B5, C4]$, as the latter configuration assigns *more tasks* to a core type with a *low energy factor* ($A5$ in this example) and *less tasks* to core types with *high energy factors* ($B5$ and $C4$ in this example) than the former configuration.

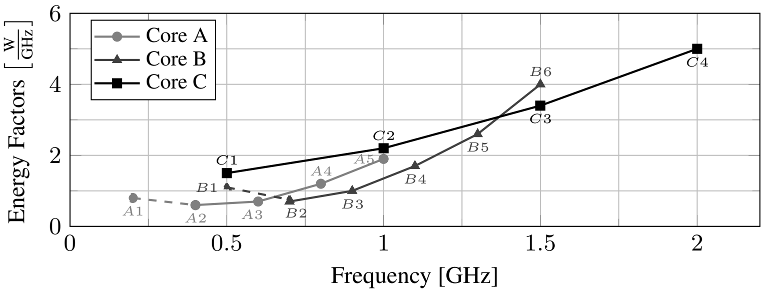


Figure 10.3: Abstract example of the energy factors of three hypothetical core types running at different DVFS levels. The critical frequencies of Core A and Core B are $A2$ and $B2$, respectively, hence running at frequencies $A1$ or $B1$ is not energy efficient.

Similarly, if we consider configurations $[A5, B5, C4]$ and $[A5, B6, C4]$, the same concepts apply and the latter configuration will generally be a more energy-efficient alternative. Particularly, since Core A runs at frequency $A5$ in both configurations and it is the core type with the lowest energy factor in both cases, it will be the first choice for task assignment when executing FFI-LTF, such that the same tasks will be assigned to Core A in both configurations. Nevertheless, less tasks will generally be assigned to Core B under configuration $[A5, B5, C4]$ than under configuration $[A5, B6, C4]$, such that Core B generally consumes more energy in the second case than in the first case, while the opposite occurs for Core C. Therefore, given that the energy factor for $B6$ remains smaller than that of $C4$, in regards to the overall energy consumption, then configuration $[A5, B5, C4]$ is probably less energy efficient than configuration $[A5, B6, C4]$. Furthermore, as explained at the end of Section 10.3.1, it even may occur that, after executing FFI-LTF for configuration $[A5, B6, C4]$, the frequency of Core B is later set to $B5$ if its total cycle utilization allows it, which implies that considering configuration $[A5, B5, C4]$ is redundant.

Based on these two examples, we can express Observation 2, from which we can formulate an energy-efficient algorithm, called HI-LTF, that *only tests (at most) one DVFS configuration for every possible order of the energy factors of the core types*. A pseudo-code for HI-LTF is presented in Algorithm 11.

Observation 2 *When selecting the DVFS configurations for executing algorithm FFI-LTF, the order of the energy factors of the types of cores is generally more important than the specific DVFS level selection. Hence, for a given order of the energy factors, only the configuration with the highest DVFS levels that satisfies the given order can be considered, and other configurations with the same energy factor order are generally redundant.*

Algorithm 11 Heterogeneous Island-Aware Largest Task First (HI-LTF)

```
Input: Tasks $\{\tau_1, \tau_2, \dots, \tau_R\}$;
Output: Task-to-core assignment of all tasks, and the DVFS levels of all clusters;
1: Sort the tasks increasingly according to their cycle utilizations when running on reference core type $y$,
   such that $\frac{e_{y,1}}{d_1} \leq \frac{e_{y,2}}{d_2} \leq \dots \leq \frac{e_{y,R}}{d_R}$;
2: for all possible energy factor orders for the types of cores do
3:     $\{F^{\text{type}}_{1,z_1}, F^{\text{type}}_{2,z_2}, \dots F^{\text{type}}_{Q,z_Q}\} \leftarrow$ Select the highest frequencies for every type of core such that the order
       under consideration in this iteration is satisfied;
4:     Execute FFI-LTF (Algorithm 10). In case that all tasks could be feasible mapped to cores for the given
       frequencies, store the results of FFI-LTF and the expected overall energy consumption for this case;
5: end for
6: return Task-to-core assignment and DVFS levels that results in the lowest overall energy consumption;
```

In regards to the time complexity of HI-LTF, the time complexity for ordering the tasks (line 1) is $O(R \cdot \log R)$, and the time complexity for executing FFI-LTF (line 4) is $O\left(Q \cdot \log Q + R \cdot V + R \cdot \log M^{\text{cluster}}_{\text{max}}\right)$. Given that FFI-LTF is executed at most $Q!$ times (the maximum number of combinations for the possible energy factor orders for the types of cores, in line 2), the total worst-case time complexity of algorithm HI-LTF is $O\left(R \cdot \log R + Q! \left(Q \cdot \log Q + R \cdot V + R \cdot \log M^{\text{cluster}}_{\text{max}}\right)\right)$. *Note that for practical systems, although a chip might have many clusters, the core heterogeneity is normally limited, i.e., Q is generally a small number, and thus the time complexity of HI-LTF will not grow on systems with a large number of cores.* For the example in Figure 10.3, instead of having a brute-force algorithm that tests all 120 possible DVFS configurations, algorithm HI-LTF will only test 6 configurations, particularly (increasingly according to the energy factors): $[A5, B6, C4]$, $[B4, A5, C4]$, $[A5, C3, B6]$, $[C1, A5, B6]$, $[B3, C1, A5]$, and $[C1, B4, A5]$. Note that we are not claiming that an algorithm based on Observation 2, e.g., HI-LTF, will derive an optimal solution. In fact, there might exist some corner cases in which a DVFS configuration not tested by HI-LTF might result in a smaller energy consumption after running FFI-LTF, e.g., due to an imperfect task partitioning. Nevertheless, *algorithm HI-LTF will derive energy-efficient solutions for the general cases.*

10.4 General Case: Different Tasks Consume Different Average Power on the Same Core

This section extends the algorithms presented in Section 10.3, to consider the general case in which there is no relation in the execution cycles of a task when executed on different types of cores, and in which different tasks consume *different* power even when running on the same core and at the same DVFS levels.

10.4.1 General Task Partitioning for Fixed Frequencies

In the general task model, the energy factor of a type of core changes from one task to another, and therefore it is no longer possible to order the core types with respect to their energy factors for all tasks, as done by algorithm FFI-LTF. Thus, the extended algorithm, called Fixed Frequency Island- and Task-Aware Largest Task First (FIT-LTF), assigns every task to the least loaded core of the cluster with the type of core that has the lowest energy factor for the corresponding task, such that all tasks can meet their timing constraints. In the general case, the *energy factor* of task τ_n, executed on a core of type q running at frequency $F^{\text{type}}_{q,j}$, is computed as $P^{\tau_n}_q \left(F^{\text{type}}_{q,j}\right) \cdot \frac{e_{q,n}}{F^{\text{type}}_{q,j}}$, which also corresponds to the energy consumption of one task instance for these settings. A pseudo-code for FIT-LTF is presented in Algorithm 12, which has some resemblance to Algorithm 10, where the main difference is that in FIT-LTF there is no global order of the core types with respect to their energy factors. Therefore, the second loop (lines 3 to 10) is not interrupted once a task can be mapped to a core, but the algorithm rather tests all Q core types and all clusters of every type of core, with time complexity $O(V)$.

Algorithm 12 Fixed Frequency Island- and Task-Aware Largest Task First (FIT-LTF)

```
Input: Tasks $\{\tau_1, \ldots, \tau_R\}$ ordered according to their cycle utilizations when running on reference core type
    $y$ such that $\frac{e_{y,1}}{d_1} \leq \cdots \leq \frac{e_{y,R}}{d_R}$, and given frequencies $\{F^{\text{type}}_{1,z_1}, F^{\text{type}}_{2,z_2}, \ldots F^{\text{type}}_{Q,z_Q}\}$ for every type of core;
Output: Task-to-core assignment of all tasks, and the adjusted DVFS levels of all clusters;
1:  for all $n = R, R-1, \ldots, 1$ (i.e., for all tasks, decreasingly according to their cycle utilizations) do
2:    $g \leftarrow 0$; $y \leftarrow 0$; {Initialize the cluster and core indexes, such that task $\tau_n$ is initially unassigned}
3:    for all $q = 1, 2, \ldots, Q$ (i.e., for all core types, increasingly according to their energy factors) do
4:      Find core $i$ in cluster $k$ (i.e., $C_{k,i}$), with the smallest cycle utilization among all cores of type $q$. If
        two or more clusters with cores of type $q$ have a core with equal smallest cycle utilization, cluster $k$
        is chosen as the cluster with the smallest highest cycle utilization among all cores in the cluster;
5:      if task $\tau_n$ fits in core $C_{k,i}$ without exceeding $F^{\text{type}}_{q,z_q}$ (i.e., if $\tau_n$ can be feasibly mapped to $C_{k,i}$) then
6:        if $g = 0$ or $P^{\tau_n}_q \left(F^{\text{type}}_{q,z_q}\right) \cdot \frac{e_{q,n}}{F^{\text{type}}_{q,z_q}} < P^{\tau_n}_{Q^{\square}_g} \left(F^{\text{type}}_{Q^{\square}_g,h_g}\right) \cdot \frac{e_{Q^{\square}_g,n}}{F^{\text{type}}_{Q^{\square}_g,h_g}}$ then
7:          $g \leftarrow k$; $y \leftarrow i$; {Store the indexes of the cluster and core that minimize the energy consumption
            of task $\tau_n$ for the given DVFS levels}
8:        end if
9:      end if
10:   end for
11:   if $g > 0$ then
12:     $C_{g,y} \leftarrow \tau_n$; {If we found a feasible mapping (i.e., if $g > 0$), then assign task $\tau_n$ to core $C_{g,y}$}
13:   end if
14: end for
15: According to the final task partitioning, reduce the DVFS levels of individual clusters as much as possible
    while meeting the timing constraints, but not below the corresponding critical frequencies;
16: return Task-to-core assignment of all tasks, and the adjusted DVFS levels of all clusters;
```

After finding the core of type q with smallest cycle utilization (line 4), i.e., $C_{k,i}$, if task τ_n can be feasibly assigned onto core $C_{k,i}$ (line 5), then indexes g and y keep track of the most energy-efficient cluster and core, respectively, in which to assign task τ_n (lines 6 to 8). In case there exist a feasible assignment (i.e., if $g > 0$ in line 11), then task τ_n is assigned to core $C_{g,y}$ (line 12), with time complexity $O\left(\log M^{\text{cluster}}_g\right)$ for this cluster, or $O\left(\log M_{\max}^{\text{cluster}}\right)$ in the worst-case among all clusters. Finally, the total worst-case time complexity of algorithm FIT-LTF is $O\left(R \cdot V + R \cdot \log M_{\max}^{\text{cluster}}\right)$.

10.4.2 Potential DVFS Configurations for all Clusters in the General Case

Given that, in the general task model, the energy factor of a type of core changes from one task to another, therefore the reasoning of Observation 2 (which supports algorithm HI-LTF) no longer applies. Nevertheless, although algorithm FIT-LTF (presented in Section 10.4.1) considers the general task model, it still requires (preliminary) fixed DVFS levels for the different types of cores as an input. Again, a naive algorithm would, e.g., simply apply brute-force and test all possible DVFS configurations. Another alternative to deal with this issue is to compute the *average* energy factor among all tasks for every available DVFS level in every type of core, and then use these average energy factors to test a limited number of preliminary DVFS configurations, similar to algorithm HI-LTF, but now executing algorithm FIT-LTF for the task-to-core assignment, thus considering the general task model. *Namely, considering the average energy factors is only needed to select the (preliminary) fixed DVFS levels and thus reduce the time complexity; however, the task-to-core assignment has no restrictions and can focus on the general model through algorithm FIT-LTF.* Therefore, there are only two differences between algorithm HI-LTF and its extension, called Heterogeneous Island- and Task-Aware Largest Task First (HIT-LTF), for which we present the pseudo-code in Algorithm 13. The first difference is that HIT-LTF needs to obtain the average energy factors among all tasks for every available DVFS level in every type of core, which are computed once at the start of the algorithm (line 2 in Algorithm 13) with time complexity $R \cdot \sum_{q=1}^{Q} \hat{F}_q^{\text{type}}$. The second difference is that HIT-LTF executes algorithm FIT-LTF (line 5 in Algorithm 13) instead of executing algorithm FFI-LTF (line 4 in Algorithm 11). With these two considerations, the resulting total worst-case time complexity of algorithm HIT-LTF is $O\left(R \cdot \log R + R \cdot \sum_{q=1}^{Q} \hat{F}_q^{\text{type}} + Q! \left(R \cdot V + R \cdot \log M_{\max}^{\text{cluster}}\right)\right)$. The remarks made for algorithm HI-LTF in regards to systems with large number of cores also applies to HIT-LTF. Particularly, *although a chip might have many clusters, the core heterogeneity is normally limited (i.e., Q is generally a small number), and thus the time complexity of HIT-LTF will not grow on systems with a large number of cores.*

Algorithm 13 Heterogeneous Island- and Task-Aware Largest Task First (HIT-LTF)

```
Input: Tasks ${\tau_1, \tau_2, \ldots, \tau_R}$;
Output: Task-to-core assignment of all tasks, and the DVFS levels of all clusters;
1: Sort all tasks increasingly according to their cycle utilizations when running on reference core type $y$,
   such that $\frac{e_{y,1}}{d_1} \leq \frac{e_{y,2}}{d_2} \leq \cdots \leq \frac{e_{y,R}}{d_R}$;
2: For all types of cores ($Q$ in total) and for all available DVFS levels of each type of core ($\hat{F}_q^{\text{type}}$ available
   DVFS levels for core type $q$), compute the average energy factor among all $R$ tasks;
3: for all possible average energy factor orders for the types of cores do
4:     ${F_{1,z_1}^{\text{type}}, F_{2,z_2}^{\text{type}}, \ldots F_{Q,z_Q}^{\text{type}}} \leftarrow$ Select the highest frequencies for every type of core such that the average
       energy factor order under consideration in this iteration is satisfied;
5:     Execute FIT-LTF (Algorithm 12). In case that all tasks could be feasible mapped to cores for the given
       frequencies, store the results of FIT-LTF and the expected overall energy consumption for this case;
6: end for
7: return Task-to-core assignment and DVFS levels that results in the lowest overall energy consumption;
```

10.5 Experimental Evaluations

This section presents experimental evaluations that compare our HIT-LTF algorithm against the intuitive Low-Power First (LPF) scheme and against the Equally-Worst-Fit-Decreasing (EWFD) scheme [18].

The LPF scheme is a simple and intuitive scheme that iteratively assigns tasks to cores according to the LTF strategy, by focusing on different types of cores increasingly according to their average power consumption, increasing the frequencies of the clusters as much as necessary for the tasks to meet their timing constraints. Particularly, LPF assigns tasks to cores decreasingly according to their cycle utilizations when executing on reference core type y, starting with the clusters composed by types of cores that have the lowest average power consumption, assigning tasks to the least loaded core among all clusters with this type of core. At the beginning, the DVFS levels of these low-power clusters are set to their lowest values. When a new task cannot fit inside any low-power cluster without violating its timing constraints, LPF increases the DVFS levels of the clusters, one cluster at a time. When the low-power clusters are all already set to run at their maximal DVFS levels and a task cannot fit in it without violating the timing constraints, then the task is assigned to a cluster composed by cores with the second lowest average power consumption, and so on.

Contrarily, the EWFD scheme attempts to balance the total cycle utilization in every cluster. In EWFD the clusters are arbitrarily ordered (e.g., increasingly according to the average power consumption of core types of different clusters, similar to LPF), and tasks are again ordered decreasingly according to their cycle utilizations when executing on reference core type y. According to this cluster and task order, EWFD first chooses the lowest index cluster that can fit the largest unmapped task, and assigns the task to a core in this cluster according to LTF. The process is then repeated for every task, while the clusters are circularly iterated according to a Next-Fit scheme, i.e., if the last task was assigned to cluster i then EWFD tries to fit the next task starting with cluster $i + 1$.

10.5.1 Setup

For our experimental evaluations, we use the simulation framework described in Chapter 4 in high-level mode. In our experiments we consider 4 different platforms composed by different combinations of 4 different types of clusters, where each type of cluster is composed by a certain type of core. The first type of cluster has 8 OOO Alpha 21264 cores, while the second type of cluster has 8 *simple in-order* Alpha 21264 cores, both based on simulations on gem5 [5] and McPAT [57] for 22 nm technology. The other two types of clusters are based on real measurements (particularly, average execution time and energy consumption of several application instances executing at different DVFS levels) on an Odroid-XU3 [26] mobile platform with an Exynos 5 Octa (5422) [92] chip with ARM's big.LITTLE architecture. Specifically, the third type of cluster has 4 *in-order* Cortex-A7 cores, and the fourth type of cluster has 4 OOO Cortex-A15 cores. Additional details about these cores and clusters were already described in Chapter 4.2.2. With these 4 different types of clusters, we compose the 4 different platforms considered in our experiments, particularly, (a) a platform with one Cortex-A7 cluster and one Cortex-A15 cluster, just like the Exynos processor; (b) a platform with one OOO Alpha cluster and one *simple* Alpha cluster; (c) a platform with one Cortex-A7 cluster, one Cortex-A15 cluster, one OOO Alpha cluster, and one *simple* Alpha cluster; and (d) a platform with four OOO Alpha clusters, and four *simple* Alpha clusters.

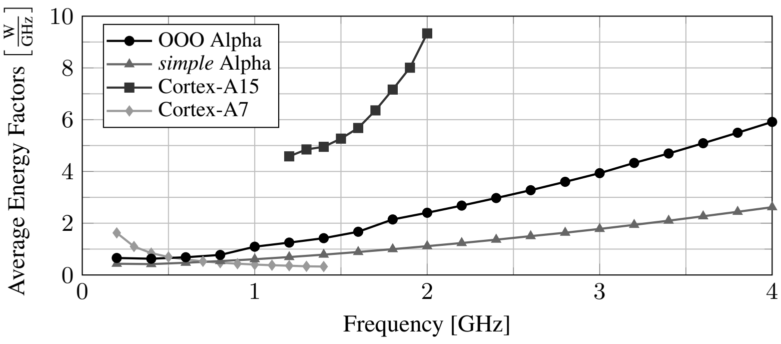
For benchmarks, we consider five representative applications (with different power consumptions and execution times) from the PARSEC benchmark suite [4] (described in Chapter 4.3), specifically, *blackscholes*, *bodytrack*, *ferret*, *swaptions*, and *x264*. In order to have real-time tasks, we assume that every task consists of one randomly-selected PARSEC application running a single thread (i.e., using a uniform distribution, we randomly choose which specific application, among the five representative applications, is executed by each task), to which we randomly assign a deadline and period (equal to the deadline), such that every application instance must finish before its deadline and a new task instance is periodically started. The deadline/period for every task is randomly chosen between 2 s and 20 s, in steps of 2 s according to a uniform distribution, such that the hyper-period D is at most 5040 s.

10.5.2 Results

As stated in Section 10.4.2, for every type of core and for all available DVFS levels on each type of core, our HIT-LTF algorithm needs to compute the average energy factors among all tasks for each available frequency in every type of core. For the given types of cores and benchmarks detailed in Section 10.5.1, the corresponding average energy factors are presented in Figure 10.4.

We now conduct the experiments to compare the overall energy consumption achieved by each one of the evaluated algorithms. For each one of the 4 different platforms under consideration, we conduct 10^4 experiments, where each experimental scenario considers a different set of R realistic PARSEC applications converted to real-time tasks randomly, as explained in Section 10.5.1. The total number of tasks (i.e., R)

Figure 10.4: Average energy factors of five representative applications from the PARSEC benchmark suite, based on simulations conducted in gem5 and McPAT (for both types of Alpha cores), and based on measurements on the Exynos 5 Octa (5422) processor (for the Cortex-A7 and Cortex-A15 cores).



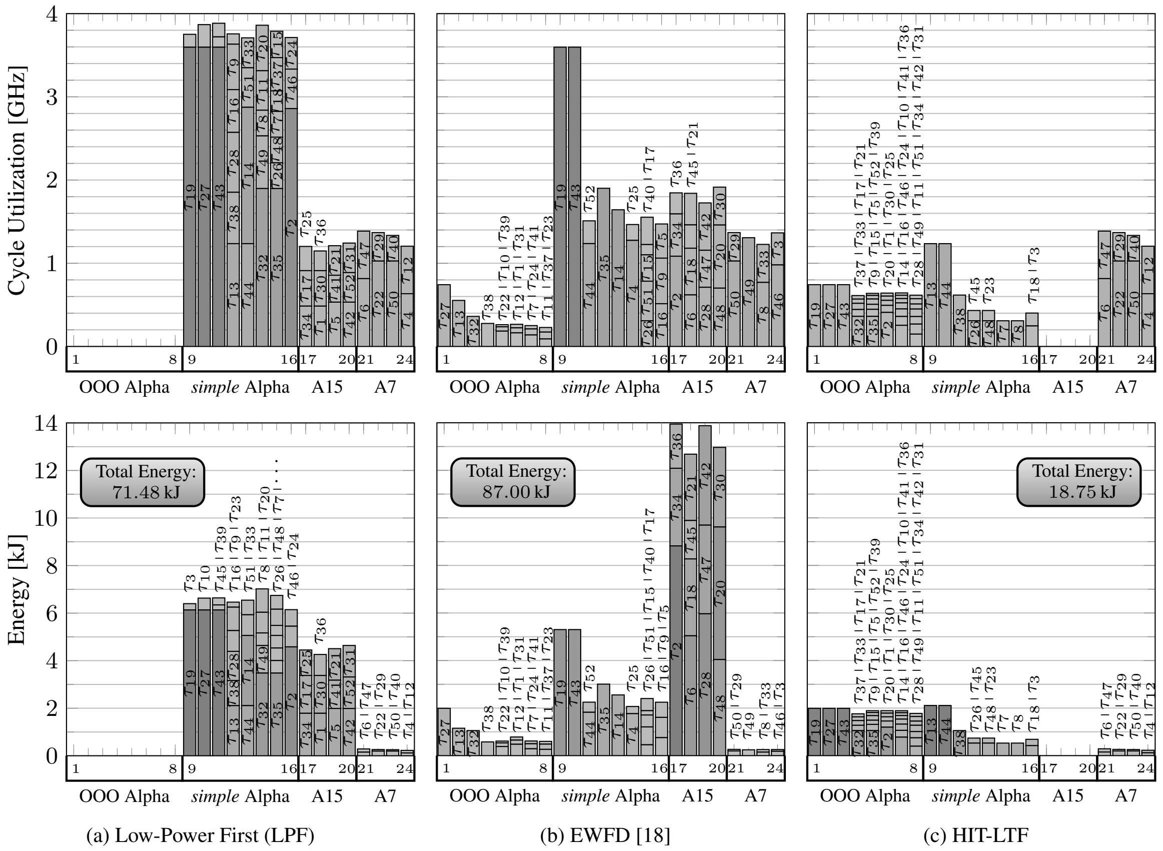
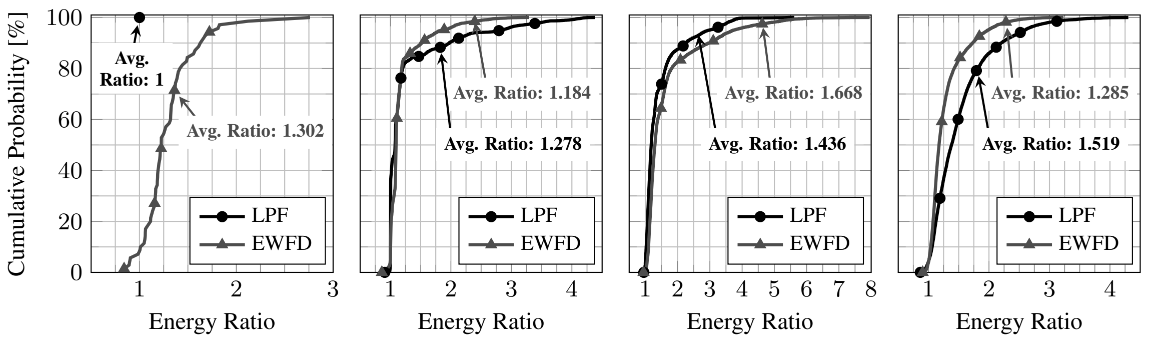


Figure 10.5: Detailed experimental results for one experiment with 52 random tasks when running on platform (c). The height of the bars represents the cycles per second required by a task to meet its timing constraints when executed on the assigned type of core (a task requires different cycles per second when assigned to different core types) on the top figures, and the associated energy consumptions on the bottom figures. The total energy consumption for using LPF and EWFD is 3.81x and 4.64x, respectively, higher than HIT-LTF.

is randomly chosen for each experiment between 10 and 200 tasks by using a uniform distribution. The algorithms used for comparison are the intuitive LPF scheme and the EWFD scheme [18], both described at the beginning of Section 10.5. According to the average power consumption of each type of core when executing at its highest DVFS levels (as seen in Figure 1.1 in Chapter 1.1.1), LPF will attempt to assign tasks to clusters of different types of cores in the following order: Cortex-A7, *simple* Alpha, Cortex-A15, and OOO Alpha. Furthermore, we assume this to be the same arbitrary cluster order considered by EWFD.

In order to have a better understanding of how the algorithms behave, Figure 10.5 presents the detailed



(a) One Cortex-A7 cluster, and one Cortex-A15 cluster

(b) One OOO Alpha cluster, and one *simple* Alpha cluster

(c) One Cortex-A7 cluster, one Cortex-A15 cluster, one OOO Alpha cluster, and one *simple* Alpha cluster

(d) Four OOO Alpha clusters, and four *simple* Alpha clusters

Figure 10.6: Overall energy consumption results for the 4 different evaluated platforms. The results are presented using a normalized empirical cumulative distribution representation. Namely, by normalizing the results to the energy consumption of HIT-LTF, each figure shows the percentage of experiments (among the 10^4 cases) for which the resulting energy consumption ratio for LPF or EWFD is below a specific value. For example, for LPF on platform (d), 80% (20%) of the tested sets of tasks resulted in an energy consumption ratio below (above) 1.82x, compared to HIT-LTF.

experimental results of all algorithms, for one specific experiment (out of the 10^4 experiments) with 52 random tasks when running on architecture (c). In this experiment, the resulting overall energy consumption for using LPF and EWFD is 3.81x and 4.64x, respectively, higher than the overall energy consumption achieved by our HIT-LTF algorithm. Particularly, Figure 10.5a shows that LPF first fills the Cortex-A7 cluster and then the *simple* Alpha cluster, both of them reaching their maximum frequencies of $1.4\,$GHz and $4.0\,$GHz, respectively. Only then LPF assigns tasks to the Cortex-A15 cluster, which is not completely full and thus executes only at $1.3\,$GHz. However, the OOO Alpha cluster remains entirely empty. Similarly, Figure 10.5b shows that EWFD attempts to balance the load among all clusters by using its Next-Fit scheme, such that all clusters have several tasks assigned to them. Nevertheless, due to the incorrect premise that naive load balancing among heterogeneous clusters is an energy-efficient approach, EWFD ignores the actual energy consumption of the tasks on different types of cores and thus assigns many tasks to the Cortex-A15 cluster, which is actually (in average) the less energy-efficient cluster as seen in Figure 10.4. On the other hand, Figure 10.5c shows a truly energy-efficient assignment achieved by our HIT-LTF algorithm. First of all, according to Figure 10.4, we can observe that the Cortex-A15 cluster is always less energy efficient than the Cortex-A7 and the *simple* Alpha clusters, since there is no overlap of their energy factors with respect to the *y*-axis for any frequency. Contrarily, the energy factor of the Cortex-A15 cluster can be smaller than that of the OOO Alpha cluster when the Cortex-A15 cores execute at low DVFS levels and the OOO Alpha cores execute at high DVFS levels, i.e., there is some overlap of their energy factors with respect to the *y*-axis for a few frequencies. However, given that the performance of these two types of cores is similar for the same execution frequencies (as already seen in Figure 1.1 in Chapter 1.1.1), and since for similar frequencies the OOO Alpha cores are more energy efficient than the Cortex-A15 cores (as seen in Figure 10.4), then the Cortex-A15 cluster will generally be the last choice in which HIT-LTF attempts to assign tasks. In fact, this happens to be the case in Figure 10.5c, where we can observe that the Cortex-A15 cluster remains empty. Furthermore, given that HIT-LTF assigns each task according to its specific energy consumption on different types of cores, we can also see that, e.g., tasks τ_2, τ_{19}, and τ_{43} are assigned to the OOO Alpha cluster running at $0.8\,$GHz, consuming considerably less energy than the assignment of these same tasks under LPF or EWFD.

Finally, Figure 10.6 and Figure 10.7 present a summary of all 10^4 experimental results by using a normalized empirical cumulative distribution representation and a box plot representation, respectively. Both figures use *energy consumption ratio* as their metric, which represents the ratio between the overall energy consumption of LPF or EWFD and the overall energy consumption of our HIT-LTF algorithm. In Figure 10.6a and

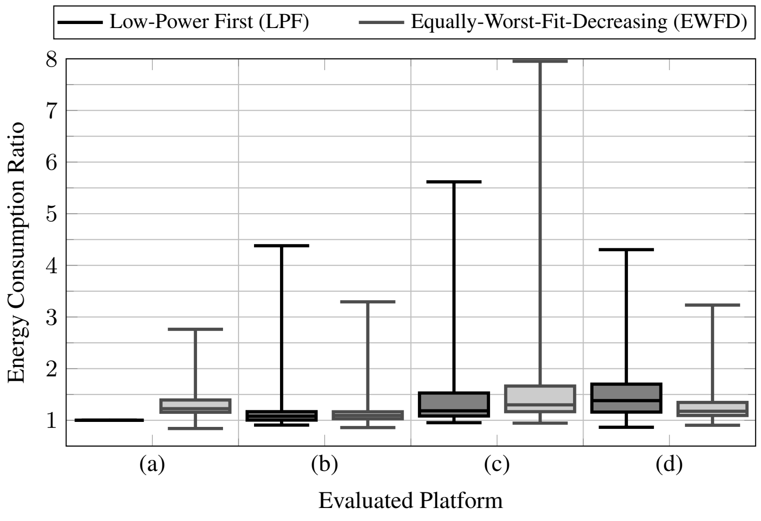


Figure 10.7: Summarized overall energy consumption experimental results for the 4 different evaluated platforms (as detailed in Section 10.5.1). The results are presented in a box plot representation (whiskers represent maximum and minimum ratios), normalized with respect to the overall energy consumption of our HIT-LTF algorithm.

Figure 10.7, we can observe that for platform (a), the LPF scheme achieves the exact same results that our HIT-LTF algorithm for all cases. The reason behind this results becomes clear by referring to the average energy factors in Figure 10.4, where we can again see that the Cortex-A15 cluster is always less energy efficient than the Cortex-A7 cluster, as there exist no overlap of their energy factors with respect to the *y*-axis for any frequency. This implies that HIT-LTF will always attempt to first assign a task to the Cortex-A7 cluster, and a task would only be assigned to the Cortex-A15 cluster when its deadline cannot be satisfied. Given that this is precisely what LPF does, for this special case both algorithms always derive equivalent solutions. For the other three platforms, i.e., platform (b), (c), and (d), Figure 10.6 and Figure 10.7 show that there are a few corner cases in which HIT-LTF consumes slightly more energy than both state-of-the-art approaches. However, among all 10^4 experiments, LPF and EWFD consume in average 1.31x and 1.36x more energy, respectively, than our HIT-LTF algorithm, which translates to an average of 25% energy savings when comparing HIT-LTF to both state-of-the-art solutions. Moreover, when the cluster heterogeneity grows (i.e., when we consider more types of cores on a platform), as is the case in platform (c), there are cases in which LPF and EWFD consume up to 5.62x and 7.96x more energy, respectively, than our HIT-LTF algorithm, which translates in up to 82% and 87% energy savings, respectively.

10.6 Summary

In this chapter, by removing some of the limiting assumptions made in Chapter 9, we have presented an efficient task-to-core assignment/mapping algorithm for energy minimization of periodic real-time tasks (or performance-constrained applications) on clustered multicore systems with heterogeneous voltage/frequency islands, that uses the SFA scheme (described and analyzed in Chapter 8) to decide the DVFS levels on individual clusters once the task-to-core assignment is finished. Unlike most works in the literature, we have considered that different tasks may consume different amounts of average power, even when running on the same core and at the same DVFS levels. Our proposed solution, called HIT-LTF, tests a limited number of different DVFS levels for the clusters, according to the possible orderings of the clusters in regards to their energy efficiency. Moreover, HIT-LTF also takes into account the energy consumption of assigning a task to different clusters for the DVFS levels under consideration, such that it is able to choose the most energy-efficient cluster for every task. Our experimental evaluations show that our HIT-LTF technique behaves much better than the state-of-the-art solutions, resulting in average in 25% less energy consumption (and up to 87% for some cases), while guaranteeing that all tasks meet their deadlines.

Chapter 11

Conclusions

11.1 Dissertation Summary

In this dissertation, we have focused on two of the most relevant problems related to power management on multicore and manycore systems, particularly, performance optimization under power/thermal constraints, and energy minimization under performance constraints.

In the first part of the dissertation (Chapters 5, 6, and 7) we focus on **performance optimization under power/thermal constraints**. Particularly, Chapter 5 showed that using a single and constant per-chip or per-core power budget as an abstraction from thermal problems (e.g., TDP), is a pessimistic approach for manycore systems. To solve this problem, we have presented a novel thermal-aware power budgeting concept called Thermal Safe Power (TSP), which is an abstraction that provides *safe* and *efficient* per-core power budget values as a function of the number of simultaneously active cores. Using TSP results in high total system performance, while the maximum temperature in the chip remains below the critical threshold level that triggers DTM. Furthermore, TSP can also serve as a fundamental tool for guiding task partitioning, core mapping, DPM, and DVFS algorithms on their attempt to achieve high predictable performance under thermal constraints.

In Chapter 6, we have shown that management decisions which are typically used to optimize resource usages (e.g., task migration, DPM, DVFS, etc.), may result in transient temperatures that are much higher than their associated steady-state temperatures. Motivated by this observation, we have presented a lightweight and accurate analytical method, called MatEx, which is based on matrix exponentials and linear algebra, that can be used for estimating/predicting transient thermal peaks, as well as computing any future transient temperatures without requiring incremental computations.

Furthermore, in Chapter 7 we have presented seBoost, an efficient and lightweight boosting technique that uses MatEx for future transient temperature estimation. Our seBoost technique guarantees meeting any performance requirements surges that can occur at runtime, and this is achieved by executing the boosted cores at the required DVFS levels for the entire boosting intervals, while throttling down the non-boosted cores. In order to minimize the performance losses of the applications being executed on the non-boosted cores, the throttling down levels are chosen such that the maximum temperature throughout the chip reaches the critical threshold temperature precisely when the boosting is expected to expire.

In the second part of the dissertation (Chapters 8, 9, and 10) we focus on **energy minimization under performance constraints**. Specifically, for performance-constrained applications or real-time tasks that are already assigned to a specific cluster (or for systems with a global supply voltage), in Chapter 8 we have presented the polynomial-time Double Largest Task First (DLTF) strategy for partitioning tasks to cores based on load balancing and idle energy reduction, as well as the linear-time Single Frequency Approximation (SFA) and Single Voltage Approximation (SVA) schemes for deciding the DVFS levels for execution. In SFA, all the cores in a cluster run at a single voltage and frequency, such that all tasks meet their performance/timing constraints. In SVA, all the cores in the cluster also run at the same single voltage as in SFA; however, the frequency of each core is individually chosen, such that the tasks in each core can meet their performance/timing constraints, but without running at frequencies higher than absolutely necessary. Most importantly, we have provided comprehensive theoretical analysis for the worst-case behavior (in terms of energy and peak power efficiency) of combining DLTF with either SFA or SVA, proving them to be simple, practical, and efficient solutions for commercial systems that have a limited number of cores in every cluster.

Moreover, for homogeneous multicore systems clustered in multiple voltage islands, when using SFA to decide the DVFS levels on individual clusters, in Chapter 9 we focus on task set assignment/mapping. Particularly, we have presented two simple and intuitive mapping heuristics, and we have theoretically analyzed the worst-case efficiency of these heuristics for energy minimization. More importantly, we developed an algorithm based on dynamic programming, called Dynamic Voltage Island Assignment (DYVIA), which derives optimal task set mapping solutions in terms of energy minimization for any task partition. Based on the approximation factor of SFA for single clusters, we have also extended the theoretical analysis from Chapter 8, and presented the approximation factor analysis of mapping task sets with DYVIA when using SFA on individual clusters, which results in the same approximation factor as that of SFA for single clusters.

Finally, Chapter 10 removes some of the limiting assumptions made in Chapter 9, particularly, by considering heterogeneous systems, assuming different average power consumptions for different tasks (even when running on the same core and at the same DVFS levels), and considering the raw set of tasks as an input (instead of already partitioned tasks, i.e., given tasks sets, as an input, as done in Chapter 9). In this way, in Chapter 10 we have presented an efficient task-to-core assignment/mapping algorithm for energy minimization on clustered multicore systems with heterogeneous voltage/frequency islands, called Heterogeneous Island- and Task-Aware Largest Task First (HIT-LTF), that uses the SFA scheme to decide the DVFS levels on individual clusters. Algorithm HIT-LTF tests a limited number of DVFS levels for the clusters, according to the possible orderings of the clusters in regards to their energy efficiency. Furthermore, HIT-LTF takes into account the energy consumption of assigning a task to different clusters for the DVFS levels under consideration, such that it is able to choose the most energy-efficient cluster for every task.

11.2 Current Impact of our Contributions

Several of the contributions presented in this dissertation, already published in peer-reviewed international conferences and journals, have already made some impact in the community and are being used by researchers.

For example, the work in [47] presents a centralized greedy runtime mapping technique for homogeneous multicores which is based on our TSP power budgeting approach (presented in Chapter 5). Specifically, the techniques presented in [47] consist of greedily mapping multi-threaded applications by first selecting the coolest squared region in which to map an application, and then deciding the specific mapping inside the region to distribute heat as much as possible. In this way, the algorithm can derive a mapping that results in higher per-core power budgets than standard concentrated mappings which focus on optimizing the inter-task communication latency. The DVFS levels of the cores are then selected to satisfy the computed TSP budgets for the derived mapping.

The work in [88] presents a multi-objective dynamic power management method based on a control approach, that simultaneously considers a limited power budget (either TDP or our TSP power budgeting technique, presented in Chapter 5), the dynamic behavior of workloads, the utilization and power consumption of individual cores, and the load of the NoC. Authors assume to have per-core DVFS and per-core power gating, both utilized to optimize the performance. In order to prevent sharp power budget violations when new applications are mapped to the system at runtime, this work also proposes a disturbance rejecter that pro-actively scales down the activity of already-running applications.

The work in [23] presents a power-aware non-intrusive runtime testing approach for manycore systems. During the idle periods of different cores, the proposed approach schedules software based self-test routines, considering that cores should be tested at different DVFS levels, while limiting the delays for the execution of true workloads, and by satisfying the power budgets derived by our TSP power budgeting approach (presented in Chapter 5). Based on a device aging model, a test criticality metric is used to select the cores to be tested at a given point in time.

The work in [111] presents a hierarchical management strategy, called *FoToNoC*, based on a folded torus-like NoC for manycore systems. Namely, *FoToNoC* consist in a superposition of regular and application-specific NoC topologies, such that physically distributed cores can be interconnected with reduced intercommunication costs and organized in logically condensed virtual NoC clusters, in order to enable the distribution of active cores throughout the chip for reducing the chip's temperature without incurring in high communication costs. For their work, authors use our MatEx transient temperature computation tool (presented in Chapter 6) to enable runtime thermal supervision inside *FoToNoC*.

11.3 Future Works

For performance optimization under power/thermal constraints, our future work will involve deriving centralized thread-to-core mapping and DVFS level selection algorithms for heterogeneous systems based on the TSP concept. Furthermore, we will consider extending our TSP technique to work on distributed systems, which will serve as a basis for us to also derive distributed thread-to-core mapping and DVFS level selection algorithms. In regards to MatEx, our plans are to extend our tool to account for the dependency between leakage power consumption and temperature, thus increasing the accuracy of MatEx by removing the current pessimistic assumption of modeling power consumption at the critical temperature. Moreover, aside of offering MatEx as an open-source software tool, we also plan to develop an open-source RTL hardware implementation of MatEx to serve as a hardware accelerator, which can be used at runtime on systems with a very large number of cores.

For energy minimization under performance constraints, our future work will focus on extending the algorithms and analyses presented in this dissertation to consider task dependency and intertask communications, which might play an important role in task scheduling and on the overall energy consumption. We would also like to extend our contributions to consider a sporadic real-time tasks model (more general than periodic real-time tasks), where the deadline and period of a task are not necessarily equivalent, and where tasks only provide the minimum arrival rate between task instances (rather than a precise period). Furthermore, besides considering EDF for dynamically scheduling tasks inside individual cores, we will also like to extend our analysis of SFA and DYVIA to consider some fixed-priority scheduling policies, e.g., rate-monotonic scheduling or deadline-monotonic scheduling.

Bibliography

[1] ANSYS Inc. ANSYS R15.0. `www.ansys.com`, 2014.

[2] Ehsan K. Ardestani and Jose Renau. ESESC: A fast multicore simulator using time-based sampling. In *Proceedings of 19th IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 448–459, 2013.

[3] H. Aydin and Q. Yang. Energy-aware partitioning for multiprocessor real-time systems. In *Proceedings of 17th International Parallel and Distributed Processing Symposium (IPDPS)*, pages 113–121, 2003.

[4] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. The PARSEC benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 72–81, 2008.

[5] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1–7, May 2011.

[6] Haseeb Bokhari, Haris Javaid, Muhammad Shafique, Jörg Henkel, and Sri Parameswaran. darkNoC: Designing energy-efficient network-on-chip with multi-vt cells for dark silicon. In *Proceedings of the 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 161:1–161:6, 2014.

[7] J. Frédéric Bonnans, Jean Charles Gilbert, Claude Lemaréchal, and Claudia A. Sagastizábal. *Numerical Optimization: Theoretical and Practical Aspects*. Springer-Verlag, 2 edition, 2006.

[8] Trevor E. Carlson, Wim Heirman, Stijn Eyerman, Ibrahim Hur, and Lieven Eeckhout. An evaluation of high-level mechanistic core models. *ACM Transactions on Architecture and Code Optimization (TACO)*, 11(3):28:1–28:25, August 2014.

[9] Jeff Casazza. First the tick, now the tock: Intel microarchitecture (nehalem). White paper, Intel Corporation, 2009.

[10] James Charles, Preet Jassi, Narayan S. Ananth, Abbas Sadat, and Alexandra Fedorova. Evaluation of the Intel core i7 turbo boost feature. In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*, pages 188–197, 2009.

[11] Jian-Jia Chen, Heng-Ruey Hsu, and Tei-Wei Kuo. Leakage-aware energy-efficient scheduling of real-time tasks in multiprocessor systems. In *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 408–417, 2006.

[12] Jian-Jia Chen and Lothar Thiele. Energy-efficient scheduling on homogeneous multiprocessor platforms. In *Proceedings of the ACM Symposium on Applied Computing (SAC)*, pages 542–549, 2010.

[13] Ayse Kivilcim Coskun, Tajana Šimunic Rosing, and Kenny C. Gross. Utilizing predictors for efficient thermal management in multiprocessor SoCs. *Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 28(10):1503–1516, October 2009.

[14] Pepijn J. de Langen and Ben H. H. Juurlink. Leakage-aware multiprocessor scheduling for low power. In *Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS)*, 2006.

[15] V. Devadas and H. Aydin. Coordinated power management of periodic real-time tasks on chip multiprocessors. In *Proceedings of the International Conference on Green Computing (GREENCOMP)*, pages 61 –72, 2010.

[16] Thomas Ebi, Mohammad Abdullah Al Faruque, and Jörg Henkel. TAPE: Thermal-aware agent-based power economy for multi/many-core architectures. In *the International Conference on Computer-Aided Design (ICCAD)*, pages 302–309, 2009.

[17] Thom J. A. Eguia, Sheldon X.-D. Tan, Ruijing Shen, Eduardo H. Pacheco, and Murli Tirumala. General behavioral thermal modeling and characterization for multi-core microprocessor design. In *Proceedings of the 18th Design, Automation and Test in Europe (DATE)*, pages 1136–1141, 2010.

[18] Abdullah Elewi, Mohamed Shalan, Medhat Awadalla, and Elsayed M. Saad. Energy-efficient task allocation techniques for asymmetric multiprocessor embedded systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 13(2s):71:1–71:27, January 2014.

[19] H. Esmaeilzadeh, E. Blem, R. St.Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. In *Proceedings of the 38th International Symposium on Computer Architecture (ISCA)*, pages 365–376, 2011.

[20] R.L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17:263–269, 1969.

[21] Peter Greenhalgh. big.LITTLE processing with ARM Cortex-A15 & Cortex-A7. White paper, ARM Limited, September 2011.

[22] A. Grenat, S. Pant, R. Rachala, and S. Naffziger. 5.6 adaptive clocking system for improved power efficiency in a 28nm x86-64 microprocessor. In *IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 106–107, 2014.

[23] M. H. Haghbayan, A. M. Rahmani, A. Miele, M. Fattah, J. Plosila, P. Liljeberg, and H. Tenhunen. A power-aware approach for online test scheduling in many-core architectures. *IEEE Transactions on Computers (TC)*, 65(3):730–743, March 2016.

[24] Jian-Jun Han, Xiaodong Wu, Dakai Zhu, Hai Jin, L.T. Yang, and J.-L. Gaudiot. Synchronization-aware energy management for VFI-based multicore real-time systems. *IEEE Transactions on Computers (TC)*, 61(12):1682–1696, Dec 2012.

[25] Vinay Hanumaiah, Sarma Vrudhula, and Karam S. Chatha. Performance optimal online DVFS and task migration techniques for thermally constrained multi-core processors. *Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 30(11):1677–1690, November 2011.

[26] Hardkernel Co., Ltd. Odroid-XU3. `www.hardkernel.com`.

[27] Jörg Henkel, Heba Khdr, Santiago Pagani, and Muhammad Shafique. New trends in dark silicon. In *Proceedings of the 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 119:1–119:6, June 2015. **[HiPEAC Paper Award]**.

[28] Jörg Henkel, Santiago Pagani, Heba Khdr, Florian Kriebel, Semeen Rehman, and Muhammad Shafique. Towards performance and reliability-efficient computing in the dark silicon era. In *Proceedings of the 19th Design, Automation and Test in Europe (DATE)*, March 2016.

[29] Sebastian Herbert and Diana Marculescu. Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, pages 38–43, 2007.

[30] J. Howard, S. Dighe, S.R. Vangal, G. Ruhl, N. Borkar, S. Jain, V. Erraguntla, M. Konow, M. Riepen, M. Gries, G. Droege, T. Lund-Larsen, S. Steibl, S. Borkar, V.K. De, and R. Van Der Wijngaart. A 48-core IA-32 processor in 45nm CMOS using on-die message-passing and DVFS for performance and power scaling. *IEEE Journal of Solid-State Circuits*, 46(1):173–183, 2011.

[31] Xing Hu, Yi Xu, Jun Ma, Guoqing Chen, Yu Hu, and Yuan Xie. Thermal-sustainable power budgeting for dynamic threading. In *Proceedings of the 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 187:1–187:6, 2014.

[32] Pei-Yu Huang and Yu-Min Lee. Full-chip thermal analysis for the early design stage via generalized integral transforms. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 17(5):613–626, May 2009.

[33] Wei Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M.R. Stan. HotSpot: a compact thermal modeling methodology for early-stage VLSI design. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 14(5):501–513, May 2006.

[34] Intel Corporation. Dual-core intel xeon processor 5100 series datasheet, revision 003, August 2007.

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

[36] Intel Corporation. SCC external architecture specification (EAS), revision 0.98, July 2010.

[37] Sandy Irani, Sandeep Shukla, and Rajesh Gupta. Algorithms for power savings. In *Proceedings of the 14th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 37–46, 2003.

[38] Canturk Isci and Margaret Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. In *Proceedings of the 36th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 93–104, 2003.

[39] ITRS. International technology roadmap for semiconductors, 2011 edition. `www.itrs.net`.

[40] Janmartin Jahn, Sebastian Kobbe, Santiago Pagani, Jian-Jia Chen, and Jörg Henkel. Work in Progress: Malleable software pipelines for efficient many-core system utilization. In *Proceedings of the 6th Many-core Applications Research Community (MARC) Symposium at ONERA*, pages 30–33, July 2012.

[41] Janmartin Jahn, Sebastian Kobbe, Santiago Pagani, Jian-Jia Chen, and Jörg Henkel. Runtime resource allocation for software pipelines. In *Proceedings of the 16th International Workshop on Software and Compilers for Embedded Systems (M-SCOPES)*, pages 96–99, June 2013.

[42] Janmartin Jahn, Santiago Pagani, Jian-Jia Chen, and Jörg Henkel. MOMA: Mapping of memory-intensive software-pipelined applications for systems with multiple memory controllers. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 508–515, November 2013.

[43] Janmartin Jahn, Santiago Pagani, Sebastian Kobbe, Jian-Jia Chen, and Jörg Henkel. Optimizations for configuring and mapping software pipelines in manycore. In *Proceedings of the 50th IEEE/ACM Design Automation Conference (DAC)*, pages 130:1–130:8, June 2013. **[HiPEAC Paper Award]**.

[44] Janmartin Jahn, Santiago Pagani, Sebastian Kobbe, Jian-Jia Chen, and Jörg Henkel. Runtime resource allocation for software pipelines. *ACM Transactions on Parallel Computing (TOPC)*, 2(1):5:1–5:23, May 2015.

[45] R. Jejurikar, C. Pereira, and R. Gupta. Leakage aware dynamic voltage scaling for real-time embedded systems. In *Proceedings of the 41st IEEE/ACM Design Automation Conference (DAC)*, pages 275–280, 2004.

[46] Charles H. Jones. Generalized hockey stick identities and N-dimensional blockwalking, 1994.

[47] 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 *33rd IEEE International Conference on Computer Design (ICCD)*, pages 573–580, 2015.

[48] Heba Khdr, Santiago Pagani, Muhammad Shafique, and Jörg Henkel. Thermal constrained resource management for mixed ILP-TLP workloads in dark silicon chips. In *Proceedings of the 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 179:1–179:6, June 2015. **[HiPEAC Paper Award]**.

[49] Heba Khdr, Santiago Pagani, Éricles Sousa, Vahid Lari, Anuj Pathania, Frank Hannig, Muhammad Shafique, Jürgen Teich, and Jörg Henkel. Power density-aware resource management for heterogeneous tiled multicores. *IEEE Transactions on Computers (TC)*, 2016.

[50] Hyunjin Kim, Hyejeong Hong, Hong-Sik Kim, Jin-Ho Ahn, and Sungho Kang. Total energy minimization of real-time tasks in an on-chip multiprocessor using dynamic voltage scaling efficiency metric. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 27(11):2088–2092, 2008.

[51] S. Klab, Andrzej Napieralski, and G. De Mey. Logi-thermal simulation of digital CMOS ICs with emphasis on dynamic power dissipation. In *Proceedings of the 16th International Conference on Mixed Design of Integrated Circuits Systems (MIXDES)*, pages 361–365, 2009.

[52] S. Kodase, Shige Wang, Zonghua Gu, and K.G. Shin. Improving scalability of task allocation and scheduling in large distributed real-time systems using shared buffers. In *the 9th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 181–188, 2003.

[53] Fanxin Kong, Wang Yi, and Qingxu Deng. Energy-efficient scheduling of real-time tasks on cluster-based multicores. In *Proceedings of the 14th Design, Automation and Test in Europe (DATE)*, pages 1–6, 2011.

[54] Emre Kultursay, Karthik Swaminathan, Vinay Saripalli, Vijaykrishnan Narayanan, Mahmut T. Kandemir, and Suman Datta. Performance enhancement under power constraints using heterogeneous CMOS-TFET multicores. In *Proceedings of the 7th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 245–254, 2012.

[55] B.C. Kuo and F. Golnaraghi. *Automatic Control Systems*. John Wiley & Sons, 8 edition, 2002.

[56] Yu-Min Lee, Tsung-Heng Wu, Pei-Yu Huang, and Chi-Ping Yang. NUMANA: A hybrid numerical and analytical thermal simulator for 3-D ICs. In *Proceedings of the 16th Design, Automation and Test in Europe (DATE)*, pages 1379–1384, 2013.

[57] Sheng Li, Jung-Ho Ahn, R.D. Strong, J.B. Brockman, D.M. Tullsen, and N.P. Jouppi. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42nd IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 469–480, 2009.

[58] Yanbing Li and Jörg Henkel. A framework for estimation and minimizing energy dissipation of embedded HW/SW systems. In *Proceedings of the 35th ACM/IEEE Design Automation Conference (DAC)*, pages 188–193, 1998.

[59] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.

[60] G.A. Moreno and D. de Niz. An optimal real-time voltage and frequency scaling for uniform multiprocessors. In *Proceedings of the 18th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 21–30, 2012.

[61] H. M. Moya-Cessa and F. Soto-Eguibar. *Differential Equations: An Operational Approach*. Rinton Press, 2011.

[62] Waqaas Munawar, Heba Khdr, Santiago Pagani, Muhammad Shafique, Jian-Jia Chen, and Jörg Henkel. Peak power management for scheduling real-time tasks on heterogeneous many-core systems. In *Proceedings of the 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, December 2014.

[63] Thannirmalai Somu Muthukaruppan, Anuj Pathania, and Tulika Mitra. Price theory based power management for heterogeneous multi-cores. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 161–176, 2014.

[64] Thannirmalai Somu Muthukaruppan, Mihai Pricopi, Vanchinathan Venkataramani, Tulika Mitra, and Sanjay Vishin. Hierarchical power management for asymmetric multi-core in dark silicon era. In *Proceedings of the 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 174:1–174:9, 2013.

[65] Nikita Nikitin and Jordi Cortadella. Static task mapping for tiled chip multiprocessors with multiple voltage islands. In *Proceedings of the 25th International Conference on Architecture of Computing Systems (ARCS)*, pages 50–62, 2012.

[66] Sebastien Nussbaum. AMD trinity APU. In *Hot Chips: A Symposium on High Performance Chips*, 2012.

[67] Santiago Pagani, Lars Bauer, Qingqing Chen, Elisabeth Glocker, Frank Hannig, Andreas Herkersdorf, Heba Khdr, Anuj Pathania, Ulf Schlichtmann, Doris Schmitt-Landsiedel, Mark Sagi, Éricles Sousa, Philipp Wagner, Volker Wenzel, Thomas Wild, and Jörg Henkel. Dark silicon management: An integrated and coordinated cross-layer approach. *Special Issue of (De Gruyter Oldenbourg) Information Technology on Invasive Computing*, 2017.

[68] Santiago Pagani and Jian-Jia Chen. Energy efficiency analysis for the single frequency approximation (SFA) scheme. In *Proceedings of the 19th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 82–91, August 2013. **[Best Paper Award]**.

[69] Santiago Pagani and Jian-Jia Chen. Energy efficient task partitioning based on the single frequency approximation scheme. In *Proceedings of the 34th IEEE Real-Time Systems Symposium (RTSS)*, pages 308–318, December 2013.

[70] Santiago Pagani and Jian-Jia Chen. Energy efficiency analysis for the single frequency approximation (SFA) scheme. *ACM Transactions on Embedded Computing Systems (TECS)*, 13(5s):158:1–158:25, November 2014.

[71] Santiago Pagani, Jian-Jia Chen, and Jörg Henkel. Energy and peak power efficiency analysis for the single voltage approximation (SVA) scheme. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 34(9):1415–1428, September 2015.

[72] Santiago Pagani, Jian-Jia Chen, and Minming Li. Energy efficiency on multi-core architectures with multiple voltage islands. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 26(6):1608–1621, June 2015.

[73] Santiago Pagani, Jian-Jia Chen, Muhammad Shafique, and Jörg Henkel. MatEx: Efficient transient and peak temperature computation for compact thermal models. In *Proceedings of the 18th Design, Automation and Test in Europe (DATE)*, pages 1515–1520, March 2015.

[74] Santiago Pagani, Jian-Jia Chen, Muhammad Shafique, and Jörg Henkel. Thermal-aware power budgeting for dark silicon chips. In *Proceedings of the 2nd Workshop on Low-Power Dependable Computing (LPDC) at the International Green and Sustainable Computing Conference (IGSC)*, December 2015.

[75] Santiago Pagani, Heba Khdr, Jian-Jia Chen, Muhammad Shafique, Minming Li, and Jörg Henkel. Thermal Safe Power: Efficient thermal-aware power budgeting for manycore systems in dark silicon. In Amir M. Rahmani, Pasi Liljeberg, Ahmed Hemani, Axel Jantsch, and Hannu Tenhunen, editors, *The Dark Side of Silicon*. Springer, 2017.

[76] Santiago Pagani, Heba Khdr, Jian-Jia Chen, Muhammad Shafique, Minming Li, and Jörg Henkel. Thermal Safe Power (TSP): Efficient power budgeting for heterogeneous manycore systems in dark silicon. *IEEE Transactions on Computers (TC)*, 66(1):147–162, Jan 2017. **[Feature Paper of the Month]**.

[77] Santiago Pagani, Heba Khdr, Waqaas Munawar, Jian-Jia Chen, Muhammad Shafique, Minming Li, and Jörg Henkel. TSP: Thermal Safe Power - Efficient power budgeting for many-core systems in dark silicon. In *Proceedings of the 9th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 10:1–10:10, October 2014. **[Best Paper Award]**.

[78] Santiago Pagani, Anuj Pathania, Muhammad Shafique, Jian-Jia Chen, and Jörg Henkel. Energy efficiency for clustered heterogeneous multicores. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2017.

[79] Santiago Pagani, Muhammad Shafique, and Jörg Henkel. Design space exploration and run-time adaptation for multi-core resource management under performance and power constraints. In Soonhoi Ha and Jürgen Teich, editors, *Handbook of Hardware/Software Codesign*. Springer, 2017.

[80] Santiago Pagani, Muhammad Shafique, Heba Khdr, Jian-Jia Chen, and Jörg Henkel. seBoost: Selective boosting for heterogeneous manycores. In *Proceedings of the 10th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 104–113, October 2015.

[81] Anuj Pathania, Santiago Pagani, Muhammad Shafique, and Jörg Henkel. Power management for mobile games on asymmetric multi-cores. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, pages 243–248, July 2015.

[82] Nathaniel Pinckney, Korey Sewell, Ronald G. Dreslinski, David Fick, Trevor Mudge, Dennis Sylvester, and David Blaauw. Assessing the performance limits of parallelized near-threshold computing. In *the 49th Design Automation Conference (DAC)*, pages 1147–1152, 2012.

[83] M.D. Powell, A. Biswas, J.S. Emer, S.S. Mukherjee, B.R. Sheikh, and S. Yardi. CAMP: A technique to estimate per-structure power at run-time using a few simple parameters. In *Proceedings of the 15th IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 289–300, 2009.

[84] Hanhua Qian, Hao Liang, Chip-Hong Chang, Wei Zhang, and Hao Yu. Thermal simulator of 3D-IC with modeling of anisotropic TSV conductance and microchannel entrance effects. In *Proceedings of the 18th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 485–490, 2013.

[85] Arun Raghavan, Yixin Luo, Anuj Chandawalla, Marios Papaefthymiou, Kevin P. Pipe, Thomas F. Wenisch, and Milo M. K. Martin. Computational sprinting. In *Proceedings of the IEEE 18th International Symposium on High-Performance Computer Architecture (HPCA)*, pages 1–12, 2012.

[86] Bharathwaj Raghunathan and Siddharth Garg. Job arrival rate aware scheduling for asymmetric multicore servers in the dark silicon era. In *Proceedings of the 9th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2014.

[87] Bharathwaj Raghunathan, Yatish Turakhia, Siddharth Garg, and Diana Marculescu. Cherry-picking: Exploiting process variations in dark-silicon homogeneous chip multi-processors. In *Proceedings of the 16th Design, Automation and Test in Europe (DATE)*, pages 39–44, 2013.

[88] A. M. Rahmani, M. H. Haghbayan, A. Kanduri, A. Y. Weldezion, P. Liljeberg, J. Plosila, A. Jantsch, and H. Tenhunen. Dynamic power management for many-core platforms in the dark silicon era: A multi-objective control approach. In *IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 219–224, July 2015.

[89] Devendra Rai, Hoeseok Yang, Iuliana Bacivarov, and Lothar Thiele. Power agnostic technique for efficient temperature estimation of multicore embedded systems. In *Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES)*, pages 61–70, 2012.

[90] Ronald L. Rardin. *Optimization in Operations Research.* Prentice Hall, 1998.

[91] E. Rotem, A. Naveh, D. Rajwan, A. Ananthakrishnan, and E. Weissmann. Power-management architecture of the intel microarchitecture code-named sandy bridge. *IEEE Micro*, 32(2):20–27, March 2012.

[92] Samsung Electronics Co., Ltd. Exynos 5 Octa (5422). `www.samsung.com/exynos`.

[93] J. Sartori and R. Kumar. Three scalable approaches to improving many-core throughput for a given peak power budget. In *Proceedings of the International Conference on High Performance Computing (HiPC)*, pages 89–98, 2009.

[94] Euiseong Seo, Jinkyu Jeong, Seon-Yeong Park, and Joonwon Lee. Energy efficient scheduling of real-time tasks on multicore processors. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 19(11):1540–1552, 2008.

[95] Muhammad Shafique, Siddharth Garg, Jörg Henkel, and Diana Marculescu. The EDA challenges in the dark silicon era: Temperature, reliability, and variability perspectives. In *Proceedings of the 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 185:1–185:6, 2014.

[96] Muhammad Shafique, Dennis Gnad, Siddharth Garg, and Jörg Henkel. Variability-aware dark silicon management in on-chip many-core systems. In *Proceedings of the 18th Design, Automation and Test in Europe (DATE)*, pages 387–392, March 2015.

[97] Timothy Sherwood, Erez Perelman, Greg Hamerly, Suleyman Sair, and Brad Calder. Discovering and exploiting program phases. *IEEE Micro*, 23(6):84–93, November 2003.

[98] Cheng Tan, T.S. Muthukaruppan, T. Mitra, and Lei Ju. Approximation-aware scheduling on heterogeneous multi-core architectures. In *Proceedings of the 20th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 618–623, Jan 2015.

[99] Michael B. Taylor. Is dark silicon useful?: harnessing the four horsemen of the coming dark silicon apocalypse. In *Proceedings of the 49th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1131–1136. ACM, 2012.

[100] Eigen C++ template library. (v3.2.2). `eigen.tuxfamily.org`, 2014.

[101] Anas Toma, Santiago Pagani, Jian-Jia Chen, Wolfgang Karl, and Jörg Henkel. An energy-efficient middleware for computation offloading in real-time embedded systems. In *Proceedings of the 22nd IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, August 2016.

[102] Vijay V. Vazirani. *Approximation Algorithms.* Springer-Verlag New York, Inc., 2001.

[103] Hai Wang, Sheldon X.-D. Tan, Duo Li, Ashish Gupta, and Yuan Yuan. Composable thermal modeling and simulation for architecture-level thermal designs of multicore microprocessors. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 18(2):28:1–28:27, April 2013.

[104] Ting-Yuan Wang and C.C. Chen. 3-D Thermal-ADI: a linear-time chip level transient thermal simulator. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 21(12):1434–1445, Dec 2002.

[105] Gerhard Weiss, editor. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, 1999.

[106] Wei Wu, Lingling Jin, Jun Yang, Pu Liu, and Sheldon X.-D. Tan. Efficient power modeling and software thermal sensing for runtime temperature monitoring. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 12(3):25:1–25:29, May 2008.

[107] Xiaodong Wu, Yuzhu Zeng, and Jian-Jun Han. Energy-efficient task allocation for VFI-based real-time multi-core systems. In *Proceedings of the International Conference on Information Science and Cloud Computing Companion (ISCC-C)*, pages 123–128, Dec 2013.

[108] Ruibin Xu, Dakai Zhu, Cosmin Rusu, Rami Melhem, and Daniel Mossé. Energy-efficient policies for embedded clusters. In *Proceedings of the ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, pages 1–10, 2005.

[109] Chuan-Yue Yang, Jian-Jia Chen, and Tei-Wei Kuo. An approximation algorithm for energy-efficient scheduling on a chip multiprocessor. In *Proceedings of the 8th Design, Automation and Test in Europe (DATE)*, pages 468–473, 2005.

[110] Chuan-Yue Yang, Jian-Jia Chen, Tei-Wei Kuo, and Lothar Thiele. An approximation scheme for energy-efficient scheduling of real-time tasks in heterogeneous multiprocessor systems. In *Proceedings of the 12th Design, Automation and Test in Europe (DATE)*, pages 694–699, 2009.

[111] L. Yang, W. Liu, W. Jiang, M. Li, J. Yi, and E. H. M. Sha. FoToNoC: A hierarchical management strategy based on folded lorus-like network-on-chip for dark silicon many-core systems. In *21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 725–730, Jan 2016.

[112] Yong Zhan and Sachin S. Sapatnekar. Fast computation of the temperature distribution in VLSI chips using the discrete cosine transform and table look-up. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 87–92, 2005.

[113] W. Zhao, R. Chellappa, P. J. Phillips, and A. Rosenfeld. Face recognition: A literature survey. *Computing Surveys (CSUR)*, 35(4):399–458, December 2003.

[114] A Ziabari, J.-H. Park, E.K. Ardestani, J. Renau, S.-M. Kang, and A Shakouri. Power blurring: Fast static and transient thermal analysis method for packaged integrated circuits and power devices. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2014.

List of Figures

List of Tables

Acronyms

ADI Alternating Direction Implicit. 21

ARMA Autoregressive Moving Average. 22

blackscholes Application from the PARSEC benchmark suite. 55, 59, 80, 155

bodytrack Application from the PARSEC benchmark suite. 7, 55, 59, 74, 75, 77–82, 117, 155, 171

BUH Balanced Utilization Heuristic. 124, 126, 127, 134, 138, 140, 173

CBGA Ceramic Ball Grid Array. 22, 171

CCH Consecutive Cores Heuristic. 123–127, 134, 138, 140, 141, 173, 183

DLTF Double Largest Task First. vii, 15, 16, 87, 88, 91–96, 99, 105, 106, 108, 117–119, 122, 123, 159, 160, 173, 183, 185, 187–190

DLTF-SFA Double Largest Task First combined with Single Frequency Approximation. 87, 88, 94, 95, 97, 102, 106–112, 114–119, 173, 181

DLTF-SVA Double Largest Task First combined with Single Voltage Approximation. 87–89, 96, 97, 110–115, 117–119, 173, 181

DPM Dynamic Power Management. 10–12, 15–17, 39, 51, 59–61, 71, 88, 89, 93, 95, 102, 106–108, 110, 111, 117, 121, 139, 141, 143, 159, 175

DTM Dynamic Thermal Management. 8, 9, 36, 39, 40, 45, 51–54, 56–58, 60–62, 71, 74, 159, 171, 187, 188

DVFS Dynamic Voltage and Frequency Scaling. vii, 10–17, 19–25, 27, 36, 37, 39, 44, 46, 51–54, 56–61, 71, 73–82, 84, 86–90, 92–95, 97, 102, 106–108, 110, 115, 116, 119, 121, 122, 125–131, 134, 136, 139–141, 143–161, 171–175, 181–183, 186–189

DYVIA Dynamic Voltage Island Assignment. 121, 130–135, 137, 138, 140, 141, 143, 160, 161, 174, 175, 182, 183

EDF Earliest Deadline First. 25, 26, 90, 95, 117, 161, 171

EOH Extremal Optimization Heuristic. 24, 124, 134, 135, 137, 138, 140

EWFD Equally-Worst-Fit-Decreasing. 24, 154–158

EXU Execution Unit. 27

FEA Finite Element Analysis. 22

ferret Application from the PARSEC benchmark suite. 155

FFI-LTF Fixed Frequency Island-Aware Largest Task First. 149–154

FFT Fast Fourier Transform. 136

FIT-LTF Fixed Frequency Island- and Task-Aware Largest Task First. 153, 154

gem5 gem5 multicore simulator. 6, 8, 20, 21, 27, 29, 35–38, 69, 117, 139, 155, 156

GIPS Giga Instructions per Second. 21, 75, 78, 80–82

GIT Generalized Integral Transforms. 22

HI-LTF Heterogeneous Island-Aware Largest Task First. 151, 152, 154

HIT-LTF Heterogeneous Island- and Task-Aware Largest Task First. 154–158, 160, 174

HotSpot HotSpot modeling and temperature computation tool. 15, 20, 22, 33, 35, 36, 38–41, 61, 62, 68–71, 172

IFU Instruction Fetch Unit. 27

ILP Instruction-Level Parallelism. 6, 20, 21

IPC Instructions per Cycle. 5

IPS Instructions per Second. 5, 57, 58

LPF Low-Power First. 154–158

LPT Longest Processing Time. 90, 189

LSU Load and Store Unit. 27

LTF Largest Task First. 14, 15, 23, 24, 87, 90–95, 108–110, 122, 123, 136, 139–141, 149, 154, 155, 173, 188–190

LTR Left to Right. 102, 106, 107

MatEx (from Matrix Exponentials) Transient and Peak Temperature Computation Tool for Compact Thermal Models. vii, 15, 16, 52, 53, 61, 63, 65, 68–71, 74, 76–78, 80, 86, 159, 161, 172, 186

McPAT McPAT power consumption simulator. 6–8, 20, 27, 29, 35–38, 69, 117, 139, 155, 156

NoC Network on Chip. 6, 36, 160

Odroid-XU3 Odroid-XU3 mobile platform, with an Exynos 5 Octa (5422) chip based on ARM's "big.LITTLE" architecture. 35, 36, 38, 59, 82, 155

OOO out-of-order. 6–8, 20, 21, 27, 29, 37, 38, 60, 82, 101, 104, 107, 110, 112, 114, 115, 117, 144, 155–157

PARSEC PARSEC benchmark suite. 6–8, 20, 21, 25, 29, 30, 35, 36, 38, 55, 59, 69, 74, 82, 84, 85, 114, 117, 144, 145, 155, 156, 171, 173, 174, 177, 179

QoS Quality of Service. 24

SCC Single Chip Cloud computer. 11, 29, 37, 121, 123, 133–141, 146, 174, 175

seBoost Selective Boosting for Manycore Systems. vii, 15, 16, 74, 76–86, 159, 172, 173

SFA Single Frequency Approximation. vii, 14–16, 23, 24, 87, 88, 91–95, 106, 115–117, 119, 121–123, 125–132, 134, 136, 137, 139–141, 143–148, 158–161, 173, 181–183, 187, 188

SVA Single Voltage Approximation. vii, 15, 16, 87, 88, 91, 92, 95, 96, 111, 115–117, 119, 159, 160, 173, 183, 187, 188

swaptions Application from the PARSEC benchmark suite. 55, 59, 155

TDP Thermal Design Power. vii, 8, 13, 15, 20, 24, 39, 51–53, 55, 60, 73, 159, 160, 172

TLP Thread-Level Parallelism. 6, 20

TSP Thermal Safe Power. vii, 15, 16, 39, 41, 42, 44–49, 51–60, 71, 159–161, 171, 172

VLSI Very Large-Scale Integration. 11, 22

x264 Application from the PARSEC benchmark suite. 13, 29, 55, 59, 74, 77–79, 82, 114, 117, 144, 155

Symbols

$\mathbf{A}$ Matrix $\mathbf{A} = [a_{i,j}]_{N \times N}$ that contains the thermal capacitance values of an RC thermal network (generally a diagonal matrix, since thermal capacitances are modeled to ground). 33, 36, 62, 68, 181, 182

$a_{i,j}$ Element in row i and column j inside the thermal capacitance matrix $\mathbf{A}$, such that $1 \leq i \leq N$ and $1 \leq j \leq N$. 33

$\mathbf{AF}^{\textbf{DVFS=SFA}}_{\textbf{ASG=ANY}}$ Approximation factor for *any* task partition mapping heuristic that uses $\left\lceil \frac{M^\star}{K} \right\rceil$ clusters with non-empty task sets, when using SFA to decide the DVFS levels on individual clusters, against the optimal assignment that also uses SFA in individual clusters. 125, 126

$\mathbf{AF}^{\textbf{energy } \cancel{\textbf{overheads}}}_{\textbf{DLTF-SFA}}$ Approximation factor (i.e., the worst-case behavior) of DLTF-SFA for energy minimization in relation to the optimal task partitioning and optimal DVFS solution for energy minimization (i.e., the task partitioning and DVFS solution that result in the minimum energy consumption) when we consider negligible *overheads for sleeping*. 102–104, 106, 107, 111

$\mathbf{AF}^{\textbf{energy}}_{\textbf{DLTF-SFA}}$ Approximation factor (i.e., the worst-case behavior) of DLTF-SFA for energy minimization in relation to the optimal task partitioning and optimal DVFS solution for energy minimization (i.e., the task partitioning and DVFS solution that result in the minimum energy consumption). 88, 107, 116

$\mathbf{AF}^{\textbf{peak power}}_{\textbf{DLTF-SFA}}$ Approximation factor (i.e., the worst-case behavior) of DLTF-SFA for peak power reduction in relation to the optimal task partitioning and optimal DVFS solution for peak power reduction (i.e., the task partitioning and DVFS solution that result in the minimum peak power consumption). 88, 89, 108–110, 116

$\mathbf{AF}^{\textbf{energy}}_{\textbf{DLTF-SVA}}$ Approximation factor (i.e., the worst-case behavior) of DLTF-SVA for energy minimization in relation to the optimal task partitioning and optimal DVFS solution for energy minimization (i.e., the task partitioning and DVFS solution that result in the minimum energy consumption). 88, 89, 111, 112, 117

$\mathbf{AF}^{\textbf{peak power}}_{\textbf{DLTF-SVA}}$ Approximation factor (i.e., the worst-case behavior) of DLTF-SVA for peak power reduction in relation to the optimal task partitioning and optimal DVFS solution for peak power reduction (i.e., the task partitioning and DVFS solution that result in the minimum peak power consumption). 88, 89, 109, 112–114, 117

α For the approximated power consumption on a CMOS core, a constant including the effective switching capacitance, the average activity factor of the core, and a scaling factor for the linear relationship between the voltage of the cluster and the highest frequency in the cluster (i.e., $V_{\text{dd}} \propto f_{\text{cluster}}$). 28–31, 89, 94–117, 123, 126

$\textbf{area}^{\textbf{core}}_{m}$ Area of core m (among all M in the chip). 27

$\textbf{area}^{\textbf{type}}_{q}$ Area of a core of type q. 27, 50, 51

$\mathbf{B}^{-1}$ Matrix $\mathbf{B}^{-1} = [\tilde{b}_{i,j}]_{N \times N}$ is the inverse of matrix $\mathbf{B}$. 33, 34, 36, 46, 50, 182

$\mathbf{B}$ Matrix $\mathbf{B} = [b_{i,j}]_{N \times N}$ that contains the thermal conductance values between vertical and lateral neighboring thermal nodes of an RC thermal network. 33, 36, 62, 63, 68, 181, 182

$\widetilde{b}_{i,j}$ Element in row i and column j inside matrix $\mathbf{B}^{-1}$, such that $1 \leq i \leq N$ and $1 \leq j \leq N$. 33, 34, 43, 46, 49, 50

$b_{i,j}$ Element in row i and column j inside the thermal conductance matrix $\mathbf{B}$, such that $1 \leq i \leq N$ and $1 \leq j \leq N$. 33

β For the approximated power consumption on a CMOS core, $\beta \cdot f_{\text{cluster}} \geq 0$ represents the leakage power consumption on the core. 28–31, 89, 94–99, 101–117, 123

$\mathbf{C}$ Matrix of an RC thermal network, such that $\mathbf{C} = -\mathbf{A}^{-1}\mathbf{B}$. 33, 36, 62, 63, 182, 184, 185

D Hyper-period, i.e., the least common multiple among all periods of all R tasks. 25, 26, 29, 30, 94–99, 101, 102, 123, 125, 126, 128, 131, 145–147, 155, 183, 186

d_n Period and implicit deadline of task τ_n. 25, 26, 29, 30, 89, 91, 92, 145, 146, 189

δ Variable of auxiliary function $U(\delta)$, used to choose a value of f_{dyn} such that $E^*_{\downarrow}$ becomes a continuous function, in order to derive an approximation factor without unnecessary pessimism. 101–105, 109, 110, 112–114, 182

$\delta^{\textbf{max}}$ Value of δ that maximizes auxiliary function $U(\delta)$, used to choose a value of f_{dyn} such that $E^*_{\downarrow}$ becomes a continuous function, in order to derive an approximation factor without unnecessary pessimism. 101–105

DYVIA (i,j) Dynamic programming function, where i is the index of the first task set to be considered in this sub-problem, and j is the index of the last task set to be considered in this sub-problem, such that function $DYVIA(i,j)$ returns the minimum energy consumption for the assignment of task sets $\mathbf{S}_i, \mathbf{S}_{i+1}, \ldots, \mathbf{S}_{j-1}, \mathbf{S}_j$ onto cores, using $v = \frac{j-i+1}{K}$ clusters (from Corollary 1, $j - i + 1$ will always be an integer multiple of K). 130–133, 182

DYVIA$^{\textbf{back-}}_{\textbf{tracking}}$ (i,j) Backtracking table in which entry *DYVIA*$^{\text{back-}}_{\text{tracking}}$ (i,j) contains the task sets indexes $\{\ell_1, \ell_2, \ldots, \ell_K\}$ that resulted in the minimum energy consumption for sub-problem *DYVIA* (i,j). 130, 132, 133

DYVIA$_{\textbf{\#combinations}}$ Total number of combinations that algorithm DYVIA needs to evaluate when building its dynamic programming table. 133

$E_{\textbf{core}}(f)$ Approximated energy consumption on a CMOS core for the case in which the core runs at the same frequency which determines the voltage of the cluster, where f is the execution frequency of the core. 30, 31, 89, 94

$E_{\textbf{core}}(f_{\textbf{cluster}}, f)$ Approximated energy consumption on a CMOS core for the general case of having voltage scaling at a cluster level and frequency scaling at a core level, where f_{cluster} is the highest execution frequency among all cores in the cluster (thus setting the voltage of the cluster), and f is the execution frequency of the core. 30, 89, 96

$e^{\mathbf{C}t}$ Matrix exponential $e^{\mathbf{C}t} = \left[{e^{\mathbf{C}t}}_{i,j}\right]_{N \times N}$. 63

$E^{\textbf{DVFS=optimal}}_{\textbf{ASG=DYVIA}}$ Total energy consumption when using an optimal DVFS algorithm to decide the DVFS levels on individual clusters, and when using DYVIA for assigning task sets to clusters (i.e., the optimal task set assignment solution under SFA). 134

$E^{\textbf{DVFS=optimal}}_{\textbf{ASG=optimal DVFS}}$ Total energy consumption when using an optimal DVFS algorithm to decide the DVFS levels on individual clusters, and when assigning task sets to clusters by using an algorithm that is optimal when using an optimal DVFS algorithm to decide the DVFS levels on individual clusters. 134

$E_{\textbf{ASG=DYVIA}}^{\textbf{DVFS=SFA}}$ Total energy consumption when using SFA to decide the DVFS levels on individual clusters, and when using DYVIA for assigning task sets to clusters (i.e., the optimal task set assignment solution under SFA). 134

$E_{\textbf{ASG=optimal DVFS}}^{\textbf{DVFS=SFA}}$ Total energy consumption when using SFA to decide the DVFS levels on individual clusters, and when assigning task sets to clusters by using an algorithm that is optimal when using an optimal DVFS algorithm to decide the DVFS levels on individual clusters. 134

$E_{j\,\textbf{ASG=CCH}}^{\textbf{DVFS=SFA}}$ Energy consumption of cluster I_j when using the CCH task partition mapping algorithm to map task sets to cores and clusters, and when using SFA to decide the DVFS levels on individual clusters. 125

$E_{j\,\textbf{ASG=ANY}}^{\textbf{DVFS=per-core}}$ Energy consumption of cluster I_j when using *any* task partition mapping algorithm to map task sets to cores and clusters, and when having per-core DVFS, which will result in the lower bound for the energy consumption since having per-core DVFS is the optimal solution and the task set assignment plays no role for such a case. 125

$E_{j\,\textbf{ASG=ANY}}^{\textbf{DVFS=SFA}}$ Energy consumption of cluster I_j when using *any* task partition mapping algorithm to map task sets to cores and clusters, and when using SFA to decide the DVFS levels on individual clusters. 125

$E_{j\,\textbf{ASG=SFA}}^{\textbf{DVFS=SFA}}$ Energy consumption of cluster I_j when using SFA to decide the DVFS levels on individual clusters, and when using the optimal task set assignment solution under SFA. 125

$e_{q,n}$ Worst-case execution cycles of task τ_n when being executed on a core of type q. 25, 26, 29, 30, 89, 91, 92, 145, 146, 189, 190

$E\left(\mathbf{L}\right)$ Energy consumption of the *highest DVFS level cluster* for each combination, which is similar to Equation (9.1), but for set $\mathbf{L}$ instead of set $\mathbf{L}_j$. 131

$E_{\downarrow}^{*}$ Lower bound for the optimal energy consumption for the optimal task partition and any feasible DVFS schedule during a hyper-period D. 88, 89, 97, 99–104, 111, 112, 116, 117, 173, 182, 189

$E_{\textbf{OPT}}^{*}$ Optimal energy consumption for the optimal task partition and optimal DVFS schedule during a hyper-period D. 88, 89

$\tilde{E}_{q}^{\tau_n}\left(F_{q,j}^{\textbf{type}}\right)$ Energy consumed during one hyper-period for executing task τ_n on a core of type q at frequency index j (such that $0 \leq j \leq \hat{F}_{q}^{\text{type}}$) in case that all tasks consume *equivalent* power when executing at the same frequency on a core of type q, i.e., $P_q^{\tau_1}\left(F_{q,j}^{\text{type}}\right) = P_q^{\tau_2}\left(F_{q,j}^{\text{type}}\right) = \cdots = P_q^{\tau_R}\left(F_{q,j}^{\text{type}}\right) = P_q\left(F_{q,j}^{\text{type}}\right)$. 146

$\tilde{E}_{q}^{\mathbf{S}_g}\left(F_{q,j}^{\textbf{type}}\right)$ Energy consumed during one hyper-period for executing task set $\mathbf{S}_g$ on a core of type q at frequency index j (such that $0 \leq j \leq \hat{F}_{q}^{\text{type}}$) in case that all tasks consume *equivalent* power when executing at the same frequency on a core of type q, i.e., $P_q^{\tau_1}\left(F_{q,j}^{\text{type}}\right) = P_q^{\tau_2}\left(F_{q,j}^{\text{type}}\right) = \cdots = P_q^{\tau_R}\left(F_{q,j}^{\text{type}}\right) = P_q\left(F_{q,j}^{\text{type}}\right)$. 146

$E_{q}^{\tau_n}\left(F_{q,j}^{\textbf{type}}\right)$ Energy consumed during one hyper-period for executing task τ_n on a core of type q at frequency index j (such that $0 \leq j \leq \hat{F}_{q}^{\text{type}}$). 30, 145

$E_{\textbf{SFA}}^{\textbf{DLTF}}$ Total energy consumption during a hyper-period D for partitioning tasks with DLTF and selecting the DVFS schedule with SFA. 88, 95, 102, 103

$E_{\textbf{SVA}}^{\textbf{DLTF}}$ Total energy consumption during a hyper-period D for partitioning tasks with DLTF and selecting the DVFS schedule with SVA. 89, 96, 111

η Amount of power consumed by a cluster for being in the active state (since there is no voltage regulator with 100% efficiency) when at least one core inside the cluster has to execute some workload. 122, 123, 125, 126, 131, 136, 139

$f_{\textbf{cluster}}$ For the approximated power consumption on a CMOS core, f_{cluster} is the highest execution frequency among all cores in the cluster, and therefore determines the minimum voltage of the cluster for stable execution. 28–31, 89, 95, 96

${f_{\textbf{crit}}}_q^{\tau_n}$ Critical frequency of task τ_n running on a core of type q, that minimizes the energy consumption for execution when the overhead for entering/leaving a low-power mode can be considered negligible. 30, 145

${f_{\textbf{crit}}}_q$ Critical frequency on a core of type q in case that all tasks consume *equivalent* power when executing at the same frequency on a core of type q, i.e., in case that $P_q^{\tau_1}\left(F_{q,j}^{\text{type}}\right) = P_q^{\tau_2}\left(F_{q,j}^{\text{type}}\right) = \cdots = P_q^{\tau_R}\left(F_{q,j}^{\text{type}}\right) = P_q\left(F_{q,j}^{\text{type}}\right)$ such that ${f_{\text{crit}}}_q^{\tau_1} = {f_{\text{crit}}}_q^{\tau_2} \cdots = {f_{\text{crit}}}_q^{\tau_R} = {f_{\text{crit}}}_q$. 146, 184, 187

$f_{\textbf{crit}}$ Critical frequency for the energy model in Equation (3.6) (focusing on homogeneous systems and assuming that all tasks have similar average activity factors), that minimizes the energy consumption for execution when the overhead for entering/leaving a low-power mode can be considered negligible. 31, 89, 91–96, 98, 99, 101–106, 108–114, 116, 122, 123, 125, 128, 129, 131, 136, 190

$f_{\textbf{dyn}}^{\textbf{max}}$ Maximum value of the auxiliary frequency used to obtain an analytical expression for the lower bound of the energy consumption which can be used for general cases. 101, 103, 104

$f_{\textbf{dyn}}$ Auxiliary frequency used to obtain an analytical expression for the lower bound of the energy consumption which can be used for general cases. 99–104, 106, 111, 112, 173, 182, 189

$F_{i,\hat{F}_i^{\textbf{core}}}^{\textbf{core}}$ Maximum frequency for core i. 27, 78

$\hat{F}_i^{\textbf{core}}$ Number of available frequencies for core i. 27, 78

$F_{\textbf{max}}$ Maximum frequency when considering homogeneous systems. 90, 97, 99, 109, 114, 116, 122

$F_{\textbf{min}}$ Minimum frequency when considering homogeneous systems. 90, 116, 122

$F_{q,\hat{F}_q^{\textbf{type}}}^{\textbf{type}}$ Maximum frequency for a core of type q. 27, 90, 115, 144

$\hat{F}_q^{\textbf{type}}$ Number of available frequencies for cores of type q. 27–29, 144, 145, 154, 183, 187

$\mathbf{G}$ Column vector $\mathbf{G} = [g_i]_{N\times 1}$ that contains the values of the thermal conductances between each thermal node and the ambient temperature of an RC thermal network. 33, 34, 36, 46, 62, 68

$\mathbf{\Gamma}^{-1}$ Matrix $\mathbf{\Gamma}^{-1} = [\tilde{\Gamma}_{i,j}]_{N\times N}$ represents the inverse of matrix $\mathbf{\Gamma}$. 63, 65

$\mathbf{\Gamma}$ Matrix $\mathbf{\Gamma} = [\Gamma_{i,j}]_{N\times N}$ represents a matrix containing the eigenvectors of matrix $\mathbf{C}$ for a given thermal model. 63, 65, 184

γ For the approximated power consumption on a CMOS core, $\gamma > 1$ is a constant related to the hardware (in CMOS processors, γ is generally modeled equal to 3). 28–31, 89, 94–117, 123

$\mathbf{H}^\rho$ Auxiliary matrix $\mathbf{H}^\rho = \left[h_{q,i,j}^\rho\right]_{Q\times Z\times M_q^{\text{type}}}$, used to compute the maximum amount of heat that any m_q cores of type q can contribute to the temperature on node i, for all core types $q = 1, 2, \ldots, Q$. 50, 51

$\mathbf{H}$ Auxiliary matrix $\mathbf{H} = [h_{i,j}]_{Z\times M}$, used to compute the maximum amount of heat that *any* m cores can contribute to the steady-state temperature on thermal node i. 46–48

$I_{\textbf{max}}$ Maximum chip current constraint for the entire chip that cannot be exceeded (e.g., from the capacity of the power supply or the wire thickness). 28, 76–78, 80

$\mathbf{K}$ Set $\mathbf{K} = \{k_1, k_2, \ldots, k_M\}$ that contains all the indexes of the thermal nodes that correspond to cores (among all cores, ignoring the types of the cores). 34, 43, 46, 47

K Total number of cores inside every cluster/island, for a system in which all the clusters have equal number of cores per cluster/island. 122–133, 138–141, 181, 182, 185

$\mathbf{K}^q$ Set $\mathbf{K}^q = \left\{ k_1^q, k_2^q, \ldots, k_{M_q^{\text{type}}}^q \right\}$ for all core types $q = 1, 2, \ldots, Q$, that contains the indexes of the thermal nodes that correspond to cores of type q. 34, 50

κ For the approximated power consumption on a CMOS core, $\kappa \geq 0$ represents the independent power consumption attributed to maintaining the core in execution mode (i.e., the voltage and frequency independent part of the power consumption). 28–31, 89, 94–99, 101–117, 122, 123, 139

$\mathbf{L}$ Set $\mathbf{L} = \{\ell_1, \ell_2, \ldots, \ell_Z\}$ that includes all the indexes of the thermal nodes that correspond to blocks in the floorplan (as opposed to thermal nodes that represent the heat sink, internal nodes of the heat spreader, the thermal interface material, etc.). 34, 41, 43, 45, 46, 49, 50, 75, 77, 80, 130

λ Lagrange multiplier inside the Lagrangian used when applying the Kuhn-Tucker conditions. 98, 99

$\mathcal{L}$ Lagrangian used when applying the Kuhn-Tucker conditions. 98

$\mathbf{L}$ Set containing the indexes of the task sets assigned to a general cluster (as opposed to set $\mathbf{L}_j$, defined for the particular cluster I_j), such that $\ell_1 < \ell_2 < \cdots < \ell_K$, with ℓ_0 auxiliary and less than ℓ_1. 130–132, 183, 185

$\mathbf{L}_j$ Set containing the indexes of the task sets assigned to cluster I_j such that $\ell_{j,1} < \ell_{j,2} < \cdots < \ell_{j,K}$ for all $j = 1, 2, \ldots, V$, it holds that $\ell_{j,i} \in [1, M]$, and $\ell_{j,i}$ is unique for all j, i. 122, 123, 125, 129–131, 183, 185

$\Lambda\left(i, j\right)$ Set that contains all possible $\mathbf{L}$ sets that satisfy Theorem 7, i.e., $\Lambda\left(i, j\right)$ stores all the *potentially optimal combinations*, such that $\ell_0 = i - 1$, $\ell_K = j$ and $\ell_h = \ell_{h-1} + 1 + n \cdot K$ for $0 < h < K$ with $\ell_h < j$ and $n \in \mathbb{N}^0$. 130–132

$\mathbf{\Lambda}$ Diagonal matrix $\lambda = \operatorname{diag}\left(e^{\lambda_1 \cdot t}, e^{\lambda_2 \cdot t}, \ldots, e^{\lambda_N \cdot t}\right)$, where $\lambda_1, \lambda_2, \ldots, \lambda_N$ are the eigenvalues of matrix $\mathbf{C}$ for a given thermal model. 63

M Total number of cores in the system. 25–28, 34, 43, 44, 46–49, 51, 55, 88, 121–124, 126, 127, 129, 131–133, 139–141, 144, 150, 181, 184–186, 190

$M^\star$ Total number of task sets in which some task partitioning algorithm partitions the tasks. 122, 123, 125, 126, 136, 139–141, 181

M Total number of task sets in which we partition the tasks. 25, 26, 89–119, 173, 189, 190

$\mathbf{m}$ Set $\mathbf{m} = \{m_1, m_2, \ldots, m_Q\}$ that represents the number of active cores for core types $\{1, 2, \ldots, Q\}$, respectively. 41, 42, 49–51, 186, 187

$M_k^{\textbf{cluster}}$ Total number of cores inside cluster/island k. 26, 88, 122, 144, 145, 147, 150, 185

$M_{\textbf{max}}^{\textbf{cluster}}$ Maximum number of cores inside a cluster among all clusters, i.e., $M_{\max}^{\text{cluster}} = \max_{1 \leq k \leq V} \left\{M_k^{\text{cluster}}\right\}$. 150, 152, 154, 185

$M^{\neq 0}$ When partitioning tasks using DLTF, the resulting number of cores after regrouping with cycle utilization larger than 0, i.e., the cores that remain active. 92, 94, 96, 108–114, 116, 118

$M_q^{\textbf{type}}$ Total number of cores of type q. 26, 27, 34, 50, 51, 184, 185, 190

N Total number of thermal nodes in the RC thermal network, such that there are at least as many thermal nodes in the RC thermal network as blocks in the floorplan, i.e., $N \geq Z$. 32–34, 43, 44, 46–51, 63–67, 75, 77, 181, 182, 184–189

$\mathbf{\Omega}$ Auxiliary matrix $\mathbf{\Omega} = [\Omega_{i,j}]_{N\times N}$, used to speed-up the computation of the transient temperatures in MatEx. 64, 65, 67, 69, 186

$\Omega_{k,i}$ Element in row k and column i inside auxiliary matrix $\mathbf{\Omega}$, such that $1 \leq k \leq N$ and $1 \leq i \leq N$. 64–67

$\mathbf{P}$ Column vector $\mathbf{P} = [p_i]_{N\times 1}$ that contains the values of the power consumption on every node of an RC thermal network. 33, 34, 62–65

$\mathbf{P}^{\text{blocks}}$ Column vector $\mathbf{P}^{\text{blocks}} = \left[p_i^{\text{blocks}}\right]_{N\times 1}$ that represents the power consumption on other blocks in the floorplan that do not correspond to cores, (e.g., a block of an L2 cache). 34, 42–44, 46, 47, 49

$P_{\textbf{core}}(f)$ Approximated average power consumption on a CMOS core for the case in which the core runs at the same frequency which determines the voltage of the cluster, where f is the execution frequency of the core. 29, 89, 94, 116, 122, 123, 126, 128, 129

$P_{\textbf{core}}(f_{\textbf{cluster}}, f)$ Approximated average power consumption on a CMOS core for the general case of having voltage scaling at a cluster level and frequency scaling at a core level, where f_{cluster} is the highest execution frequency among all cores in the cluster (thus setting the voltage of the cluster), and f is the execution frequency of the core. 28, 89, 96

$P^{\textbf{core}}_{\textbf{inact}j}$ Power consumption of core j (among all M in the chip) when the core is inactive (i.e., idle or in a low-power mode). 49, 51

$P^{\textbf{core}}_{\textbf{inact}m}$ Power consumption of core m (among all M in the chip) when the core is inactive (i.e., idle or in a low-power mode). 28

$P^{\textbf{core}}_{\textbf{inact}}$ Power consumption of an inactive core (i.e., idle or in a low-power mode) for the special case of homogeneous manycore systems. 28, 43, 44, 46, 47, 51

$P^{\textbf{core}\rho}_{\textbf{max}}(\mathbf{m})$ Auxiliary function used to assist in deriving the amount of power density that any $\mathbf{m} = \{m_1, m_2, \ldots, m_Q\}$ active cores are allowed to consume, such that the total power consumption precisely reaches the value of P_{max}. 51

$P^{\textbf{core}\rho}_{\textbf{max}}(\mathbf{X})$ Auxiliary function used to assist in deriving the amount of power density that the active cores in mapping $\mathbf{X}$ are allowed to consume, such that the total power consumption precisely reaches the value of P_{max}. 49

$P^{\textbf{core}}_{\textbf{max}}(m)$ Auxiliary function used to assist in deriving the amount of power that any m active cores are allowed to consume, such that the total power consumption precisely reaches the value of P_{max}. 44, 47, 48

$\mathbf{P}^{\textbf{cores}}$ Column vector $\mathbf{P}^{\text{cores}} = [p_i^{\text{cores}}]_{N\times 1}$ represents the power consumption on the cores. 34

$P^{\rho}_{\textbf{equal}}$ For a heterogeneous or homogeneous manycore system, power density on all active cores when we assume that all active cores have equal power density at a given point in time. 49, 50

$P_{\textbf{equal}}$ For a homogeneous manycore system, power consumption of all active cores when we assume that all active cores are consuming equal power at a given point in time. 43–48, 57

$\hat{P}^*_{\downarrow}$ Lower bound for the optimal peak power consumption for the optimal task partition and any feasible DVFS schedule during a hyper-period D. 88, 89, 102, 108, 112, 116, 117

$P_{\textbf{max}}$ Maximum chip power constraint for the entire chip that cannot be exceeded (e.g., from the capacity of the power supply or the wire thickness). 28, 36, 42–44, 47–51, 76–78, 80, 186, 187

$\hat{P}^*_{\textbf{OPT}}$ Optimal peak power consumption for the optimal task partition and optimal DVFS schedule during a hyper-period D. 88, 89

$P_q\left(F_{q,j}^{\textbf{type}}\right)$ Average power consumption on a core of type q running at frequency index j (such that $0 \leq j \leq \hat{F}_q^{\text{type}}$), in case that all tasks consume *equivalent* power when executing at the same frequency on a core of type q, i.e., in case that $P_q^{\tau_1}\left(F_{q,j}^{\text{type}}\right) = P_q^{\tau_2}\left(F_{q,j}^{\text{type}}\right) = \cdots = P_q^{\tau_R}\left(F_{q,j}^{\text{type}}\right) = P_q\left(F_{q,j}^{\text{type}}\right)$ such that ${f_{\text{crit}}}_q^{\tau_1} = {f_{\text{crit}}}_q^{\tau_2} \cdots = {f_{\text{crit}}}_q^{\tau_R} = {f_{\text{crit}}}_q$. 146, 183, 184, 187

$P_q^{\tau_n}\left(F_{q,j}^{\textbf{type}}\right)$ Average power consumption on a core of type q when executing task τ_n at frequency index j (such that $0 \leq j \leq \hat{F}_q^{\text{type}}$). 28–30, 122, 145, 153

$\mathbf{P}^{\textbf{rest}}$ Column vector $\mathbf{P}^{\text{rest}} = [p^{\text{rest}}]_{N\times 1}$ that represents the power consumption of thermal nodes that are on the floorplan (e.g., internal thermal nodes of the heat sink), for which it holds that $p_i^{\text{rest}} = 0$ for all i. 34

$\hat{P}_{\textbf{SFA}}^{\textbf{DLTF}}$ Total peak power consumption for partitioning tasks with DLTF and selecting the DVFS schedule with SFA. 88, 89, 94, 108

$\hat{P}_{\textbf{SVA}}^{\textbf{DLTF}}$ Total peak power consumption for partitioning tasks with DLTF and selecting the DVFS schedule with SVA. 89, 96, 112

$P_{\textbf{TSP}}^{\rho}(\mathbf{X})$ Per-core power density budget for each active core in the specified core mapping (independent of the type of core), that results in a maximum steady-state temperature among all cores which does not exceed the critical threshold temperature for triggering DTM, for heterogeneous or homogeneous manycore systems. 41, 48, 49

$P_{\textbf{TSP}}(\mathbf{X})$ Per-core power budget for each active core in the specified core mapping, that results in a maximum steady-state temperature among all cores which does not exceed the critical threshold temperature for triggering DTM, for homogeneous manycore systems. 41–45

$P_{\textbf{TSP}}^{\rho\star}(\mathbf{X})$ Per-core power density budget for each active core in the specified core mapping (independent of the type of core) while ignoring P_{max}, that results in a maximum steady-state temperature among all cores which does not exceed the critical threshold temperature for triggering DTM, for heterogeneous or homogeneous manycore systems. 49

$P_{\textbf{TSP}}^{\star}(\mathbf{X})$ Per-core power budget for each active core in the specified core mapping while ignoring P_{max}, that results in a maximum steady-state temperature among all cores which does not exceed the critical threshold temperature for triggering DTM, for homogeneous manycore systems. 42–44

$P_{\textbf{TSP}}^{\rho\,\textbf{worst}}(\mathbf{m})$ Per-core power density budget (independent of the type of core) for each active core in *any* possible core mapping with $\mathbf{m} = \{m_1, m_2, \ldots, m_Q\}$ active cores for core types $\{1, 2, \ldots, Q\}$, respectively, that results in a maximum steady-state temperature among all cores which does not exceed the critical threshold temperature for triggering DTM, for heterogeneous or homogeneous manycore systems. 42, 49, 51

$P_{\textbf{TSP}}^{\textbf{worst}}(m)$ Per-core power budget for each active core in *any* possible core mapping with m simultaneously active cores, that results in a maximum steady-state temperature among all cores which does not exceed the critical threshold temperature for triggering DTM, for homogeneous manycore systems. 41, 44, 45, 47, 48, 55

$P_{\textbf{TSP}}^{\rho\star\textbf{worst}}(\mathbf{m})$ Per-core power density budget (independent of the type of core) for each active core in *any* possible core mapping with $\mathbf{m} = \{m_1, m_2, \ldots, m_Q\}$ active cores for core types $\{1, 2, \ldots, Q\}$, respectively, that results in a maximum steady-state temperature among all cores which does not exceed the critical threshold temperature for triggering DTM while ignoring P_{max}, for heterogeneous or homogeneous manycore systems. 50, 51

$P_{\textbf{TSP}}^{\star\textbf{worst}}(m)$ Per-core power budget for each active core in *any* possible core mapping with m simultaneously active cores while ignoring P_{max}, that results in a maximum steady-state temperature among all cores which does not exceed the critical threshold temperature for triggering DTM, for homogeneous manycore systems. 47, 48

$P^{\textbf{type}}_{\textbf{inact}q}$ Power consumption of a core of type q when the core is inactive (i.e., idle or in a low-power mode). 28, 50, 51, 145

Q Number of different types of cores in the system. 26, 27, 34, 41, 42, 49–51, 144, 146–148, 150, 152–154, 184–187, 190

$Q^{\square}_{k}$ Index identifying the types of cores inside cluster/island k. 26, 144–146

R Total number of tasks to be executed. 25, 26, 88, 90, 92, 122, 139, 146, 149, 150, 152–155, 182–184, 187, 190

$\rho^{\textbf{energy}}_{\textbf{SFA}}$ Worst-case energy consumption and peak power consumption ratio for SFA when we consider discrete DVFS levels against the continuous cases. 116

$\rho^{\textbf{peak power}}_{\textbf{SFA}}$ Worst-case energy consumption and peak power consumption ratio for SFA when we consider discrete DVFS levels against the continuous cases. 116

$\rho_{\textbf{SVA}}$ Worst-case energy consumption and peak power consumption ratio for SVA when we consider discrete DVFS levels against the continuous cases. 116, 117

$\mathbf{S}^{\textbf{DLTF}}_{i}$ Task set assigned to core i after partitioning the tasks using the DLTF strategy. 91, 92, 95

$\mathbf{S}^{\textbf{LTF}}_{i}$ Task set assigned to core i after partitioning the tasks using the LTF strategy. 90

$\mathbf{S}_i$ Task set assigned to core i. 26, 123, 130, 131, 182, 189, 190

$\mathbf{T}'$ Column vector $\mathbf{T}' = [T'_i(t)]_{N\times1}$ that accounts for the first-order derivative of the temperature on each thermal node of an RC thermal network with respect to time. 33, 62, 188

$\mathbf{T}$ Column vector $\mathbf{T} = [T_i(t)]_{N\times1}$ that represents the temperatures on the thermal nodes of an RC thermal network. 33, 62, 63, 188

$T_{\textbf{amb}}$ Ambient temperature. 32–34, 43, 44, 46, 47, 49, 50, 62–64, 75

$T_{\textbf{DTM}}$ Critical temperature which serves as thermal threshold for triggering DTM. 36, 41–45, 47–50, 52–54, 56, 59, 75–80

$T'_i(t)$ Element in row i of the column vector $\mathbf{T}'$, i.e., the first-order derivate of the temperature on thermal node i with respect to time, such that $1 \leq i \leq N$. 33

$T'_k(t)$ Element in row k of the column vector $\mathbf{T}'$, i.e., the first-order derivate of the temperature on thermal node k with respect to time, such that $1 \leq k \leq N$. 66, 67, 172

$T_{\textbf{steady}i}$ Element in row i of the column vector $\mathbf{T}_{\text{steady}}$, i.e., the steady-state temperature on thermal node i, such that $1 \leq i \leq N$. 33, 34, 41, 43, 45–47, 49, 50

$T_{\textbf{steady}k}$ Element in row k of the column vector $\mathbf{T}_{\text{steady}}$, i.e., the steady-state temperature on thermal node k, such that $1 \leq k \leq N$. 64–68, 75, 77, 80

$T_i(t)$ Element in row i of the column vector $\mathbf{T}$, i.e., the temperature on thermal node i, such that $1 \leq i \leq N$. 33

$T_k(t)$ Element in row k of the column vector $\mathbf{T}$, i.e., the temperature on thermal node k, such that $1 \leq k \leq N$. 62–67, 77

$T_{\textbf{init}k}$ Element in row k of the column vector $\mathbf{T}_{\text{init}}$, i.e., the initial temperature on thermal node k at time zero, such that $1 \leq k \leq N$. 68

$\mathbf{T}_{\textbf{init}}$ Column vector $\mathbf{T}_{\text{init}} = [T_{\text{init}k}]_{N\times1}$ that contains the initial temperatures on all nodes of an RC thermal network at time zero. 33, 63–65, 188

$T_k\left(t^{\uparrow k}\right)$ Maximum temperature on thermal node k, which occurs at time $t^{\uparrow k}$. 63, 65, 67, 68

$T_k''(t)$ Second-order derivate of the temperature on thermal node k with respect to time, such that $1 \leq k \leq N$. 67

$t^{\uparrow k}$ Time point in which the temperature on thermal node k reaches it maximum value. 63, 65–67, 172, 189

$\mathbf{T}_{\textbf{steady}}$ Column vector $\mathbf{T}_{\text{steady}} = \left[T_{\text{steady}_i}\right]_{N\times 1}$ that represents the steady-state temperatures on the thermal nodes of an RC thermal network. 33, 34, 63–65, 188

τ_n Periodic real-time task n with implicit deadline. 25, 26, 28–30, 89–92, 122, 145, 146, 149, 150, 153, 182–184, 187, 189

$\theta_{\textbf{LTF}}$ Approximation factor of the LTF strategy in terms of task partitioning, due to the approximation factor of the LPT algorithm for the *Makespan* problem. 90, 91, 93, 100, 104–107, 109, 110, 112, 114

$U\left(\delta^{\textbf{max}}\right)$ Maximum value of auxiliary function $U(\delta)$, used to choose a value of f_{dyn} such that $E_{\downarrow}^{*}$ becomes a continuous function, in order to derive an approximation factor without unnecessary pessimism. 101, 104–107, 173

$U(\delta)$ Auxiliary function used to choose a value of f_{dyn} such that $E_{\downarrow}^{*}$ becomes a continuous function, in order to derive an approximation factor without unnecessary pessimism. 101–105, 109, 110, 112–114, 173, 182, 189

$u_{\tau_n}(t)$ Instantaneous activity factor of a core executing task τ_n at time t. 28

V Total number of clusters/islands in the system. 26, 121–133, 138–141, 144–146, 148, 150, 152–154, 185, 190

$V_{\textbf{dd}}$ Supply voltage of a core. 10, 28, 181

$V_{\textbf{th}}$ Transistor threshold voltage. 10, 28

W Number of iterations used when applying the Newton-Raphson method. 67

w_i' Cycle utilization (unit $\frac{\text{cycles}}{\text{second}}$) of task set $\mathbf{S}_i$ that results in a lower bound of the optimal energy consumption for the optimal task partition and the optimal DVFS schedule for homogeneous systems. 100–105, 108, 110–114

w_i^* Cycle utilization (unit $\frac{\text{cycles}}{\text{second}}$) of task set $\mathbf{S}_i$ that results in the optimal energy consumption for the optimal task partition and the optimal DVFS schedule for homogeneous systems. 89, 97, 99, 100, 104, 105, 110, 112, 114

$w_i^{\textbf{DLTF}}$ Cycle utilization (unit $\frac{\text{cycles}}{\text{second}}$) of task set $\mathbf{S}_i$ when partitioning tasks using the DLTF strategy for homogeneous systems. 91, 92, 94–96, 103, 106, 108–114, 116

$w_i^{\textbf{LTF}}$ Cycle utilization (unit $\frac{\text{cycles}}{\text{second}}$) of task set $\mathbf{S}_i$ when partitioning tasks using the LTF strategy for homogeneous systems. 90, 92, 105, 109, 110

$w_{q,i}$ Cycle utilization (unit $\frac{\text{cycles}}{\text{second}}$) of task set $\mathbf{S}_i$ running on a core of type q, computed as $\sum_{\tau_n \in \mathbf{S}_i} \frac{e_{q,n}}{d_n}$. 26

w_M' Maximum cycle utilization (unit $\frac{\text{cycles}}{\text{second}}$) among all task sets $\mathbf{S}_i$ that results in a lower bound of the optimal energy consumption for the optimal task partition and the optimal DVFS schedule for homogeneous systems. 99–106, 108–114, 173

w_M^* Maximum cycle utilization (unit $\frac{\text{cycles}}{\text{second}}$) among all task sets $\mathbf{S}_i$ that results in the optimal energy consumption for the optimal task partition and the optimal DVFS schedule for homogeneous systems. 89–93, 98–101, 104–106, 108–114, 173

$w_M^{\textbf{DLTF}}$ Maximum cycle utilization (unit $\frac{\text{cycles}}{\text{second}}$) among all task sets $\mathbf{S}_i$ when partitioning tasks using the DLTF strategy for homogeneous systems. 91–96, 99, 100, 102–106, 108–114, 116, 117, 173

$w_M^{\textbf{LTF}}$ Maximum cycle utilization (unit $\frac{\text{cycles}}{\text{second}}$) among all task sets $\mathbf{S}_i$ when partitioning tasks using the LTF strategy for homogeneous systems. 90–93, 95, 105, 108–110, 173

w_M Maximum cycle utilization (unit $\frac{\text{cycles}}{\text{second}}$) among all task sets $\mathbf{S}_i$ running on a homogeneous system. 122, 124, 126

$w_{\textbf{max}}^{\textbf{DLTF}}$ Auxiliary cycle utilization used as a maximum cycle utilization for the task regrouping procedure of DLTF, computed as $w_{\text{max}}^{\text{DLTF}} = \max\left\{f_{\text{crit}}, w_M^{\text{DLTF}}\right\}$. 91, 92, 108

$\mathbf{X}$ Column vector $\mathbf{X} = [x_i]_{M \times 1}$ for a particular mapping of *active* cores (among all cores, ignoring the types of the cores), where $x_i = 1$ means that core i corresponds to an *active* core, while $x_i = 0$ means that core i corresponds to an *inactive* core. 27, 41–46, 48, 49, 186

$\mathbf{X}^q$ Column vector $\mathbf{X}^q = [x_i^q]_{M_q^{\text{type}} \times 1}$ for all types of cores $q = 1, 2, \ldots Q$, which represents a particular mapping of *active* cores of type q, where $x_i^q = 1$ means that core i corresponds to an *active* core of type q, while $x_i^q = 0$ means that core i corresponds to an *inactive* core of type q. 27

χ_q Constant scaling factor for the worst-case execution cycles of all tasks executing on cores of type q in relation to the worst-case execution cycles of the same tasks executing on an arbitrary reference type of core y, such that $e_{q,n} = \alpha_q \cdot e_{y,n}$ for all core types $q = 1, 2, \ldots, Q$ and for all tasks $n = 1, 2, \ldots, R$. 146–148, 150

$\mathbf{Y}$ Set $\mathbf{Y} = \{y_1, y_2, \ldots, y_V\}$ containing the indexes of the task sets with the highest cycle utilizations mapped to every cluster, such that task set $\mathbf{S}_{y_j}$ will have the highest cycle utilization inside cluster I_j. 127–130

Z Total number of blocks in the floorplan, such that $Z - M$ is the number of blocks corresponding components other than cores, e.g., L2 caches and memory controllers. 27, 32, 34, 44, 46–51, 184, 185, 190