Design Space Exploration and Run-Time Adaptation for Multicore Resource Management Under Performance and Power Constraints

10

Santiago Pagani, Muhammad Shafique, and Jörg Henkel

Abstract

This chapter focuses on resource management techniques for performance or energy optimization in multi-/many-core systems. First, it gives a comprehensive overview about resource management in a broad perspective. Secondly, it discusses the possible optimization goals and constraints of resource management techniques: computational performance, power consumption, energy consumption, and temperature. Finally, it details the state-of-the-art techniques on resource management for performance optimization under power and thermal constraints, as well as for energy optimization under performance constraints.

Acronyms

DPM	Dynamic Power Management
DSE	Design Space Exploration
DSP	Digital Signal Processor
DTM	Dynamic Thermal Management
DVFS	Dynamic Voltage and Frequency Scaling
EOH	Extremal Optimization meta-Heuristic
EWFD	Equally-Worst-Fit-Decreasing
GIPS	Giga-Instruction Per Second
GPP	General-Purpose Processor
ILP	Instruction-Level Parallelism
IPC	Instructions Per Cycle
IPS	Instruction Per Second
ISA	Instruction-Set Architecture
ITRS	International Technology Roadmap for Semiconductors
LTF	Largest Task First

S. Pagani (✉) • M. Shafique • J. Henkel
Chair for Embedded Systems (CES), Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
e-mail: pagani@kit.edu; shafique@kit.edu; henkel@kit.edu

S. Ha, J. Teich (eds.), *Handbook of Hardware/Software Codesign*,
DOI 10.1007/978-94-017-7267-9_11

MPSoC	Multi-Processor System-on-Chip
NoC	Network-on-Chip
QoS	Quality of Service
SCC	Single Chip Cloud computer
SFA	Single Frequency Approximation
SoC	System-on-Chip
TDP	Thermal Design Power
TLP	Thread-Level Parallelism
TSP	Thermal Safe Power

Contents

10.1 Introduction

In the past decade, single-core processors have reached a practical upper limit with respect to their maximum operational frequency, mostly due to power dissipation. This has motivated chip manufacturers to shift their focus toward designing processors with multiple cores which operate at lower frequencies than their single-core counterparts, such that they can potentially achieve the same computational performance while consuming less power. Furthermore, computational performance demands of modern applications have substantially increased and can no longer be satisfied only by increasing the frequency of a single-core processor or by customizing such a processor. In other words, modern computing systems require processors with multiple cores in the same chip (expected to increase in number every year, as shown in Fig. 10.1), which can efficiently communicate with each other and provide increased parallelism. The main idea is therefore to consider an application as a group of many small tasks, such that these tasks can be executed in parallel on multiple cores and thus meet the increased performance demands [1].

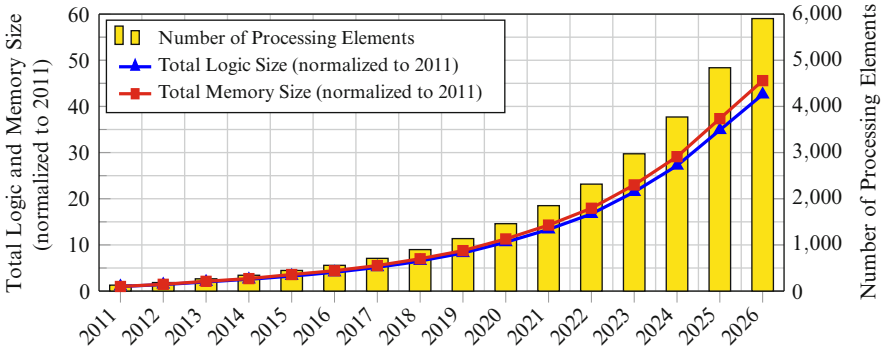

Fig. 10.1 System-on-Chip (SoC) consumer portable design complexity trends [24]. The figure shows the expected total logic and memory sizes (normalized to 2011), as well as the expected number of processing elements to be found in future SoCs, as predicted by ITRS in 2001

Aside for motivating the use of multi-core processors, the continuous increasing performance demands and power budget constraints have led to the emergence of heterogeneous multi-core processors composed by more than one type of core, where each core type has different performance and power characteristics [62]. The distinct architectural features of the different types of cores can be potentially exploited in order to meet both the functional and nonfunctional requirements (e.g., computational performance, power consumption, temperature, etc.). This makes heterogeneous multi-core and many-core systems a very promising alternative over their homogeneous counterpart, where an application may witness large improvements in power and/or performance when mapped to a suitable type of core, as also discussed in ▸ Chap. 8, "Architecture and Cross-Layer Design Space Exploration".

Furthermore, as nanotechnology evolves, it is expected that in upcoming years thousands of cores will be integrated on the same chip [24]. Such a large number of cores need a Network-on-Chip (NoC) based on an efficient and scalable interconnection infrastructure for core-to-core communication (as also discussed in ▸ Chap. 15, "Network-on-Chip Design"). To efficiently manage these processors, we need sophisticated *resource and power management* techniques, which can be categorized as *"centralized or distributed"* (with respect to their view of the system and information exchange during the management decision), and as *"design time or run time"* (with respect to the time of the decision making algorithms). Resource and power management techniques are responsible for mapping threads/tasks to specific resources on the chip (i.e., cores of different types, accelerators, remote servers, etc.), migrating threads/tasks at run time, managing the power modes of the resources (active, clock gated, sleep, power gated, etc.), selecting the voltage and frequency levels of the resources, etc. Considering all these management options leads to a very large design space from which to choose. For example, when executing N threads/tasks on a system with M cores, in which each core can run at F different voltage/frequency levels, there are $M^N + F^M$ possible mapping

and voltage/frequency combinations. According to the desired goals and given constraints, the appropriate combination can be therefore selected applying Design Space Exploration (DSE). Furthermore, most run-time techniques (and also several design-time techniques) considerably limit the design space in order to reduce the execution time of the management decisions.

10.1.1 Centralized and Distributed Techniques

Centralized techniques assume to have (or be able to obtain) a global view of the system. Namely, a centralized resource management technique would require to know what applications are being executed throughout the chip, in how many threads, mapped to which specific cores, executing at what voltage and frequency level, their power consumption, etc. With this information, a centralized technique can potentially arrive to very efficient resource management decisions. However, if such detailed information is not known by the centralized manager a priori (most likely scenario in real systems), then it needs to be transmitted through the corresponding communication infrastructure, requiring high communication bandwidths. This is the main reason why centralized techniques are generally not scalable, where scalability is defined as the ability of a technique to remain within feasible computation and communication constraints when the problem size grows (i.e., when we have an increasing number of cores on a processor). Therefore, *centralized techniques are well suited for processors with a limited number of cores*.

In practice however, a more realistic assumption, in terms of latency and communication bandwidth, is to assume that every core is restricted to a local view of the system where only information of the immediate surrounding neighborhood is known at any given time. This can be easily achieved by having cores periodically communicate with their neighbors, keeping the communication load distributed throughout the chip. In this way, the resource management decisions are made in a **distributed** fashion, where small groups of cores conduct local optimization, maintaining scalability. This local optimization is managed differently depending on the considered distributed technique. For example, each core could manage itself individually after exchanging information with other cores, or contrarily, some cores could act as local managers and make decisions that affect a small group of neighboring cores. In either case, the challenge for distributed techniques is to make high-quality resource management decisions with respect to the chosen optimization goal with such limited local information.

In summary, mainly due to scalability issues, distributed techniques are better suited for large multi-/many-core systems than centralized techniques. Specifically, in distributed systems:

(*continued*)

- The communication load is balanced throughout the processor, avoiding communication bottlenecks in the central manager.
- The computational complexity of the each distributed (local) resource management decision is much smaller than the complexity of a centralized decision, and therefore the problem size is decoupled from the total number of cores in the chip.

10.1.2 Design-Time Decisions and Run-Time Adaptations

Resource management and power management methodologies based on design-time decisions require to have advance knowledge of the applications to execute and their requirements, as well as an estimation of their characteristics (execution time, power consumption, etc.). Naturally, they have a global view of the system and are therefore generally implemented in a centralized manner. Furthermore, the execution time of design-time optimization algorithms is not of major importance, as long as it remains below reasonable limits (e.g., in terms of minutes or hours and not months or years). On the other hand, methodologies based on run-time adaptations only have information of the current application queue (i.e., list of applications ready for execution) and requirements. Moreover, the execution time of run-time optimization algorithms (also known as on-the-fly algorithms) contributes to the overall application execution time, and therefore, it is vital that they have a short execution time. Given that having a global knowledge of the system can potentially consume a high communication bandwidth and also requires considerable time, run-time methodologies are usually restricted to a local view of the system where only information of the immediate surrounding neighborhood may be available and are therefore generally implemented in a distributed manner.

In more detail, methodologies based on **design-time decisions**:

- Have a global view of the system.
- Do not have stringent timing constraints and therefore can involve complex dynamic programming algorithms [25, 44], integer linear programming [26], time-consuming heuristics (e.g., simulated annealing [33, 40] or genetic algorithms [11]), or a large DSE.
- Can generally result in higher quality decisions than run-time methodologies.
- Require previous knowledge (or predictions) of the system behavior, which is not always feasible to obtain a priori.
- Cannot adapt to changes in the system behavior, for example, the insertion of a new application unknown at design time.

Contrarily, methodologies based on **run-time (or on-the-fly) adaptations**:

- Only have information of the current application queue and requirements.
- Generally, restricted to a local view of the system having only information of the immediate surrounding neighborhood.
- Optimization decisions require a short execution time.
- Trade quality in order to reduce the communication bandwidth and the execution time.
- Generally implemented using (probably distributed) lightweight heuristics.
- Can adapt to changes in the system behavior, for example, to take advantage of dynamic workload variations of the applications (as also discussed in ▸ Chap. 9, "Scenario-Based Design Space Exploration", e.g., early completions, performance requirement changes, etc.) or to execute new applications unknown at design time (even after the delivery of the system to the end-user).
- Can adapt to hardware changes after the production of a System-on-Chip (SoC), for example, permanent hardware failures or core degradations due to aging effects.

Finally, aside from design-time or run-time algorithms, there exist **hybrid** methodologies which partially rely on results previously analyzed at design time and also on run-time adaptations [50, 56, 59, 60, 63, 68]. Namely, based on design-time analysis of the applications (stored on the system for a specific platform), lightweight heuristics can make very efficient run-time decisions that adapt to the system behavior (current applications on the ready queue, available system resources, desired performance, etc.). Such techniques still suffer from some of the downsides of design-time methods, for example, knowing all potential applications to be executed at design time such that they can be properly analyzed on the desired platform. Nevertheless, they can generally result in better resource management decisions than design-time algorithms (as they can adapt to the current system behavior) and than on-the-fly heuristics (as they can make more informed decisions or require less time to evaluate certain alternatives).

10.1.3 Parallel Applications

For an application to be executed in a multi-/many-core system, the application has to be parallelized (or partitioned) into multiple threads/tasks that can be concurrently executed on different cores. Although far from a solved problem, there exist some state-of-the-art application parallelization tools [7, 34] that can be used to take care of the task partitioning and manual analysis, involving finding a set of tasks, adding the required synchronization and inter-task communication to the corresponding tasks, management of the memory hierarchy, verifying the parallelized code in order to ensure a correct functionality [35], etc.

A task binding process is also required for the case of heterogeneous platforms, such that the system can know which tasks can be mapped to which type of cores

and with what cost [61]. Namely, depending on the underlying architecture, it is possible that not all tasks can be mapped to all core types, for example, a task requiring a floating point unit may not be mapped to a core that does not have such a unit. Moreover, the binding process must analyze the implementation costs (e.g., performance, power consumption, and resource utilization) of every task on all the types of cores that support each task, such as a General-Purpose Processor (GPP), a Digital Signal Processor (DSP), or some reconfigurable hardware.

10.2 Optimization Goals and Constraints

There are several possible optimization goals and constraints for resource management techniques: computational performance, power consumption, energy consumption, and temperature. For example, a system could choose to maximize the computational performance under a power or temperature constraint, while another system would prefer to minimize energy consumption under performance constraints.

10.2.1 Computational Performance

In few words, computational performance refers to how fast and efficiently can the system execute a given set of applications. It can be measured in many different ways, for example, application's execution time, throughput, Instructions Per Cycle (IPC), Instruction Per Second (IPS), normalized speed-up factor with respect to a known reference, etc. Generally, IPC and IPS are not well-suited metrics to use in heterogeneous systems, as different types of cores might have different Instruction-Set Architectures (ISAs) or require a different number of instructions to finish the same amount of work.

> Maximizing the *overall system performance* (generic term which can, e.g., refer to maximizing the summation of the weighted throughput of all applications, minimize the longest execution time among all applications, etc.) is generally the most commonly pursued optimization goal. Nevertheless, for applications with hard real-time deadlines, meeting the deadlines can be formulated as satisfying certain performance requirements, and therefore for such cases performance is considered as a constraint.

The execution time of an application (or the resulting performance of the application) will depend on how the application is executed, for example, in how many threads the application is parallelized in, the types of cores to which the application is mapped to, the execution frequency, the utilization of shared resources

(caches, NoC, etc.) by other applications, etc. The application's characteristics also play a major role in its execution time, for example, its Instruction-Level Parallelism (ILP) or its Thread-Level Parallelism (TLP) have a direct impact in how well an application scales with respect to frequency and the number of threads, respectively. Applications with high ILP generally scale well with increasing frequencies, while applications with high TLP generally scale better when parallelized in many threads. For example, Fig. 10.2 shows the execution time and speed-up factors of three applications from the PARSEC benchmark suite [3] with respect to the frequency when executing a single thread. Similarly, Fig. 10.3 shows the execution time and speed-up factors of the same application with respect to the number of parallel threads when executing at 2 GHz. From the figures, we can observe that the impact of the frequency and the number of parallel threads on the speed-up factors are entirely application dependent and that the application's performance will eventually stop scaling properly after a certain number of threads, known as the parallelism wall.

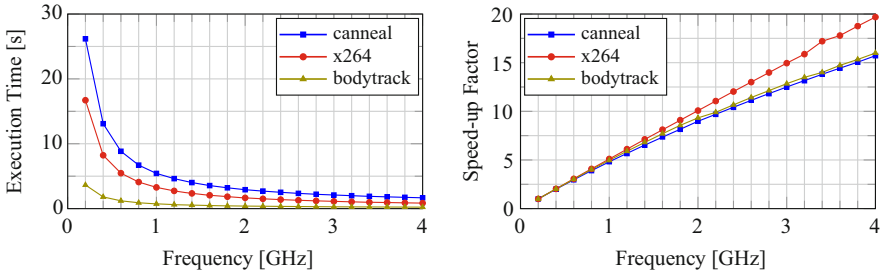

Fig. 10.2 Execution time and speed-up factors with respect to frequency based on simulations conducted on gem5 [4] for three applications from the PARSEC benchmark suite [3] executing a single thread on an *out-of-order* Alpha 21264 core. The speed-up factors are normalized to the execution time of each application running at 0.2 GHz

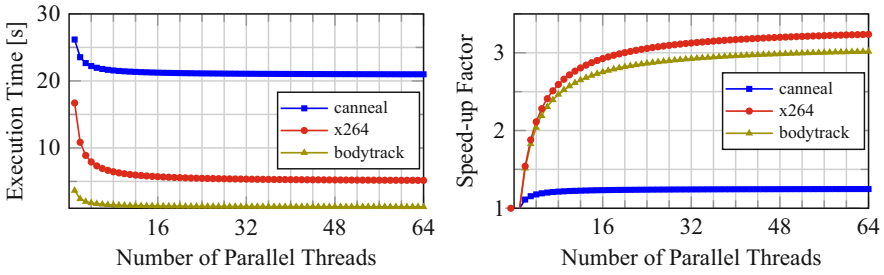

Fig. 10.3 Execution time and speed-up factors based on simulations conducted on gem5 [4] and Amdahl's law for three applications from the PARSEC benchmark suite [3] executing at 2 GHz on an *out-of-order* Alpha 21264 core. The speed-up factors are normalized to the execution time of each application running a single thread

10.2.2 Power and Energy Consumption

Power consumption is in nature an instantaneous metric which changes through time. Particularly, a core executing a certain thread of an application will consume different amounts of power at different time instants and application phases. For example, Fig. 10.4 illustrates the power consumption results of simulations conducted with Sniper [5] and McPAT [31], for a PARSEC bodytrack application executing four parallel threads on a quad-core Intel Nehalem cluster running at 2.2 GHz. The power consumption values observed on a specific core at a given point in time depend on several parameters, e.g., the underlying architecture of the core, the technology scaling generation, the mode of execution of the core (e.g., active, idle, or in a low-power mode), the selected voltage/frequency levels for execution, the temperature on the core (for leakage power thermal dependency), the application phase begin executed, etc.

Generally, as detailed in [17], the power consumption of a CMOS core can be modeled as formulated in Equation (10.1):

$$P = \alpha \cdot C_{\text{eff}}^{\text{app}} \cdot V_{\text{dd}}^2 \cdot f + V_{\text{dd}} \cdot I_{\text{leak}}\left(V_{\text{dd}}, T\right) + P_{\text{ind}} \tag{10.1}$$

where α represents the activity factor (or utilization) of the core, $C_{\text{eff}}^{\text{app}}$ represents the effective switching capacitance of a given application, V_{dd} represents the supply

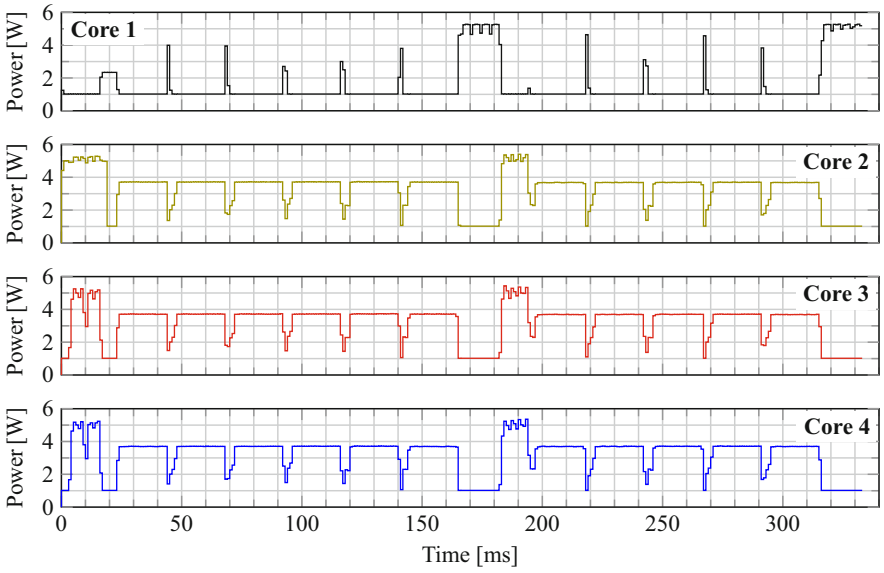

Fig. 10.4 Power consumption results of simulations conducted using Sniper [5] and McPAT [31], for a PARSEC bodytrack application with *simmedium* input, executing four parallel threads at 2.2 GHz on a quad-core Intel Nehalem cluster

voltage, f represents the execution frequency, I_{leak} represents the leakage current (which depends on the supply voltage and the core's temperature T), and P_{ind} represents the independent power consumption (attributed to keeping the core in execution mode, i.e., the voltage-/frequency-independent component of the active power consumption). Moreover, in Equation (10.1), $\alpha \cdot C_{\text{eff}}^{\text{app}} \cdot V_{\text{dd}}^2 \cdot f$ represents the dynamic power consumption, while $V_{\text{dd}} \cdot I_{\text{leak}}(V_{\text{dd}}, T)$ represents the leakage power consumption. Hence, if a core is clock gated, it still consumes leakage and indirect power. On the other hand, cores can also be set to some low-power mode (e.g., sleep or power-gated), each mode with an associated power consumption and different latencies for entering and leaving each low-power mode.

With respect to the voltage and frequency of the core, in order to stably support a specific frequency, the supply voltage of the core has to be adjusted above a minimum value. This minimum voltage value is frequency dependent, and higher frequencies require a higher minimum voltages. Furthermore, as shown by Pinckney et al. [49], the relation between the frequency and the corresponding minimum voltage can be modeled according to Equation (10.2):

$$f = k \cdot \frac{(V_{\text{dd}} - V_{\text{th}})^2}{V_{\text{dd}}} \tag{10.2}$$

where V_{th} is the threshold voltage and k is a fitting factor. Expressed in other words, the physical meaning of Equation (10.2) is that for a given supply voltage, there is a maximum stable frequency at which a core can be executed, and running at lower frequencies is stable but power/energy inefficient. Therefore, if the system runs at the corresponding power-/energy-efficient voltage and frequency pairs, we can derive a linear relationship between voltage and frequency and thus arrive at a *cubic* relation between the frequency and the dynamic power consumption. Figure 10.5 uses Equation (10.2) to model the minimum voltages necessary for stable execution on a 28 nm x86-64 microprocessor [14], and Fig. 10.6 shows how the power model from Equation (10.1) fits average power consumption results from McPAT [31] simulations for an x264 application from the PARSEC benchmark suite [3].

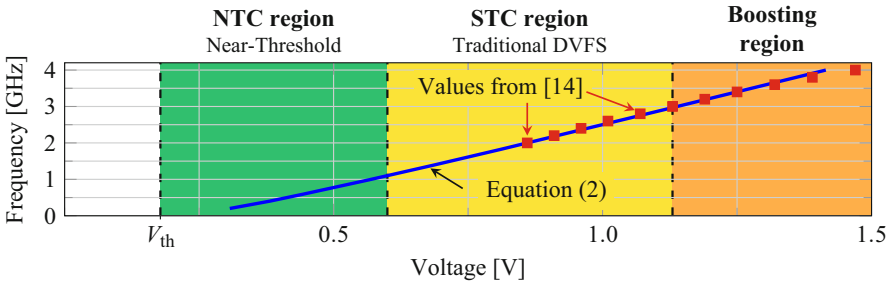

Fig. 10.5 Frequency and voltage relation modeled with Equation (10.2) for the *experimental results of a* 28 *nm x86-64 microprocessor* developed in [14]

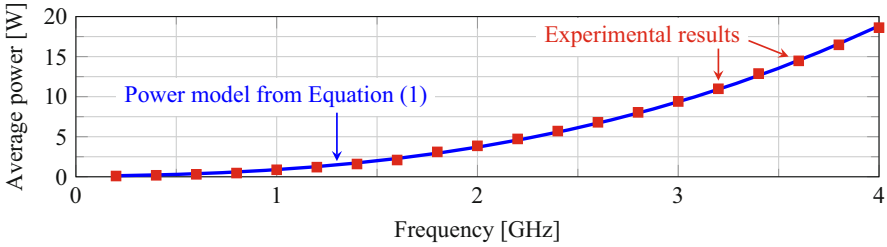

Fig. 10.6 Experimental results for a 22 nm *out-of-order* Alpha 21264 core, based on our simulations conducted on gem5 [4] and McPAT [31] for an x264 application from the PARSEC benchmark suite [3] running a single thread, and the derived power model from Equation (10.1)

a

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

b

Voltage	Frequency
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

c

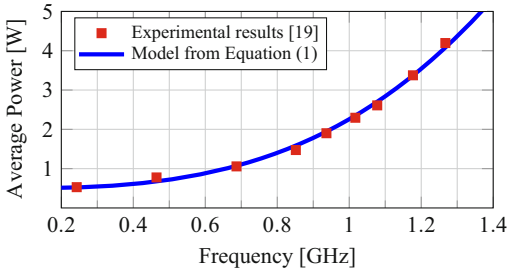

Fig. 10.7 Experimental results for the 48-core system developed in [19] and the power model from Equation (10.1). (**a**) Frequency vs. voltage [19] (**b**) Power vs. voltage [19] (**c**) Power model for a single core

Similarly, we can also use Equation (10.1) to model the experimental results from a research paper on Intel's Single Chip Cloud computer (SCC) [19], which developed a many-core system that integrates 48 cores. The work in [19], presents a relationship between the minimum voltages necessary for stable execution when running the cores at different frequencies, as well as average power consumption values for the entire chip when executing computational intensive applications running at the maximum allowed frequency for a given voltage, and these results are summarize in Fig. 10.7a, b. Therefore, we can fit the power model from Equation (10.1) based on the values of these two tables, such that the power consumption on individual cores can be modeled as illustrated in Fig. 10.7c.

Energy is the integration of power over time, and thus, when plotted, the energy consumed between two time points is equivalent to the area below the power curve between the two time points. Therefore, energy is associated with a time window, for example, an application instance. Figure 10.8 presents *average* power consumption and energy consumption examples for one instance of some applications from the

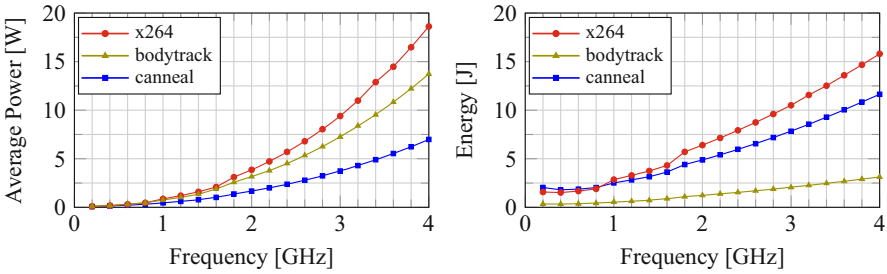

Fig. 10.8 Average power and energy values based on simulations in gem5 [4] and McPAT [31] for one instance of three applications from the PARSEC benchmark suite [3] executing a single thread on an *out-of-order* Alpha 21264 core

PARSEC benchmark suite [3] executing a single thread on an *out-of-order* Alpha 21264 core.

Previous work in the literature [27, 42] has shown that there exists a critical frequency for every application executing on a certain type of core which minimizes the energy consumption for execution. Namely, although executing at slow frequencies reduces the power consumption (due to the cubic relationship between dynamic power consumption and the frequency), it also prolongs the execution time of an application. Therefore, the critical frequency represents the frequency for which the energy savings achieved by reducing the power consumption (mainly savings in dynamic energy) are less significant than the corresponding increases in the energy consumption for prolonging the execution time (mainly due to leakage effects). In simple terms, this means that executing an application below its critical frequency for the corresponding type of core is not energy efficient, and it should hence be preferably avoided, even if it reduces the power consumption and meets the performance and timing constraints. The examples in Fig. 10.8 show the presence of such discussed critical frequency, where we can see that executing an application below 0.4 GHz (0.2 GHz in the figure) consumes more energy than executing it at 0.4 GHz.

Minimizing the overall energy consumption under timing (performance) constraints is a common optimization goal for real-time mobile systems, in which prolonging the battery lifetime is of major importance. Furthermore, on other battery-operated systems for which we can estimate the elapsed time between charging cycles (e.g., mobile phones), energy could also be used as a constraint, such that the system optimizes (maximizes) the overall performance under the battery's energy budget. Contrarily, it is very rare to optimize for power consumption, and thus power is mostly considered as a constraint, for example, to run the system under the given Thermal Design Power (TDP).

10.2.3 Temperature

Whenever some part of the chip is consuming power, it is also generating heat. Given that excessively high temperatures on the chip can cause permanent failures in transistors, maintaining the temperature throughout the chip below a certain threshold is of paramount importance. This issue is even more significant in modern chips due to voltage scaling limitations, which lead to increasing power densities across technology scaling generations and the so-called *dark silicon problem* [17, 37], that is, all parts of a chip cannot be simultaneously active at full speed. The use of a cooling solution (e.g., the combination of the thermal paste, the heat spreader, the heat sink, the cooling fan, etc.) and Dynamic Thermal Management (DTM) techniques is employed for such a purpose. DTM techniques are generally reactive (i.e., only become active after the critical temperature is reached or exceeded) and may power-down cores, gate their clocks, reduce their supply voltage and execution frequency, boost-up the fan speed, etc. Nevertheless, DTM techniques are generally aimed at avoiding the chip from overheating, not to optimize its performance. An abstract example of a DTM technique based on voltage and frequency scaling is presented in Fig. 10.9.

Although there exist some work which aims at reducing the peak temperature under performance constraints or at minimizing the thermal gradients in the chip, temperature is mostly considered as a constraint rather than a goal. Furthermore, thermal constraints tend to be the biggest limiting factor for performance optimization, especially in modern computing platforms in which power densities are ever increasing.

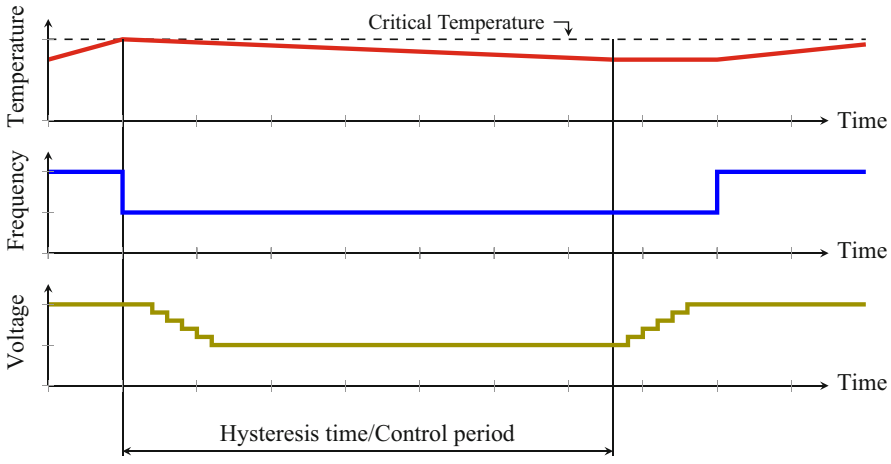

Fig. 10.9 Example of a DTM technique based on voltage and frequency scaling

The most widely adopted models used for thermal modeling in electronics are RC thermal networks, which are based on the well-known duality between thermal and electrical circuits [20]. In an RC thermal network, thermal conductances interconnect the thermal nodes among each other. Furthermore, there is a thermal capacitance associated with every thermal node that accounts for the transient effects in the temperatures, but there is no thermal capacitance associated with the ambient temperature as it is considered to be constant for long periods of time. The power consumption of cores and other elements corresponds to heat sources. In this way, the temperatures throughout the chip can be modeled as a function of the ambient temperature, the power consumptions inside the chip, and by considering the heat transfer among neighboring thermal nodes.

An example of a simplified RC thermal network for a chip with two cores is presented in Fig. 10.10. In the figure, T_1 and T_2 are voltages that represent the temperatures on core 1 and core 2, respectively. Voltages T_3 and T_4 represent the temperatures on the heat sink immediately above the cores. Current supplies p_1 and p_2 represent the power consumptions on each core. The thermal conductances b_{c}, $b_{\text{c-hs}}$, b_{hs}, and g_{amb} account for the heat transfer among the thermal nodes. Finally, the thermal capacitances of thermal node i are represented by capacitor a_i. By using Kirchoff's first law and linear algebra for the example in Fig. 10.10, we can derive a system of first-order differential equations as:

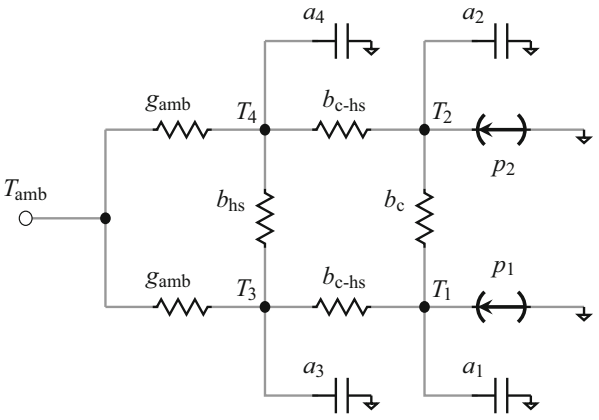

Fig. 10.10 Simple RC thermal network example for *two* cores (Figure from [46]), where we consider that cores are in direct contact with the heat sink and being the only connection between cores and the ambient temperature. A more detailed example would consider more layers between a core and the heat sink, for example, the ceramic packaging substrate, the thermal paste, and the heat spreader; and there would be more paths leading to the ambient temperature, for example, through the circuit board

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

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

$$\begin{bmatrix} b_{\text{c-hs}} + b_{\text{c}} & -b_{\text{c}} & -b_{\text{c-hs}} & 0 \\ -b_{\text{c}} & b_{\text{c-hs}} + b_{\text{c}} & 0 & -b_{\text{c-hs}} \\ -b_{\text{c-hs}} & 0 & b_{\text{c-hs}} + b_{\text{hs}} + g_{\text{amb}} & -b_{\text{hs}} \\ 0 & -b_{\text{c-hs}} & -b_{\text{hs}} & b_{\text{c-hs}} + b_{\text{hs}} + g_{\text{amb}} \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} + \begin{bmatrix} a_1 & 0 & 0 & 0 \\ 0 & a_2 & 0 & 0 \\ 0 & 0 & a_3 & 0 \\ 0 & 0 & 0 & a_4 \end{bmatrix} \begin{bmatrix} T_1' \\ T_2' \\ T_3' \\ T_4' \end{bmatrix} = \begin{bmatrix} p_1 \\ p_2 \\ 0 \\ 0 \end{bmatrix} + T_{\text{amb}} \begin{bmatrix} 0 \\ 0 \\ g_{\text{amb}} \\ g_{\text{amb}} \end{bmatrix}.$$

Therefore, RC thermal networks can serve as mathematical expressions to model the temperatures on the chip. In condensed matrix and vector form, the system of first-order differential equations of an RC thermal network is expressed as:

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

where for a system with N thermal nodes, T_{amb} denotes the ambient temperature, matrix $\mathbf{A} = \left[a_{i,j}\right]_{N\times N}$ holds the values of the thermal capacitances (generally a diagonal matrix, since thermal capacitances are modeled to ground), matrix $\mathbf{B} = \left[b_{i,j}\right]_{N\times N}$ contains the values of the thermal conductances between vertical and lateral neighboring nodes (in $\left[\frac{\text{Watt}}{\text{Kelvin}}\right]$), column vector $\mathbf{T} = [T_i]_{N\times 1}$ represents the temperature on each node, column vector $\mathbf{T}' = \left[T_i'\right]_{N\times 1}$ represents the first-order derivative of the temperature on each 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 conductance between every node and the ambient temperature. In practice, the RC thermal network model of a chip and cooling solution can be modeled through profiling or by using a modeling tool like HotSpot [20].

10.2.4 Optimization Knobs

In order to achieve the abovementioned optimization goals under the corresponding constraints, efficient resource management techniques are required. Such techniques, however, are based on some basic hardware and software methods (commonly present in standard chip designs) which are used and exploited as optimization knobs. Among such optimization knobs, the most commonly used are thread-level selection, thread-to-core mapping and run-time task migration, Dynamic Power Management (DPM), and Dynamic Voltage and Frequency Scaling (DVFS). Specifically:

- **Thread-level selection** refers to selecting an appropriate level of parallelism for every application, which depending on the TLP of each application can have a major impact in the resulting performance, as seen in Sect. 10.2.1.
- **Thread-to-core mapping** involves 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. In other words, the type of core to which a thread is mapped to is very important. Nevertheless, the selection of the physical location of the core is also a nontrivial issue, as this will have an impact on the performance (due to communication latencies among cores, potential link congestions, and the utilization of the shared resources) and also on the resulting temperature distribution (due to the heat transfer among cores, potentially creating or avoiding hotspots).
- **Run-time task migration** is simply the ability to migrate a task/thread from one core to another, at run-time. When migrating tasks at run-time, binary compatibility of tasks needs to be considered, given that, for example, different cores might have different ISAs, or a software task may be migrated to a reconfigurable fabric as a hardware implementation. Furthermore, it is also important to consider the non-negligible migration overheads, which could potentially result in larger performance penalties than benefits when applying too frequent migrations. For example, depending on the memory hierarchy, both instruction and data cache will experience many misses after a task is migrated from one core to another.
- **Dynamic Power Management (DPM)** refers to the dynamic power state selection of cores. For example, cores could be set to execution (active) mode, or they could be set to a low-power mode (e.g., clock gated, sleep, power gated, etc.). Every low-power mode has an associated power consumption and different latencies for entering and leaving each mode.
- **Dynamic Voltage and Frequency Scaling (DVFS)** refers to the ability to dynamically scale the voltage and/or frequency of a single core or a group of cores. Depending on the chip, voltage scaling and frequency scaling could be available at a per-core level, there could be only one global voltage and frequency, or it could be managed by groups of cores (i.e., clusters or voltage/frequency islands). For example, in Intel's Single Chip Cloud computer (SCC) [23], cores

are clustered into groups of eight cores that share the same voltage (dynamically set at run time), while the frequency of the cores can be selected every two cores, such that we can have up to four different frequencies inside each cluster of eight cores sharing a voltage.

10.3 Performance Optimization Under Power Constraints

As explained in Sect. 10.2, maximizing the overall system performance is generally the most commonly pursued optimization goal. In regard to the constraints, some resource management techniques consider power consumption, others consider temperature, and some consider both power and temperature. Furthermore, with respect to the power consumption, every system will have a physical power constraint which can, for example, be determined by the wire thickness or the supply voltage. Nevertheless, it is also very common to use power constraints as abstractions that allow system designers to indirectly deal with temperature. Namely, running the system under a given power constraint should presumably avoid violations of the thermal constraints. In line with such a concept, a power constraint aimed as a thermal abstraction is considered to be *safe* if satisfying it guarantees no thermal violations, and it is considered to be *efficient* if it results in temperatures that are not too far away from the critical temperature. Figure 10.11 shows an abstract example of a *safe* and *efficient* power budget, in which the maximum temperature throughout the chip remains just below the critical value when the system does not exceed the power budget.

The motivation for this approach is mainly to simplify the problem, given that proactive resource management techniques that directly deal with temperature are potentially more complex than those that only deal with power, mostly due to the heat transfer among cores and transient thermal effects.

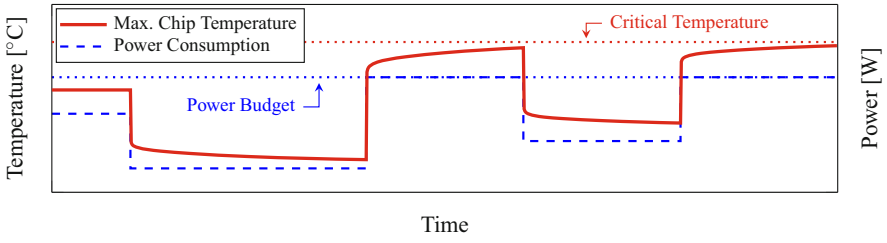

Fig. 10.11 Abstract example of a *safe* and *efficient* power budget

10.3.1 Traditional Per-Chip Power Constraints

The most common scheme in this direction (i.e., to have a power constraint as a thermal abstraction) is to use **TDP as a per-chip power constraint**, and there are several works in the literature aiming at performance optimization for such scenarios [30, 37, 52, 55].

Muthukaruppan et al. [37] propose a control-based framework that attempts to obtain the optimal trade-off between performance and power consumption, for homogeneous multi-core systems, while using TDP as a per-chip power constraint. DVFS and task migrations are used at different levels, specifically, at a task level, at a cluster level, and on the chip controllers. The controllers are coordinated such that they can throttle down the power consumption in case TDP is violated and to map tasks to cores in order to optimize the overall performance.

Also for overall performance optimization on homogeneous systems, Raghunathan et al. [52] try to exploit process variations between cores as a means for choosing the most suitable cores for every application. Their results show that, mostly due to the proportional increment of the process variations, the performance efficiency can potentially be increased along with the increase in the dark silicon area.

Sartori and Kumar [55] focus on maximizing many-core processor throughput for a given peak power constraint. It proposes three design-time techniques: mapping the power management problem to a knapsack problem, mapping it to a genetic search problem, and mapping it to a simple learning problem with confidence counters. These techniques prevent power from exceeding the given constraint and enable the placement of more cores on a die than what the power constraint would normally allow.

Kultursay et al. [30] build a 32-core TFET-CMOS heterogeneous multi-core processor and present a run-time scheme that improves the performance of applications running on these cores, while operating under a given power constraint. The run-time scheme combines heterogeneous thread-to-core mapping, dynamic work partitioning, and dynamic power partitioning.

> However, using a single and constant value as a power constraint, either at a per-chip or per-core level, for example, TDP, can easily result in thermal violations or significantly underutilized resources on multi-/many-core systems. This effect and a solution are discussed in Sect. 10.3.2.

10.3.2 Efficient Power Budgeting: Thermal Safe Power

For a system with 16 cores (*simple in-order* Alpha 21264 cores in 45 nm, simulated with McPAT [31]) and HotSpot's default cooling solution [20], Fig. 10.12 shows

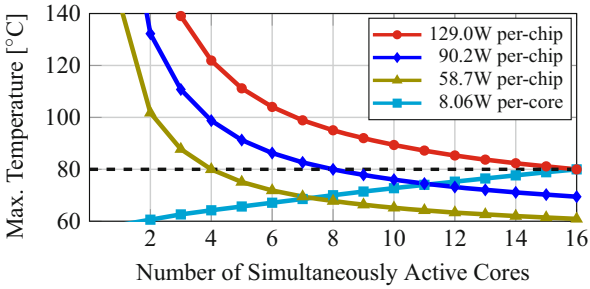

Fig. 10.12 Maximum steady-state temperature among all cores (DTM deactivated) as a function of the number of active cores, when using different *single* and *constant* per-chip and per-core power budgets

the maximum temperature among all cores (in the steady state, for concentrated mappings, and with DTM deactivated) as a function of the number of simultaneously active cores when using several traditional per-chip or per-core power budgets. Assuming that the critical temperature in this case is 80 °C, the figure shows that (for these specific concentrated mappings) there is only one point for each power budget in which the maximum temperature on the chip matches the critical value. For the other number of active cores, the temperature is either below or above the threshold, meaning that the power budget was not efficient or that it was not safe, respectively.

Therefore, the **Thermal Safe Power (TSP)** [47] power budget concept is introduced, proposing a *safe and efficient* alternative. The idea behind TSP is to have a per-core power constraint that depends on the number of active cores, rather than considering a single and constant value. Executing cores at power levels that satisfy TSP can result in a higher overall system performance when compared to traditional per-chip and per-core power budgeting solutions while maintaining the temperature throughout the chip below the critical value. Based on the RC thermal network of a given chip and its cooling solution, TSP can be computed at design time in order to obtain safe power constints for the worst-case mappings (namely, concentrated mappings promoting hotspots) as shown in the example in Fig. 10.13a, thus allowing the system to make thread-level selection decisions independent of the thread-to-core mapping. Furthermore, TSP can also be computed at run time for a (given) specific mapping of active cores, such that the system can further optimize the power budget for dispersed core mappings, for example, as shown in Fig. 10.13b.

Generally, as the number of active cores increases, the TSP power constraints decrease (as seen in Fig. 10.14), which in turn translates to executing cores at lower voltage and frequency levels. In this way, TSP derives a very simple relation between the number of active cores in a processor and their (application dependent) maximum allowed voltage and frequency levels for execution.

The major limitation with the techniques discussed in this section (both traditional power constraints and TSP) is that power is generally not easily measured in practical systems, mainly due to the lack of per-core power meters. In order to

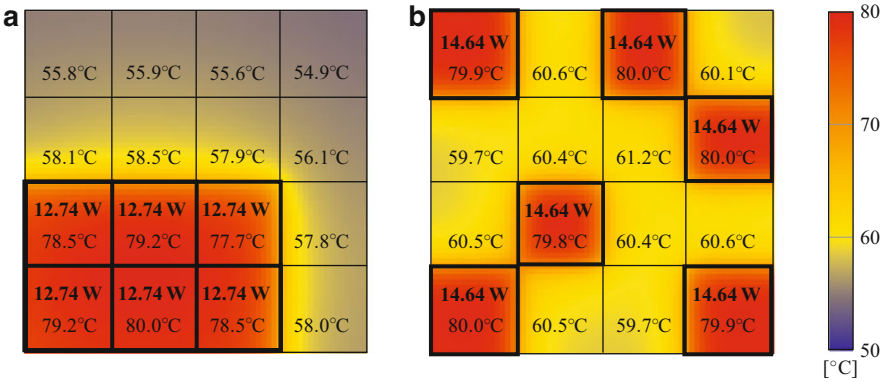

Fig. 10.13 Example of TSP for two different mappings for a maximum temperature of 80 °C (Figure from [46]). *Top numbers* are the power consumptions of each active core (boxed in *black*). *Bottom numbers* are the temperatures in the center of each core. Detailed temperatures are shown according to the *color bar*. (**a**) Concentrated mapping example with 6 active cores. (**b**) Distributed mapping example with 6 active cores

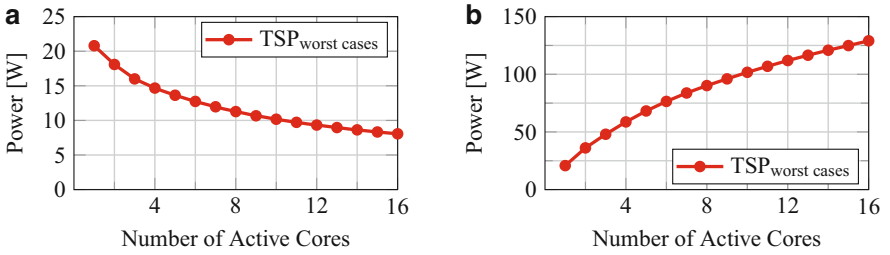

Fig. 10.14 Example of TSP for the worst-case core mappings. Per-chip values are estimated by multiplying per-core values with the number of active cores. (**a**) Per-core budget. (**b**) Estimated per-chip budget

address this issue, there are some works in the literature that attempt to estimate power consumption by measuring performance counters and core utilization [32]. Otherwise, extensive design-time application profiling with potential run-time refinement is required to estimate the power consumption of different applications running on different types of cores.

10.4 Performance Optimization Under Thermal Constraints

A different approach is to avoid using a power constraint as a thermal abstraction (as discussed in Sect. 10.3) and rather deal with temperature directly, either through thermal models or by using thermal sensors.

Using thermal models that rely on power estimations (as seen in Sect. 10.2.3) merely adds complexity to the problem (by considering the heat transfer among cores and the transient temperature effects), and it does so by maintaining the same issues with regard to how power consumption can be estimated in practice. Although this might look as a downside, the motivation for using these techniques instead of those presented in Sect. 10.3 is to avoid possible pessimistic or unsafe scenarios that can exist when using power constraints as thermal abstractions. Furthermore, these techniques can be proactive in nature, and we discuss them in more detail in Sect. 10.4.1.

Techniques that directly measure temperature by using thermal sensors can potentially be more accurate and easier to implement, as it is very common to find many thermal sensors in modern processors, to the point of possibly having one thermal sensor for every core in the chip. In this way, techniques that rely on thermal sensors can avoid the need for power consumption estimation tools. Nevertheless, the problem with these techniques is that it is very hard to do proactive thermal management without having thermal models. Therefore, they are generally reactive techniques which exploit the available thermal headroom, commonly also known as **boosting**, as discussed in Sect. 10.4.2.

10.4.1 Techniques Based on Thermal Modeling

Khdr et al. [28] propose a design-time dark silicon-aware resource management technique based on dynamic programming, called **DsRem**, which attempts to distribute the processors resources among different applications. Based on extensive application profiling, DsRem determines the number of active/dark cores and their voltage/frequency levels by considering the TLP and ILP characteristics of the applications, in order to maximize the overall system performance. Specifically, DsRem will attempt to map applications with high TLP by using a large number of cores executing at low voltage/frequency levels while mapping applications with high ILP to a small number of cores executing at high voltage/frequency levels. Applications that exhibit both high TLP and high ILP will potentially be mapped to a large number of cores executing at high voltage/frequency levels whenever possible.

Another work in the literature proposes a variability-aware dark silicon manager, called **DaSiM** [58]. In order to optimize the system performance under a thermal constraint, the idea behind DaSiM is to exploit the so-called dark silicon patterning in tandem with the core-to-core leakage power variations. Different dark silicon patterns denote different spatiotemporal decisions for the power state of the cores, namely, which cores to activate and which to put in low-power mode. These patterns directly influence the thermal profile on the chip due to improved heat dissipation, enabling to activate more cores to facilitate high-TLP applications and/or boosting certain cores to facilitate high-ILP applications. In order to enable run-time optimizations, DaSiM employs a lightweight run-time temperature prediction mechanism that estimates the chip's thermal profile for a given candidate solution.

Hanumaiah et al. [16] propose a thermal management technique based on RC thermal networks which attempts to minimize the latest completion time among the applications. This technique first derives an optimal solution by using convex optimization, which has a high time complexity and is therefore only suited for design-time decisions. Nevertheless, the structure of certain matrices in the convex optimization formulation can be exploited in order to have an approximate solution which is 20 times faster than the convex optimization approach. The implementation of such a technique for run-time adaptations is however debatable, as the experiments in [16] show that it may require more than 15 ms to compute the voltage and frequency levels of *each* core, which is generally not fast enough for run-time usage in multi-/many-core systems.

Pagani et al. [48] presents **seBoost**, a run-time boosting technique (see Sect. 10.4.2) based on analytical transient temperature estimation on RC thermal networks [45]. This technique attempts to meet run-time performance requirement surges, by executing the cores mapped with applications that require boosting at the specified voltages and frequencies while throttling down the cores mapped with application of lower priority. The performance losses of the low-priority applications are minimized by choosing the throttling down levels such that the critical temperature is reached precisely when the boosting interval is expected to expire. Furthermore, in order to select the throttle down levels of the cores mapped with the low-priority applications, seBoost performs a Design Space Exploration (DSE) but limiting the number of evaluated combinations by using a binary search like approach proportional to the nominal voltage/frequency operation levels on every core. A limitation of seBoost is that it assumes that the thread to core is given as an input, that is, it requires mapping decisions to be known a priori. Therefore, similar to the boosting techniques later explained in Sect. 10.4.2, seBoost needs to rely on another resource management technique to do the thread-level selection of applications and the mapping of threads to cores, such that it can then handle the boosting decisions and exploit the available thermal headroom.

10.4.2 Boosting Techniques

Boosting techniques have been widely adopted by chip manufacturers in commercial multi-/many-core systems, mostly because they provide the ability to exploit the existing thermal headroom in order to optimize the performance of a group of cores at run-time. Basically, by using DVFS, boosting techniques allow the system to execute some cores at high voltage and frequency levels during short intervals of time, even if this means exceeding standard operating power budgets (e.g., TDP), such that the system can optimize its performance under a thermal constraint. Given that executing at high voltage and frequency levels increases the power consumption in the chip, boosting techniques will incur in increments to the chip's temperature through time. Because of this increase in the temperature, once any part of the chip reaches a critical (predefined) temperature, the system should return to nominal operation (requiring some cool-down time before another

boosting interval is allowed) or use a closed loop control-based technique in order to oscillate around the critical temperature (allowing the boosting interval to be prolonged indefinitely) [48].

Boosting techniques generally do not aim at selecting the number of threads in which to parallelize applications, to make thread-to-core mapping decisions, or to migrate tasks between cores. Therefore, they are mostly well suited to exploit any available thermal headroom left by some other resource management technique (e.g., a thread-level selection and thread-to-core mapping technique based on a pessimistic per-chip power budget), in order to increase the system performance at run-time.

Intel's Turbo Boost [6, 8, 21, 22, 53] allows for a group of cores to execute at high voltage and frequency levels whenever there exists some headroom within power, current, and temperature constraints. In other words, when the temperature, power, and current are below some given values, the cores boost their voltage and frequency in single steps (within a control period) until it reaches a predetermined upper limit according to the number of active cores. Similarly, when either the temperature, power, or current exceeds the constraints, the cores reduce their voltage and frequency in single steps (also within a control period) until the corresponding constraints are satisfied or until the nominal voltage and frequency levels are reached. Although boosting to very high voltage and frequencies has an undesired effect on the power consumption (due to the cubic relationship between frequency and dynamic power consumption), Turbo Boost exploits the thermal capacitances of the thermal model knowing that, although there will be a temperature increase, this increase will require some time to reach the critical temperature rather than reach it immediately after the change in power. Figure 10.15 shows an example of Turbo

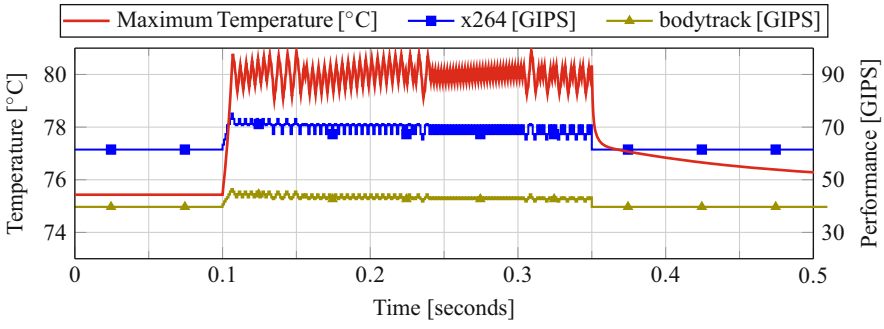

Fig. 10.15 Turbo Boost [6] example. The *red line* shows the maximum temperature among all cores (*left axis*). The performance of the applications is measured in Giga-Instruction Per Second (GIPS)

Boost's behavior based on simulations conducted in gem5 [4], McPAT [31], and HotSpot [20] for executing two applications from the PARSEC benchmark suite [3] in a multi-core processor with 16 *out-of-order* Alpha 21264 cores (each application running *eight* parallel dependent threads, one thread per core), under a temperature constraint of 80 °C.

Computational sprinting [51] is another boosting technique which proposes optimizing performance at run time via parallelism, by activating cores that are normally off (i.e., power gated) during short bursts of time (typically shorter than 1 s). Due to the cubic relationship between frequency and dynamic power consumption, computational sprinting intentionally discourages boosting through DVFS. Contrarily, it is motivated by the (ideally) linear relationship between performance and power expected when the system activates several cores at the same voltage and frequency levels. However, although this is a very valid point, the latency for waking up cores from low-power modes and the correspondent thread migrations can potentially result in significant overheads, especially when taking into consideration the short duration of the sprinting periods. Because of this, Turbo Boost will generally result in a higher overall system performance than computational sprinting.

10.5 Energy Optimization Under Performance Constraints

Energy-efficient scheduling and power management to minimize the overall energy consumption for **homogeneous** multi-core systems has been widely explored for real-time embedded systems with **per-core DVFS**, for example, [2, 9, 10, 36, 65]. Chen and Thiele [10] present a theoretical analysis of the Largest Task First (LTF) strategy, proving that, in terms of energy consumption, using LTF for task mapping results in solutions with approximation factors (particularly, the analytical worst-case ratio between the optimal solutions and the algorithms of interest) that depend on the hardware platforms. Moreover, [10, 65] propose polynomial-time algorithms that derive task mappings which attempt to execute cores at their critical frequency. For the special case in which there are uniform steps between the available core frequencies and also negligible leakage power consumption (a very restricting assumption), the work in [36] presents an algorithm that requires polynomial time for computing the optimal voltage and frequency assignments. Nevertheless, although having per-core DVFS can be very energy-efficient, Herbert and Marculescu [18] conducts extensive VLSI circuit simulations suggesting that it suffers from complicated design problems, making it costly for implementation. Therefore, assuming global DVFS, or DVFS at a cluster level (i.e., groups of cores or voltage/frequency islands), is much more realistic for practical systems, as seen in [23, 54].

For **homogeneous** systems with one global supply voltage and frequency, also referred to as **global DVFS**, Yang et al. [66] provide energy optimization solutions for systems with negligible leakage power consumption and frame-based real-time tasks (all tasks have the same arrival time and period). These are both very restricting

assumptions, and, therefore, the work in [12, 57] relaxes them in order to consider periodic real-time tasks (tasks have different arrival times and periodicity) with non-negligible leakage power consumption and non-negligible overhead for turning cores to low-power modes. Specifically, Seo et al. [57] proposes to dynamically balance the task loads of multiple cores and efficiently select the number of active cores, such that the power consumption for execution is minimized and the leakage power consumption for low workloads is reduced. Devadas and Aydin [12] first decide the number of active cores, and the voltage and frequencies of such cores are decided in a second phase. However, [12] does not provide theoretical analysis for the approximation factor of their proposed approach in terms of energy optimization. Furthermore, the basic ideas of [12] are used in [41] to theoretically derive the approximation factor, in terms of energy minimization, of the so-called Single Frequency Approximation (SFA) scheme. SFA is a simple strategy for selecting the DVFS levels on individual clusters. Particularly, after the tasks are assigned to clusters and cores, SFA uses a single voltage and frequency for all the cores in the cluster, specifically, the lowest DVFS level that satisfies the timing constraints of all tasks mapped to the cluster. Given that different tasks are assigned to different cores, not all cores in a cluster will have the same frequency requirements to meet the timing constraints. Therefore, the DVFS level of the cluster is defined by the core with the highest frequency demand in the cluster. Figure 10.16 presents an example of a cluster with four cores using SFA. The analysis of SFA is extended in [42] and [43] in order to consider the task partitioning phase.

For **homogeneous** systems with multiple **clusters of cores sharing their voltage and frequency**, there exist several heuristic algorithms [15, 29, 39, 44, 64], and

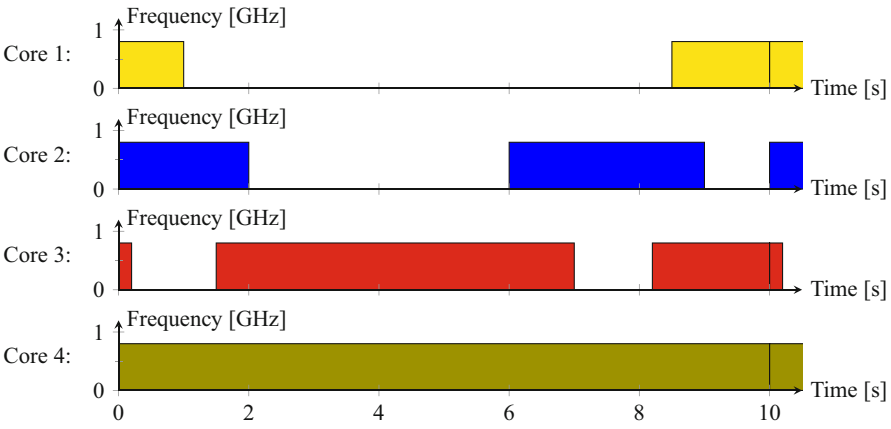

Fig. 10.16 SFA example for a cluster with four cores. The hyper-period of all tasks (i.e., the least common multiple among all periods of all tasks) is 10 s. To meet all deadlines, the frequency demand of the cores are 0.2, 0.4, 0.6, and 0.8 GHz. Hence, the single frequency is set to 0.8 GHz. In order to save energy, cores go individually to sleep when there is no workload on their ready queues

also an optimal dynamic programming solution [44], among which [13, 39, 44, 64] use SFA to choose the voltage and frequency of individual clusters. Particularly, Kong et al. [29] present a heuristic which first chooses the number of active clusters and then partitions the tasks by using the LTF task partitioning strategy. The task model used in [29] is later extended in [15] in order to consider shared resources and a synchronized version of LTF that considers that only one task can access specific resources at any given time instant. An Extremal Optimization meta-Heuristic (EOH) that considers a task graph and the communication costs among tasks is presented in [39], with the limitation that only one task can be assigned to each core. A heuristic based on genetic algorithms is presented in [64], where the energy consumption is gradually optimized in an iterative process through the selection, crossover, and mutation operations.

The work in [44] presents an optimal dynamic programming algorithm for given task sets (i.e., the tasks are already partitioned into cores, but the specific cores are not yet selected), called DYVIA, and suggests to use LTF for the task partitioning phase. Specifically, the authors of [44] first prove that when the average power consumption of different tasks executing at a certain DVFS level are equal or very similar and when the highest cycle utilization task sets in every cluster are given (i.e., when the DVFS levels of operation for every cluster are known), then the optimal solution will assign the highest cycle utilization task sets to the clusters running at the lowest possible DVFS levels, while still guaranteeing that the deadlines of all tasks are satisfied, as illustrated in Fig. 10.17. Furthermore, based on such a property, the DYVIA algorithm is able to reduce the number of combinations evaluated during its internal Design Space Exploration (DSE). For example, for a system with three clusters and three cores per cluster, if when finding the task sets to be assigned to cluster 3 we assume that the highest cycle utilization task set (i.e., $\mathbf{T}_{12}$) is always assigned to cluster 3, there are in total $\binom{8}{2} = 28$ possible combinations for selecting the other two task sets to be assigned to cluster 3. However, as shown in the example

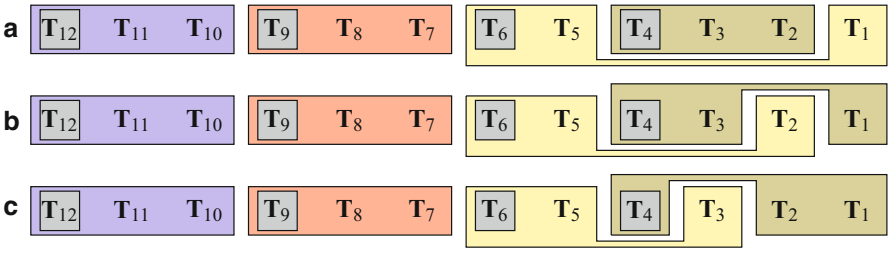

Fig. 10.17 Examples of possible task set assignments, for a chip with four clusters and three cores per cluster, where the task sets are increasingly ordered according to their cycle utilization. The figure shows the three possible assignments when the highest cycle utilization task sets in the clusters are $\mathbf{T}_4$, $\mathbf{T}_6$, $\mathbf{T}_9$, and $\mathbf{T}_{12}$ (boxed in *gray*), for which [44] proves that combination (**a**) minimizes the energy consumption

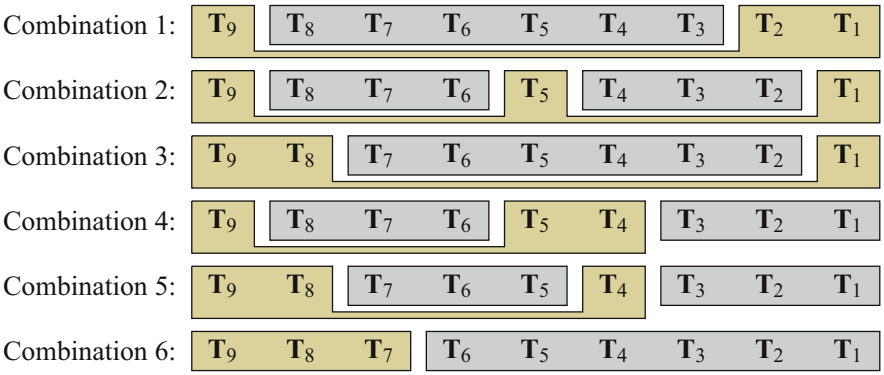

Fig. 10.18 Example of the potentially optimal combinations evaluated by the DYVIA algorithm [44], for a system with three clusters and three cores per cluster, where the task sets are increasingly ordered according to their cycle utilization. Each combination corresponds to the possible task sets to assign in cluster 3, for which DYVIA evaluates the resulting energy consumption, returning the minimum energy among all evaluated cases. In the figure, the task sets assigned to cluster 3 in a combination are boxed in *green*, and the resulting subproblems are colored in *gray*

in Fig. 10.18, for such case, DYVIA is able to reduce the design space, such that it only needs to evaluate *six potentially optimal combinations* in order to find the optimal assignment for cluster 3. DYVIA then finds the optimal assignment for cluster 2 by solving the associated subproblems.

There are several works focusing on **heterogeneous** systems with **per-core DVFS**. For example, Yang et al. [67] present a dynamic programming approach that uses trimming by rounding, in which the rounding factor can trade quality (in terms of energy optimization) of the derived solution with the total execution time of the algorithm.

There is also some work focusing on energy optimization for the more general model of **heterogeneous** multi-core systems with **clusters of cores sharing their voltage and frequency**, for example, [13, 38]. Muthukaruppan et al. [38] present a distributed framework based on price theory which attempts to minimize the overall power consumption, not the overall energy. Therefore, with the existence of critical frequencies, such a framework may fail to minimize the energy consumption when executing at very low frequencies (even when all performance constraints are satisfied). Moreover, this approach is not well suited for real-time systems, as it does not guaranty that all real-time tasks meet their deadlines, and only a *best effort* can be accomplished. Elewi et al. [13] propose a task partitioning and mapping scheme called Equally-Worst-Fit-Decreasing (EWFD), which attempts to balance the total utilization in every cluster. However, EWFD is an overly simplistic heuristic that assumes that executing different tasks on a given core type and frequency consumes equivalent power, which is not true in real systems as already observed in Fig. 10.8.

10.6 Hybrid Resource Management Techniques

Sharifi et al. [59] propose a joint temperature and energy management hybrid technique for heterogeneous Multi-Processor Systems-on-Chips (MPSoCs). In case that the system becomes underutilized, the technique focuses on energy minimization while satisfying the performance demands and thermal constraints. When the performance demands of the applications become higher, satisfying the thermal constraints has higher priority than minimizing energy, and thus the proposed technique applies a thermal balancing policy. This work includes a design-time application profiling phase that characterizes possible incoming tasks. It also derives different DVFS levels that balance the temperature throughout the chip for several performance demands. Then, at run-time, the technique integrates DVFS (including the design-time analysis) with a thread-to-core assignment strategy that is performance and temperature aware. When the performance demands can be satisfied only in some cores, the technique chooses which cores to power gate in order to minimize energy.

Ykman-Couvreur et al. [68] present a hybrid resource management technique for heterogeneous systems that aims at maximizing the overall Quality of Service (QoS) under changing platform conditions. In the design-time phase, by using an automated design-space exploration tool, the technique derives a set of Pareto-optimal application configurations under given QoS requirements and optimization goals. Then, the run-time resource management dynamically switches between the predefined configurations evaluated at design-time.

Singh et al. [60] present a hybrid management technique for mapping throughput-constrained applications on generic MPSoCs. The technique first performs design-time analysis of the different applications in order to derive multiple *resources/cores vs. throughput* trade-off points, therefore performing all the compute intensive analysis and leaving a minimum pending computation for the run-time phase. The run-time mapping strategy then selects the best point according to the desired throughput and available resources/cores.

Schor et al. [56] present a scenario-based technique for mapping a set of applications to a heterogeneous many-core system. The applications are specified as Kahn process networks. A finite state machine is used to coordinate the execution of the applications, where each state represents a scenario. During design-time analysis, the technique first precomputes a set of optimal mappings. Then, at run-time, hierarchically organized controllers monitor behavioral events and apply the precomputed mappings to start, stop, resume, and pause applications according to the finite state machine. In order to handle architectural failures, the technique allocates spare cores at design-time, such that the run-time controllers can move all applications assigned to a faulty physical core to a spare core. Given that this does not require additional design-time analysis, the proposed technique has a high responsiveness to failures.

Acknowledgments This work was partly supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Centre *Invasive Computing* [SFB/TR 89] http://invasic.de.

References

1. Al Faruque MA, Krist R, Henkel J (2008) ADAM: run-time agent-based distributed application mapping for on-chipcommunication. In: Proceedings of the 45th IEEE/ACM design automation conference (DAC), pp 760–765. doi:10.1145/1391469.1391664
2. Aydin H, Yang Q (2003) Energy-aware partitioning for multiprocessor real-time systems. In: Proceedings of 17th international parallel and distributed processing symposium (IPDPS), pp 113–121
3. Bienia C, Kumar S, Singh JP, Li K (2008) The PARSEC benchmark suite: characterization and architectural implications. In: Proceedings of the 17th international conference on parallel architectures and compilation techniques (PACT), pp 72–81
4. Binkert N, Beckmann B, Black G, Reinhardt SK, Saidi A, Basu A, Hestness J, Hower DR, Krishna T, Sardashti S, Sen R, Sewell K, Shoaib M, Vaish N, Hill MD, Wood DA (2011) The gem5 simulator. ACM SIGARCH Comput Archit News 39(2):1–7
5. Carlson TE, Heirman W, Eyerman S, Hur I, Eeckhout L (2014) An evaluation of high-level mechanistic core models. ACM Trans Archit Code Optim (TACO) 11(3):28:1–28:25. doi:10.1145/2629677
6. Casazza J (2009) First the tick, now the tock: intel microarchitecture (Nehalem). White paper, Intel Corporation
7. Ceng J, Castrillon J, Sheng W, Scharwächter H, Leupers R, Ascheid G, Meyr H, Isshiki T, Kunieda H (2008) MAPS: an integrated framework for MPSoC application parallelization. In: Proceedings of the 45th IEEE/ACM design automation conference (DAC), pp 754–759. doi:10.1145/1391469.1391663
8. Charles J, Jassi P, Ananth NS, Sadat A, Fedorova A (2009) Evaluation of the Intel core i7 turbo boost feature. In: IISWC, pp 188–197
9. Chen JJ, Hsu HR, Kuo TW (2006) 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), pp 408–417
10. Chen JJ, Thiele L (2010) Energy-efficient scheduling on homogeneous multiprocessor platforms. In: Proceedings of the ACM symposium on applied computing (SAC), pp 542–549
11. Choi J, Oh H, Kim S, Ha S (2012) Executing synchronous dataflow graphs on a SPM-based multicore architecture. In: Proceedings of the 49th IEEE/ACM design automation conference (DAC), pp 664–671. doi:10.1145/2228360.2228480
12. Devadas V, Aydin H (2010) Coordinated power management of periodic real-time tasks on chip multiprocessors. In: Proceedings of the international conference on green computing (GREENCOMP), pp 61–72
13. Elewi A, Shalan M, Awadalla M, Saad EM (2014) Energy-efficient task allocation techniques for asymmetric multiprocessor embedded systems. ACM Trans Embed Comput Syst (TECS) 13(2s):71:1–71:27
14. Grenat A, Pant S, Rachala R, Naffziger S (2014) 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), pp 106–107
15. Han JJ, Wu X, Zhu D, Jin H, Yang L, Gaudiot JL (2012) Synchronization-aware energy management for vfi-based multicore real-time systems. IEEE Trans Comput (TC) 61(12):1682–1696

16. Hanumaiah V, Vrudhula S, Chatha KS (2011) Performance optimal online DVFS and task migration techniques for thermally constrained multi-core processors. Trans Comput Aided Des Integr Circuits Syst (TCAD) 30(11):1677–1690
17. Henkel J, Khdr H, Pagani S, Shafique M (2015) New trends in dark silicon. In: Proceedings of the 52nd ACM/EDAC/IEEE design automation conference (DAC), pp 119:1–119:6. doi:10.1145/2744769.2747938
18. Herbert S, Marculescu D (2007) Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. In: Proceedings of the international symposium on low power electronics and design (ISLPED), pp 38–43
19. Howard J, Dighe S, Vangal S, Ruhl G, Borkar N, Jain S, Erraguntla V, Konow M, Riepen M, Gries M, Droege G, Lund-Larsen T, Steibl S, Borkar S, De V, Van Der Wijngaart R (2011) A 48-core IA-32 processor in 45 nm CMOS using on-die message-passing and DVFS for performance and power scaling. IEEE J Solid State Circuits 46(1):173–183. doi:10.1109/JSSC.2010.2079450
20. Huang W, Ghosh S, Velusamy S, Sankaranarayanan K, Skadron K, Stan MR (2006) HotSpot: a compact thermal modeling methodology for early-stage VLSI design. IEEE Trans VLSI Syst 14(5):501–513. doi:10.1109/TVLSI.2006.876103
21. Intel Corporation (2007) Dual-core intel xeon processor 5100 series datasheet, revision 003
22. Intel Corporation (2008) Intel turbo boost technology in Intel CoreTM microarchitecture (Nehalem) based processors. White paper
23. Intel Corporation (2010) SCC external architecture specification (EAS), revision 0.98
24. International technology roadmap for semiconductors (ITRS), 2011 edition. www.itrs.net
25. Jahn J, Pagani S, Kobbe S, Chen JJ, Henkel J (2013) Optimizations for configuring and mapping software pipelines in manycore. In: Proceedings of the 50th IEEE/ACM design automation conference (DAC), pp 130:1–130:8. doi:10.1145/2463209.2488894
26. Javaid H, Parameswaran S (2009) A design flow for application specific heterogeneous pipelined multiprocessor systems. In: Proceedings of the 46th IEEE/ACM design automation conference (DAC), pp 250–253. doi:10.1145/1629911.1629979
27. Jejurikar R, Pereira C, Gupta R (2004) Leakage aware dynamic voltage scaling for real-time embedded systems. In: Proceedings of the 41st design automation conference (DAC), pp 275–280
28. Khdr H, Pagani S, Shafique M, Henkel J (2015) 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), pp 179:1–179:6. doi:10.1145/2744769.2744916
29. Kong F, Yi W, Deng Q (2011) Energy-efficient scheduling of real-time tasks on cluster-based multicores. In: Proceedings of the 14th design, automation and test in Europe (DATE), pp 1–6
30. Kultursay E, Swaminathan K, Saripalli V, Narayanan V, Kandemir MT, Datta S (2012) Performance enhancement under power constraints using heterogeneous CMOS-TFET multicores. In: Proceedings of the 8th international conference on hardware/software codesign and system synthesis (CODES+ISSS), pp 245–254
31. Li S, Ahn JH, Strong R, Brockman J, Tullsen D, Jouppi N (2009) McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In: Proceedings of the 42nd annual IEEE/ACM international symposium on microarchitecture (MICRO), pp 469–480
32. Li Y, Henkel J (1998) A framework for estimation and minimizing energy dissipation of embedded hw/sw systems. In: Proceedings of the 35th ACM/IEEE design automation conference (DAC), pp 188–193. doi:10.1145/277044.277097
33. Lin LY, Wang CY, Huang PJ, Chou CC, Jou JY (2005) Communication-driven task binding for multiprocessor with latency insensitive network-on-chip. In: The 15th Asia and South Pacific design automation conference (ASP-DAC), pp 39–44. doi:10.1145/1120725.1120739
34. Mallik A, Marwedel P, Soudris D, Stuijk S (2010) MNEMEE: a framework for memory management and optimization of static and dynamic data in MPSoCs. In: Proceedings of the international conference on compilers, architectures and synthesis for embedded systems (CASES), pp 257–258. doi:10.1145/1878921.1878959

35. Martin G (2006) Overview of the MPSoC design challenge. In: Proceedings of the 43rd IEEE/ACM design automation conference (DAC), pp 274–279. doi:10.1109/DAC.2006.229245
36. Moreno G, de Niz D (2012) 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), pp 21–30
37. Muthukaruppan T, Pricopi M, Venkataramani V, Mitra T, Vishin S (2013) Hierarchical power management for asymmetric multi-core in dark silicon era. In: DAC, pp 174:1–174:9
38. Muthukaruppan TS, Pathania A, Mitra T (2014) 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), pp 161–176
39. Nikitin N, Cortadella J (2012) Static task mapping for tiled chip multiprocessors with multiple voltage islands. In: Proceedings of the 25th international conference on architecture of computing systems (ARCS), pp 50–62
40. Orsila H, Kangas T, Salminen E, Hämäläinen TD, Hännikäinen M (2007) Automated memory-aware application distribution for multi-processor system-on-chips. J Syst Archit 53(11):795–815. doi:10.1016/j.sysarc.2007.01.013
41. Pagani S, Chen JJ (2013) 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), pp 82–91. doi:10.1109/RTCSA.2013.6732206
42. Pagani S, Chen JJ (2013) Energy efficient task partitioning based on the single frequency approximation scheme. In: Proceedings of the 34th IEEE real-time systems symposium (RTSS), pp 308–318. doi:10.1109/RTSS.2013.38
43. Pagani S, Chen JJ, Henkel J (2015) Energy and peak power efficiency analysis for the single voltage approximation (SVA) scheme. IEEE Trans Comput Aided Des Integr Circuits Syst (TCAD) 34(9):1415–1428. doi:10.1109/TCAD.2015.2406862
44. Pagani S, Chen JJ, Li M (2015) Energy efficiency on multi-core architectures with multiple voltage islands. IEEE Trans Parallel Distrib Syst (TPDS) 26(6):1608–1621. doi:10.1109/TPDS.2014.2323260
45. Pagani S, Chen JJ, Shafique M, Henkel J (2015) MatEx: efficient transient and peak temperature computation for compact thermal models. In: Proceedings of the 18th design, automation and test in Europe (DATE), pp 1515–1520
46. Pagani S, Khdr H, Chen JJ, Shafique M, Li M, Henkel J (2016) Thermal safe power: efficient thermal-aware power budgeting for manycore systems in dark silicon. In: The dark side of silicon. Springer
47. Pagani S, Khdr H, Munawar W, Chen JJ, Shafique M, Li M, Henkel J (2014) TSP: thermal safe power – efficient power budgeting for many-core systems in dark silicon. In: The international conference on hardware/software codesign and system synthesis (CODES+ISSS), pp 10:1–10:10. doi:10.1145/2656075.2656103
48. Pagani S, Shafique M, Khdr H, Chen JJ, Henkel J (2015) seBoost: selective boosting for heterogeneous manycores. In: Proceedings of the 10th IEEE/ACM international conference on hardware/software codesign and system synthesis (CODES+ISSS), pp 104–113
49. Pinckney N, Sewell K, Dreslinski RG, Fick D, Mudge T, Sylvester D, Blaauw D (2012) Assessing the performance limits of parallelized near-threshold computing. In: 49th design automation conference (DAC), pp 1147–1152
50. Quan W, Pimentel AD (2015) A hybrid task mapping algorithm for heterogeneous MPSoCs. ACM Trans Embed Comput Syst (TECS) 14(1):14:1–14:25. doi:10.1145/2680542
51. Raghavan A, Luo Y, Chandawalla A, Papaefthymiou M, Pipe KP, Wenisch TF, Martin MMK (2012) Computational sprinting. In: Proceedings of the IEEE 18th international symposium on high-performance computer architecture (HPCA), pp 1–12
52. Raghunathan B, Turakhia Y, Garg S, Marculescu D (2013) Cherry-picking: exploiting process variations in dark-silicon homogeneous chip multi-processors. In: DATE, pp 39–44
53. Rotem E, Naveh A, Rajwan D, Ananthakrishnan A, Weissmann E (2012) Power-management architecture of the Intel microarchitecture code-named sandy bridge. IEEE Micro 32(2):20–27

54. Samsung Electronics Co., Ltd.: Exynos 5 Octa (5422). www.samsung.com/exynos
55. Sartori J, Kumar R (2009) Three scalable approaches to improving many-core throughput for a given peak power budget. In: International conference on high performance computing (HiPC), pp 89–98
56. Schor L, Bacivarov I, Rai D, Yang H, Kang SH, Thiele L (2012) Scenario-based design flow for mapping streaming applications onto on-chip many-core systems. In: Proceedings of the 15th international conference on compilers, architectures and synthesis for embedded systems (CASES), pp 71–80. doi:10.1145/2380403.2380422
57. Seo E, Jeong J, Park SY, Lee J (2008) Energy efficient scheduling of real-time tasks on multicore processors. IEEE Trans Parallel Distrib Syst (TPDS) 19(11):1540–1552. doi:10.1109/TPDS.2008.104
58. Shafique M, Gnad D, Garg S, Henkel J (2015) Variability-aware dark silicon management in on-chip many-core systems. In: Proceedings of the 18th design, automation and test in Europe (DATE), pp 387–392
59. Sharifi S, Coskun AK, Rosing TS (2010) Hybrid dynamic energy and thermal management in heterogeneous embedded multiprocessor SoCs. In: Proceedings of the Asia and South Pacific design automation conference (ASP-DAC), pp 873–878
60. Singh AK, Kumar A, Srikanthan T (2011) A hybrid strategy for mapping multiple throughput-constrained applications on MPSoCs. In: Proceedings of the 14th international conference on compilers, architectures and synthesis for embedded systems (CASES), pp 175–184. doi:10.1145/2038698.2038726
61. Smit L, Smit G, Hurink J, Broersma H, Paulusma D, Wolkotte P (2004) Run-time mapping of applications to a heterogeneous reconfigurable tiled system on chip architecture. In: Proceedings of the IEEE international conference on field-programmable technology (FPT), pp 421–424. doi:10.1109/FPT.2004.1393315
62. Tan C, Muthukaruppan T, Mitra T, Ju L (2015) Approximation-aware scheduling on heterogeneous multi-core architectures. In: The 20th Asia and South Pacific design automation conference (ASP-DAC), pp 618–623
63. Weichslgartner A, Gangadharan D, Wildermann S, GlaßM, Teich J (2014) DAARM: design-time application analysis and run-time mapping for predictable execution in many-core systems. In: Proceedings of the international conference on hardware/software codesign and system synthesis (CODES+ISSS), pp 34:1–34:10. doi:10.1145/2656075.2656083
64. Wu X, Zeng Y, Han JJ (2013) 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), pp 123–128
65. Xu R, Zhu D, Rusu C, Melhem R, Mossé D (2005) Energy-efficient policies for embedded clusters. In: Proceedings of the 2005 ACM SIGPLAN/SIGBED conference on languages, compilers, and tools for embedded systems (LCTES), pp 1–10
66. Yang CY, Chen JJ, Kuo TW (2005) An approximation algorithm for energy-efficient scheduling on a chip multiprocessor. In: Proceedings of the 8th design, automation and test in Europe (DATE), pp 468–473
67. Yang CY, Chen JJ, Kuo TW, Thiele L (2009) 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), pp 694–699
68. Ykman-Couvreur C, Hartmann PA, Palermo G, Colas-Bigey F, San L (2012) Run-time resource management based on design space exploration. In: Proceedings of the 8th IEEE/ACM/IFIP international conference on hardware/software codesign and system synthesis (CODES+ISSS), pp 557–566. doi:10.1145/2380445.2380530