Computing (2013) 95:537–566
DOI 10.1007/s00607-012-0236-6

Efficient inter-core power and thermal balancing for multicore processors

Juan M. Cebrián · Daniel Sánchez · Juan L. Aragón · Stefanos Kaxiras

Received: 9 May 2012 / Accepted: 31 October 2012 / Published online: 15 November 2012

Abstract Nowadays the market is dominated by processor architectures that employ multiple cores per chip. These architectures have different behavior depending on the applications running on the processor (parallel, multiprogrammed, sequential), but all happen to meet what is called the power and temperature wall. For future technologies (less than 22 nm) and a fixed die size, it is still uncertain the percentage of processor that can be simultaneously powered on. Power saving and power budget mechanisms can be useful to precisely control the amount of power been dissipated by the processor. After an initial analysis we discover that legacy power saving techniques work properly for matching a power budget in thread-independent and multi-programmed workloads, but not in parallel workloads. When running parallel shared-memory applications sacrificing some performance in a single core (thread) in order to be more energy-efficient can unintentionally delay the rest of cores (threads) due to synchronization points (locks/barriers), having a negative impact on global performance. In order to solve this problem we propose power token balancing (PTB) aimed at accurately matching an external power constraint by balancing the power consumed among the different cores. Experimental results show that PTB matches more accurately a predefined power budget (50 % of the original peak power) than other mechanisms like DVFS.

J. M. Cebrián (✉) · D. Sánchez · J. L. Aragón
Department of Computer Architecture, University of Murcia, Murcia, Spain
e-mail: jcebrian@ditec.um.es

D. Sánchez
e-mail: dsanchez@ditec.um.es

J. L. Aragón
e-mail: jlaragon@ditec.um.es

S. Kaxiras
Department of Information Technology, University of Uppsala, Uppsala, Sweden
e-mail: stefanos.kaxiras@it.uu.se

The total energy consumed over the budget is reduced to only 8 % for a 16-core CMP with only a 3 % energy increase (overhead). We also introduce a novel mechanism named "Nitro". *Nitro* will overclock the core that enters a critical section (delimited by locks) in order to free the lock as soon as possible. Experimental results have shown that *Nitro* is able to reduce the execution time of lock-intensive applications in more than 4 % by overclocking the frequency by 15 % in selected program phases over a period of time that represents a 22 % of the total execution time. We conclude the work with an analysis of the thermal effects of PTB in different CMP configurations using realistic power numbers and heatsink/fan configurations. Results show how PTB not only balances temperature between the different cores, reducing temperature gradient and increasing signal reliability, but also allows a reduction of 28–30 % of both average and peak temperatures for the studied benchmarks when a peak power budget of 50 % is exceeded.

Keywords Power consumption · Power budget · Power tokens · Chip multiprocessor

1 Introduction

Chip multiprocessors (CMPs) are the new standard in processor design for a wide range of devices, from small mobile devices to computation clusters. Whenever the number of processing cores is doubled, the power dissipation is also approximately doubled. Fortunately, technology scaling trends keep the dynamic power increase partially under control on each new generation. However, the complexity of the interconnection network and caches increases when more cores are incorporated into the die resulting in higher power dissipation. When a CMP runs a parallel multithreaded program where threads have dependencies (i.e., synchronization), if a traditional power saving mechanism is independently applied to a single core it can affect the rest of the threads in the next synchronization point. This may slow down the whole program execution and increase the overall energy consumption. We will require global information in order to avoid this situation by reducing energy mostly in threads that are not in the critical path of execution [1,2,8,12,14].

Moreover, since upcoming process technologies (under 22 nm) will fail to benefit from voltage scaling at the same pace as is the recent past, future many-core designs will be severely power and thermal constrained. We are inevitably heading to an era of dark silicon with fast and extremely dense chips that we cannot afford to power up [6]. The so-called *utilization wall* will limit the fraction of the chip we can use at full speed at once. In such context processors will run within a so tight power budget that only a small fraction of transistors could be activated simultaneously. Therefore, a research effort is needed to develop novel and more sophisticated techniques so power and temperature constraints do not strangle next-generation designs.

One possible solution to this *utilization wall* is specialization and/or heterogeneity. In a heterogeneous multicore with different components (GPU + CMP + Accelerators) an accurate power control of the different components will be required in order to be able to provide enough power to the processor (CMP power control is addressed in this paper). This is specially important when using different components simultaneously and also to control temperature, because an accurate power budget matching can reduce the temperature gradient[1] inside the cores and among the cores of the CMP. Another possible answer is to selectively throttle different cores of the chip balancing their aggregate power (proposed in this paper).

In addition, the continuous increase in the number of cores on a CMP is causing hotspots in inner cores, decreasing the reliability of the processor or even causing major damage to the die. A straightforward way to reduce temperature is to set a *power budget* to the processor [3,8]. It is important to note that the granularity level required to control temperature is much coarser than to control power and current. However, as we will see later, a good power control and balancing policy can lower temperature and reduce the processor temperature gradient without major performance degradation.

Summarizing, a processor's power budget can be used either to satisfy external power constraints (i.e., power cuts or shared power supply), to increase the number of cores in a CMP maintaining the same thermal design power (TDP), to reuse an existent processor design with a cheaper thermal package, to precisely control power dissipation of a sub-area of the die when matching a utilization wall, to control power domains in a heterogeneous multicore and/or to control temperature.

Dynamic voltage and frequency scaling (DVFS) [5,15,22] is a well known mechanism to make the processor's power converge to a given power budget. DVFS is based on fact that dynamic power depends on both voltage (quadratically) and frequency (linearly), so, if we lower any of these terms, we obtain a power reduction. However, DVFS exhibits some important drawbacks: (a) long exploration windows in order to compensate DVFS overheads; (b) when activated DVFS affects all instructions within a thread/core regardless of their usefulness to the forward-progress of a program; (c) voltage swings; and (d) energy overheads during DVFS changes. DVFS is a good mechanism to control temperature, as temperature variations happen during long time intervals, but we require additional accuracy levels in order to measure and control power and current.

In the CMP field we can find many specific proposals to match a predefined power budget such as [8,18,19], but these proposals are only suitable for CMPs running multiple single-threaded (or multi-programmed) applications, and have not been tested with parallel workloads. Moreover, they do not perform any accuracy analysis on the budget requirements (i.e., total energy consumed over the power budget). There are other works aimed at reducing the energy wasted when cores wait at synchronization points, either putting cores to sleep [14] or trying to make all cores reach the synchronization point simultaneously (e.g., *meeting points* [2] or *thrifty barriers* [12]). However, these mechanisms are not suitable for matching a power budget on their own, the main goal of this work. In order to overcome those limitations, Cebrián et al. [3] proposed the use of fine-grain microarchitectural power-saving techniques to accurately match a predefined power budget in a single-core scenario. However, when applied in a CMP scenario, these techniques still have a great impact on performance due to synchronization points.

[1] The difference between maximum and minimum die temperatures in a given time.

To address the shortcomings of previous proposals we introduce PTB [4], a mechanism that balances the CMP power among the available cores. When we set a global power budget to a CMP, local power budgets are applied to all running cores. Without any global mechanism the power would be just equally split between the cores. PTB globally manages power so cores that are under their local power budget give away their remaining allotment of power (up to the local budget) to cores over the budget so they can continue execution without performance degradation. PTB is based on the power imbalance that exists between cores due to cache misses, pipeline stalls, etc. In addition, PTB transparently benefits from thread's busy-waiting synchronization. When a core is waiting in a barrier it naturally reduces its power dissipation. PTB allows its spare power tokens[2] to be given to other cores doing useful work (i.e., critical threads). The same applies to locks: a core that enters a critical section receives extra power tokens from spinning cores. Thanks to the extra power tokens its local budget is less restrictive and the core can leave the critical section earlier.

Furthermore, when several cores enter a spinning state in a lock-delimited critical section they need to wait until the core currently in the critical section leaves it. If we are working in a power-constrained scenario using PTB, the core in the critical section will receive extra power from all spinning cores, and most likely will run at full speed because the additional power will prevent the core to restrain itself to match the power budget. However, some times, even at full speed, we still have power left to burn. To reuse this remaining power we introduce *Nitro*. *Nitro* reuses power to overclock the core that enters the critical section. During this time the core will run faster than usual, leaving the critical section earlier. This novel mechanism allows to obtain speedups with minimal overclocking run-time while ensuring that overclocking can be safely done since it is achieved by reusing power from cores under their local power budget. This approach can be used alone in any CMP but, as our major focus is a power-constrained scenario, we use *Nitro* as part of our global power balancing mechanism for matching a power budget (PTB).

At this point we want to summarize the main contributions of the current paper:

- Introduction of the PTB approach.
  - PTBs main goal is to make power dissipation go below a predefined power budget while maximizing accuracy on matching this budget in an energy-efficient way. Recall that, because of triggering power control mechanisms, there will be some performance and energy penalties but it is the purpose of our PTB approach to minimize them.
  - PTB is designed, but not limited, to work in a multicore (CMP) scenario running parallel workloads.
  - PTB is a fine-grain approach (unlike [2,12,14,19]), as it relies on real-time power estimations and not on coarse-grain time/performance estimations.
  - PTB can identify critical threads faster than [2,12,14], (the critical thread can change during execution) since it relies on fine-grain information (basic-block/cycle level), increasing its adaptability to the application behavior.

[2] These spare tokens represent the amount of power that the core can dissipate and still be under the power budget, will be defined in Sect. 3.2.

  - PTB is able to obtain greater reductions on both the average power dissipation and the average/peak chip temperature than other DVFS / microarchitectural approaches due to its increased power control resolution (tens of cycles) and the associated additional precision on matching the predefined power budget.
- Accuracy and performance analysis of previous proposals which are compared with PTB.
- Temperature analysis of different core and floorplan configurations is performed for PTB.
- Introduction of a novel dynamic overclocking mechanism, *Nitro*, which selectively overclocks cores that enter a critical section (delimited by locks) as long as the rest of the cores do not exceed the predefined power budget.

The rest of the paper is organized as follows. Section 2 provides some background on power-saving techniques. Section 3 describes our simulation methodology and shows a first analysis on the individual techniques and motivates the need for CMP-specific approaches to match the power budget. Section 4 reports the main experimental results. Finally, Sect. 5 shows our concluding remarks.

2 Background and related work

2.1 Single-core power control mechanisms

Dynamic voltage and frequency scaling (DVFS) has been widely used since the early 1990s [15] offering a great promise to reduce energy consumption in microprocessors. DVFS relies on the fact that dynamic power dissipation depends on both voltage and frequency ($P_D \approx V_{DD}^2 \cdot f$), and it dynamically scales these terms to save dynamic power [21,22,25]. In the recent years, researchers and designers have moved to multicore architectures as a way of maintaining performance scaling while staying within tight power constraints [5,8]. This trend motivates the need for fast per-core DVFS control. Kim et al. [11] recently proposed the use of on-chip regulators to achieve fast transition speeds of 30–50 mV/ns. However, as the building process goes into deep submicron, the margin between V_{DD} (supply voltage) and V_T (threshold voltage) is reduced, and as this margin decreases, the processor's reliability is reduced (among other undesirable effects). Moreover, the transistor's delay (switching speed) depends on: $\delta \approx 1/(V_{DD} - V_T)^\alpha$, with $\alpha > 1$. This means that we can lower V_{DD} for DVFS as long as we keep the margin between V_{DD} and V_T constant, so we can obtain the desired speed. However, the counterpart of reducing V_T is twofold: (a) leakage power increases as it exponentially depends on V_T, which makes leakage an important source of power dissipation as the process technology scales below 65 nm [7,9,10]; and (b) processor reliability is further reduced.

Sasanka et al. [20] proposed the use of DVFS and some micro-architectural techniques to reduce energy consumption in real time video programs. Winter et al. [24] proposed the use of a two-level approach that merges DVFS and thread migration to reduce temperature in SMT processors. More recently, in [3] we introduced the concept of power tokens in a single-core scenario along with a two-level approach that first applies DVFS as a coarse-grain approach to reduce power dissipation towards a predefined power budget, and then selects between different microarchitectural techniques to remove the remaining power spikes. The second-level mechanism depends on how far the processor is over the power budget in order to select the most appropriate microarchitectural technique. Experimental results show improvements in terms of both energy reduction and accuracy for a single-core scenario.

2.2 Multicore power control mechanisms

Multicore architectures exhibit some peculiarities when running parallel workloads, especially in terms of power and performance. In such workloads threads must periodically synchronize (e.g., for communication purposes) and any delay introduced in one of the threads may end up delaying the whole application.

2.2.1 Saving power from spinning

One of the main sources of useless power dissipation in CMPs running parallel workloads is "spinning" or "busy waiting". When a core is trying to acquire a lock or waiting in a barrier it enters in a spinning state that may become an important source of useless power. In order to detect spinning, initial approaches used source code or binary instrumentation but that requires recompilation and might be infeasible for certain situations. Li et al. [14] proposed a real-time hardware mechanism to detect processors in spinning state. Their mechanism checks the machine's state between instructions that cause a backward control transfer (BCT), usually a branch or a jump instruction. If the machine's state remains the same between several BCTs, the processor has entered a spinning state. They also propose scaling frequency for processors in a spinning state assuming that they can wake up a processor. However, this mechanism does not provide precise power management and cannot be applied outside locks/barriers.

2.2.2 Multicore processors and DVFS

In 2004, Li et al. [12] proposed *thrifty barriers*. This DVFS-based mechanism reduces power dissipation in CMPs by estimating the per-core time interval between synchronization points and disabling or using DVFS in cores that get to the synchronization point. They approximate the wakeup time by the time the slowest thread takes to get to the synchronization point. If the sleep/wakeup takes more time than they can save then the technique is not used. Later on, in 2006, Isci et al. [8] proposed a chip-level dynamic power management for CMPs just focusing on single-threaded programs while Sartori et al. [19] extended this work to reduce peak power in a distributed way. However, these proposals rely on the use of performance counters and/or coarse-grain time estimation and only work properly for multi-programmed or single-threaded applications. These power estimations can be unreliable due to spinning situations (spinning cores report high IPC). Moreover, none of the mentioned techniques provides any accuracy analysis when matching an imposed power budget. In 2008, Cai et al. [2] proposed *meeting points* which locates critical threads in parallel regions and uses DVFS to reduce energy consumption of non-critical threads. They achieve substantial energy reduction as long as the critical thread can be identified. In the commercial area, Intel's i7 turbo mode shuts off idle cores, reducing their voltage to zero rather than just lowering the power provided to them. Not having as many cores that produce heat will allow other cores to use more power, increasing the performance of those cores while still not exceeding the maximum TDP of the processor. This is useful when running sequential or low-parallel applications. However, for parallel workloads, overclocking two cores does not necessarily mean a performance improvement due to memory dependences and synchronization points.

3 Enforcing a power budget in CMPs

Most of the previous proposals in power control focus on global power or energy reduction. However, sometimes we require a precise core-level power/energy control during regular usage of the processor due to temporal power or thermal constraints. With this precise power control we can reuse existent hardware in a different scenario it was originally intended for, increase the number of computation cores reusing a TDP, ensure strict power control of different domains in an heterogeneous multicore, etc. Note again that temperature variations are not so fast as power variations, and thus power measurements to calculate temperature require less "resolution" than power measurements. In the present work we introduce PTB, a mechanism to restrain the power dissipation so that the processor can accurately match an imposed power budget in an energy-efficient way. To achieve this precise power control we first detect program points where power can be saved without harming performance (e.g., spin-locks, wrong execution paths, cache misses, etc.) and reduce it; second, we balance the power among the cores; and finally (when nothing else can be done), we reduce power locally even at the cost of performance degradation (by means of DVFS and/or microarchitectural techniques).

Generally speaking, we can distinguish three operation levels (Fig. 1) to estimate power. The first level (coarse-grain, Fig. 1-top) has low resolution (500 K-cycles) and low predictability, but the cost of measuring (or more precisely, estimating) power is low: usually performed by means of performance counters. This level can be useful for coarse-grain power saving mechanisms like DVFS and to perform temperature estimations. However, as illustrated by Fig. 1, if the average power of the search interval is under the power budget its low resolution can hide long periods where the processor is exceeding the power budget. In the initial *Power Token* approach for the single-core scenario (in [3] and further detailed in Sect. 3.2), we incorporated two additional levels of resolution to increase power predictability and to trigger microarchitectural power saving mechanisms. The second level power estimations will be performed at a basic block level[3] (tens of instructions, Fig. 1-mid). This level increases predictability as we can store the power dissipation of a previously executed basic block in the branch instruction that points to the beginning of the basic block. Next time we access the branch instruction we can obtain a power estimation for the next group of instructions. This power estimation can be used to decide which power saving mechanism should

[3] Group of instructions delimited by branches.

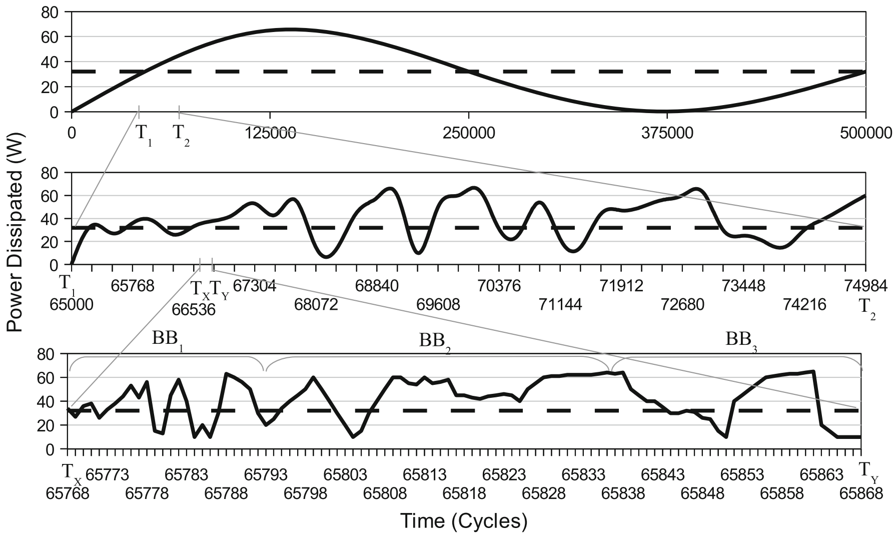


Fig. 1 Three levels of operation to estimate power, coarse-level (*top*), fine-level (*mid*), cycle-level (*bottom*)

be used depending on how far we are from a target power budget. Finally, most of the studied micro-architectural techniques perform power decisions at a cycle level. Therefore, we need to provide power estimations at a cycle-level (Fig. 1-bottom) for microarchitectural mechanisms to work properly. This section will present our simulation environment, analyze why legacy power control mechanisms are unable to match a power budget in CMPs and introduce the PTB approach.

3.1 Simulation environment

For evaluating the proposed approaches we have used the Virtutech Simics platform [16] extended with Wisconsin GEMS v2.1 [17]. GEMS provides both detailed memory simulation through a module called Ruby and a cycle-level pipeline simulation through a module called Opal. We have extended both Opal and Ruby with all the studied mechanisms that will be explained later. The simulated system is a homogeneous CMP consisting of a number of replicated cores connected by a switched 2D-mesh direct network. Table 1 shows the most relevant parameters of the simulated system. Power scaling factors for a 32 nm technology were obtained from McPAT [13]. To evaluate the performance and power dissipation of the different mechanisms we used scientific applications from the SPLASH-2 benchmark suite in addition to some PARSEC applications (the ones that finished execution in <3 days in our cluster). Results have been extracted from the parallel phase of each benchmark (initialization, screen outputs and result write-backs are removed). Benchmark sizes are specified in Table 2.

We will provide overall CMP energy consumption along with the *accuracy* of each evaluated technique on matching a predefined global power budget in the simulation results. To measure each technique's accuracy we define the metric Area over the Power Budget (AoPB). This metric measures the amount of energy (in joules) between

Table 1 Core configuration

Processor core	
Process technology	32 nm
Frequency	3,000 Mhz
V_{DD}	0.9 V
Instruction window	128 RUU + 64 IW
Load store queue	64 Entries
Decode width	4 inst/cycle
Issue width	4 inst/cycle
Functional units	4 Int Alu; 2 Int Mult
	4 FP Alu; 2 FP Mult
Branch predictor	16bit Gshare
Memory hierarchy	
Coherence Prot.	MOESI
Memory latency	300
L1 I-cache	64KB, 2-way, 1 cycle lat.
L1 D-cache	64KB, 2-way, 1 cycle lat.
L2 cache	1MB/core, 4-way, unified
	12 cycle latency
TLB	256 entries
Network parameters	
Topology	2D mesh
Link latency	4 cycles
Flit size	4 bytes
Link bandwidth	1 flit / cycle

Table 2 Evaluated benchmarks and working sets

Benchmark	Size	Benchmark	Size
SPLASH-2			
Barnes	8192 bodies, 4 time steps	Raytrace	Teapot
Cholesky	tk16.0	Water-NSQ	512 molecules, 4 time steps
FFT	256K complex doubles	Water-SP	512 molecules, 4 time steps
Ocean	258 × 258 ocean	Tomcatv	256 elements, 5 iterations
Radix	1M keys, 1024 radix	Unstructured	Mesh.2K, 5 time steps
PARSEC			
Blackscholes	simsmall	Swaptions	simsmall
Fluidanimate	simsmall	x264	simsmall

the power budget and each core dynamic power (represented by shadowed areas in Fig. 2). The lower the area (energy) the more accurately the technique will match the imposed power budget (note that the ideal AoPB is zero). However, this metric by itself is not enough in some cases. To illustrate this we can think of two scenarios, one

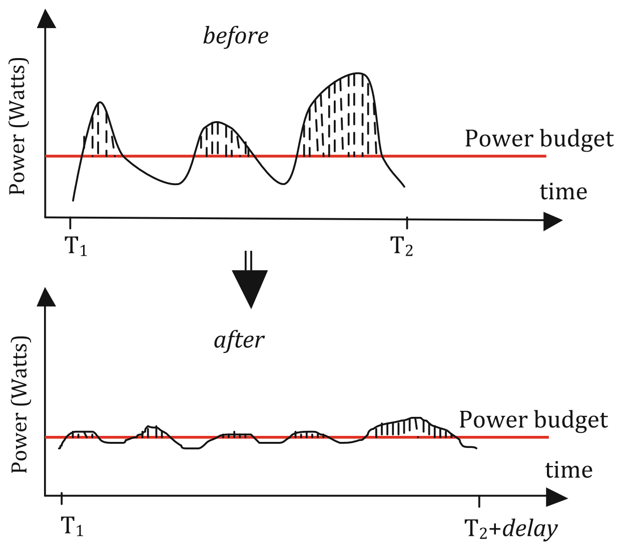


Fig. 2 Example of the area over power budget (AoPB) metric. *Shadowed areas* represent the energy consumed over the target power budget

where the processor is going over the PB by 100 % for 0.1 of the time and another where it is going over PB by 10 % for 1.0 of the time. These two scenarios will give the same area over PB value, but the first case could be much more harmful. In order to properly assess the benefits of the studied power saving mechanisms we will also provide, for our final results (Sect. 4.1), a population chart where we show the amount of energy spent in different power intervals on cycles over the power budget, that go from 50 % of the power budget to our processor peak power consumption. Performance results will also be shown for the dynamic power balancing approach (see Sect. 4.1).

3.2 Measuring power in real-time

Power tokens were introduced in [3] as a way to approximate the power being dissipated by the processor at a cycle level. The dynamic energy consumed by an instruction can be estimated at commit stage by adding to the base energy consumption of the instruction (i.e., all regular accesses to structures done by that instruction which are known *a priori*) a variable component that depends on the time it spends in the pipeline. A power-token unit is defined as the joules consumed by one instruction staying in the ROB[4] for one cycle. The number of power-tokens consumed by an instruction will be calculated as the addition of its base power-tokens plus the number of cycles it spends in the ROB. As in [3], the implementation of the Power-Token approach is done by means of an 8K-entry history table (Power-Token History Table—PTHT), accessed

4 Reorder Buffer.

by PC, which stores the per cycle power cost (in tokens) of each instruction's last execution (total energy/cycles) , and requires 5 bits per entry. The total size of the PTHT structure is small when compared with other processor structures such as the L1 or L2. The PTHT is updated with the current number of power-tokens consumed when an instruction commits. We calculated the base power-tokens of every instruction type (per op-code power) by running the SPECint2000 benchmark suite with the processor configuration shown in Table 1. Once we had the base power-tokens for all the possible instructions, we used a K-mean algorithm to group instructions with similar base energy consumption. The results presented in [3] showed that having just 8 groups of instructions is accurate enough for the Power-Token approach to properly work with an error lower than 1 % (compared to the actual energy consumption in joules as provided by HotLeakage). Although we assume equal distribution of the per-cycle power of an instruction in this approximation, this is not completely true. However, as we are simulating a full-pipelined out of order processor, this error is mostly hidden by similar instructions on a different stage. Hence, the overall processor power dissipation in a given cycle can be easily estimated based on the instructions that are traversing the pipeline without using performance counters just by accumulating the number of power-tokens (provided by the PTHT) of each instruction being fetched. Note that the extra energy consumption of the PTHT structure is also accounted in our results.

3.3 Matching a power budget in a CMP running parallel workloads

Once we have a mechanism to estimate power in a core, the next step is to analyze how different power saving mechanisms behave under power constraints in a multicore processor running parallel workloads. Initially, we will adapt and tune the proposed techniques in [3] to a CMP scenario. The evaluated techniques are:

- DVFS with five power modes (Voltage, Frequency): (100 % V_{DD}, 100 % f); (95 % V_{DD}, 95 % f); (90 % V_{DD}, 90 % f); (90 % V_{DD}, 75 % f); and (90 % V_{DD}, 65 % f). As we mentioned in Sect. 2.2.2 reducing supply voltage may require a threshold voltage reduction to obtain a certain switching speed (and that reduces reliability and increases leakage power), that is why we don't reduce V_{DD} any further than 90 %.
- DFS: Similar to (a) but only scaling down frequency. I.e., V_{DD} remains 100 % in all cases.
- Two-level: As in [3], this 2-Level approach uses DVFS to lower the average power dissipation towards the power budget and then uses different microarchitectural techniques to remove the remaining power spikes (2-level in the graphs). The studied mechanisms include instruction reordering based on criticality and fetch throttling based on branch confidence information and the ratio of decoded/committed instructions.

Microarchitectural mechanisms are enabled at a cycle level, but have effects after certain number of cycles, depending on the processor state. The decisions of what technique should be applied are taken at a basic block level.[5] DVFS is applied with an exploration window of 500 K cycles (check [3] for further details). All the techniques studied in [3] were designed for the single-core scenario and hence, they are applied at the core-level instead of at the CMP-level. Therefore, for this new scenario the first step should be to decide how to split the available power for the whole CMP (as determined by the global power budget) among the individual cores. An initial and straightforward implementation is to equally split the available power among all cores. In this case, power budget techniques will be locally applied to a particular core under two conditions:

1. The whole CMP is over the global power budget:

$$\sum Core_i\ Power > GlobalPowerBudget$$

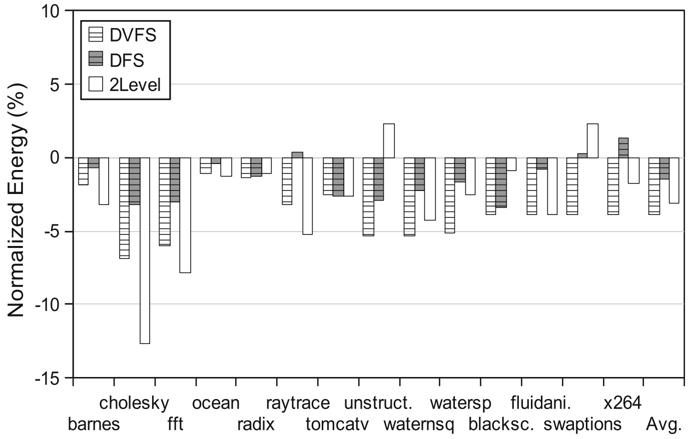
2. A particular core is over its local power budget:

$$Core_i\ Power > GlobalPowerBudget/\#Cores$$

To analyze whether the single-core mechanisms work properly in the CMP scenario we will apply the above power matching techniques (DVFS, DFS, 2-level) to a 16-core CMP (results for 2, 4 and 8 cores have been omitted for the sake of visibility and space limitation) for the SPLASH-2 benchmark suite and some PARSEC benchmarks with a global power budget set to 50 % of the original processor peak power consumption using clock gating. It is important to note that we have selected Kim's implementation [11] as a best case scenario for DVFS with a fast transition time of 30–50 mV/ns. Using a slower and more realistic DVFS will mean that microarchitecture-level techniques (used by 2-Level) will become even more accurate and energy-efficient than DVFS. DVFS is applied at a core-level to increase its accuracy when matching the power budget.

In Fig. 3 we can see the normalized energy and AoPB with respect to a base case where no power-control mechanisms are used (full speed) to match the global power budget. If we take a look at the energy numbers we can notice that all the evaluated techniques behave accordingly with the reported numbers in [3] for the single-core scenario. In benchmarks like Cholesky, the 2-Level approach is able to reduce energy by almost 13 %. In terms of performance, the average degradation is under 1 % for the studied benchmarks (although we don't show the figure due to space constraints). However, differences arise when looking at the accuracy metric (AoPB). Although there are particular benchmarks that report a reduced AoPB (depending on the evaluated technique—for example Blackscholes, Swaptions and x264 from PARSEC), the average AoPB is still very high. We obtained an average of 45 % AoPB, which is far from the average 10 % AoPB obtained for the single-core scenario in [3]. Moreover, for benchmarks like Ocean and Radix, the AoPB is especially high, around 70–80 %, which means that the global power budget constraint is not properly respected.

[5] As mentioned before, we can store information about past power behavior at a basic block level to predict future power trends and increase power matching accuracy.



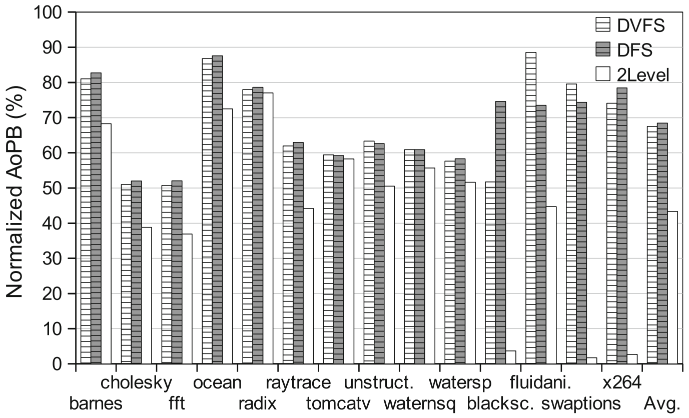


Fig. 3 Normalized energy (*top*) and AoPB (*bottom*) for a 16-core CMP with a power budget of 50 %

There is a key difference between single-threaded applications and parallel workloads: synchronization points. In parallel workloads we noticed that synchronization points alter the optimal execution of the code, so it is no longer dictated by each individual core, but by the whole CMP. These syncronization points are messing up the AoPB results. There are some benchmarks with low AoPB in Fig. 3-bottom, these benchmarks have no lock/barrier contention because they do not have synchronization points, so single-core mechanisms work properly for them. This initial analysis shows that previous mechanisms for managing power under temporary power constraints are not suitable for a CMP scenario when using this initial distribution policy that equally splits the power among cores.

3.4 Analysis on the power consumed in spinning

Figure 4 shows an analysis on the time spent by a CMP (addition of the data from individual cores) with a varying number of cores (from 2 to 16) either spinning or

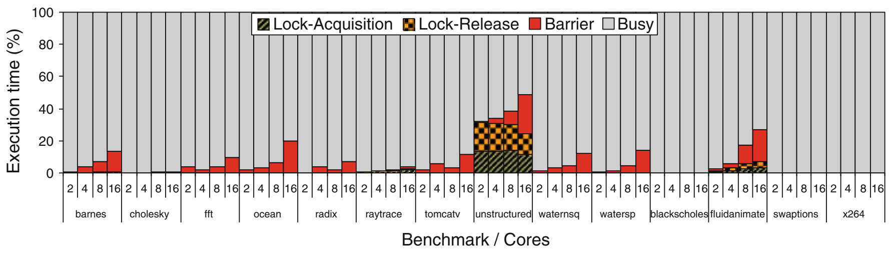


Fig. 4 Execution time breakdown for a varying number of cores

performing useful work. Each bar shows the fraction of time spent in lock acquisition, lock release, barriers, and useful computation (busy). As expected, the time each application wastes in spinning grows linearly with the number of cores. Some applications (Unstructured/Fluidanimate) spend a significant time in Lock-Acq and Lock-Rel states (contended locks) while others (Cholesky/Blackscholes/Swaptions/x264), in contrast, have no lock/barrier contention. The spinlock power is close to a 10 % on average for a 16-core processor running all the studied benchmarks. It is important to note that the key factor that causes legacy power saving mechanisms to fail is the existence of synchronization points, not how much power is wasted in spinning. A well balanced application with very little time wasted in spinning can waste a lot of power in spinning if we randomly slow down threads (create imbalance). In any case, these potential power savings due to spinning are not enough to accurately match a restrictive power budget (e.g., 50 % of the peak power) since (a) it is a small amount (10 %); and (b) spinning is located in very specific points over time while we are aimed at meeting the power budget constraint as long as it lasts. Therefore, we need a more generic approach that could benefit from other wasteful situations such as mispredictions events.

3.5 Power token balancing (PTB)

3.5.1 PTB motivation and fundamentals

Imagine a power-constrained scenario with a global power budget that we need to satisfy and local power budgets that individual cores try to match. Now that we can account power at a very fine granularity by means of the power tokens, we can find out situations where power imbalance exists among the cores of the CMP. Once detected we can balance their power and minimize performance degradation when matching the power constraint, and that is what PTB tries to achieve. Figure 5 shows an example where the total power dissipated by the CMP is over the global power budget, but some cores are under their power budget share. In this example we assume a 4-core CMP and global power budget of 40 W, with a simple implementation that equally splits power between cores, so each core has a local power budget of 10 W. We can notice that phases 1, 2 and 4 are over the global power budget (40 W), so we enable the local power-saving mechanisms. In phase 1 no power-control mechanisms are applied to cores 1 and 2, since they are under their local power budget (10 W). On the other hand,

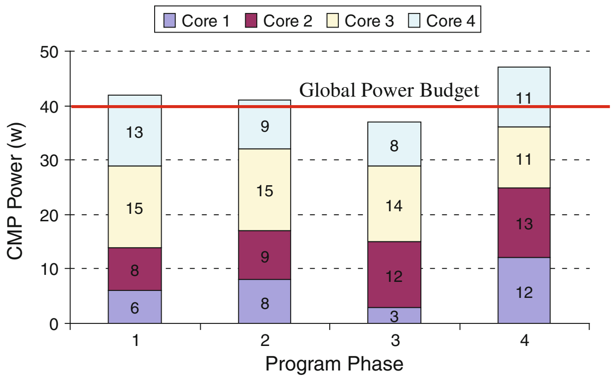


Fig. 5 PTB motivation (not real numbers)

cores 3 and 4 need to use their local power-saving mechanisms to match their local power budget. However, if those cores run critical execution threads and we slow them down, we could potentially harm performance and energy in a future synchronization point. The same happens in phase 2 with core 3. In phase 3, even though there are cores exceeding their local power budget, no mechanism is applied as the global CMP power is under the budget. Finally, in phase 4, all cores exceed their local power budget (so does the CMP exceeding the global one), and hence local mechanisms are applied to all cores. However, if we were able to tell cores 3 and 4 in phases 1 and 2 that their respective local power budgets are less restrictive than 10 W (since cores 1 and 2 have some power left, 4 + 2 W in phase 1; 2 + 1 W in phase 2) then the effects on the performance should be less harmful.

In PTB each individual core will count, at a cycle level, the number of power tokens it has consumed from the available local power tokens. In a given cycle, if a core still has available power tokens and the CMP is over the global power budget then the core offers its spare tokens to the PTB load-balancer (as detailed next in Sect. 3.5.2). Please note that, although this information is gathered at a cycle level, power variations take place over and during tens of cycles. This means that the situation where a core goes under the power budget and decides to give tokens to the PTB load-balancer will usually repeat during the next tens of cycles. Tokens are used as a currency to account for power, so it is important to note that they are neither sent nor received. In PTB cores just send the number of spare tokens. Analogously, cores over their local power budget will receive extra tokens from the PTB load-balancer which will prevent them to enable a power-saving technique (that can reduce performance) as long as the global power budget constraint is met. The PTB load-balancer calculates every cycle the overall available power tokens based on the spare tokens that cores have for each cycle. Therefore, PTB is not a loan/refund mechanism since a core can reuse power from others but there is no need to give it back. In PTB we define two power distribution policies that will be discussed later: (a) give tokens to the most power-hungry core (*ToOne* policy); or (b) equally distribute the extra tokens among all cores over the power budget (*ToAll* policy). For the later, if the number of tokens is not a multiple of the power hungry cores a round robin distribution is used for the remaining tokens. PTB also exhibits two inherent features that allow "transparent"

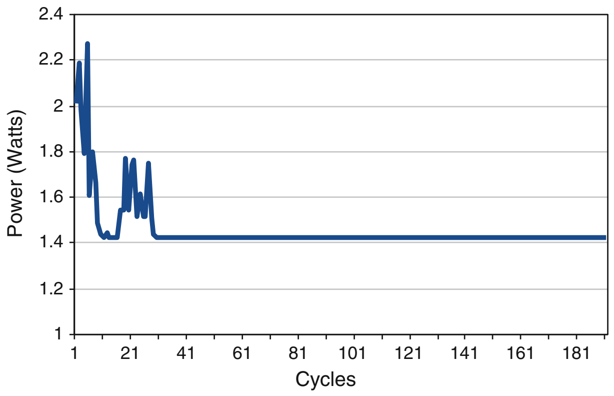


Fig. 6 Per-cycle power behavior of a spinning core

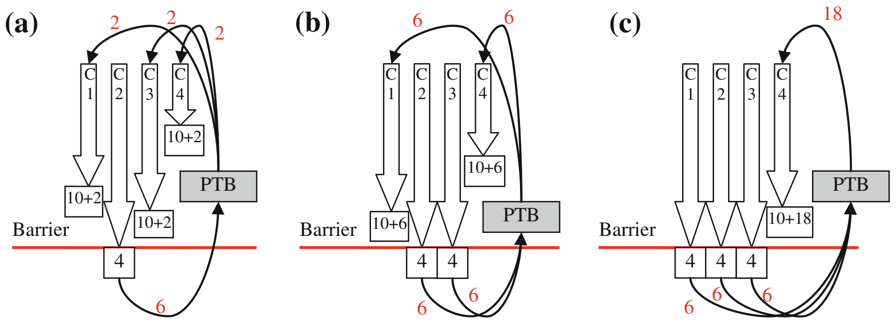


Fig. 7 Power token balancing (PTB) example in the case of a barrier (using the *ToAll* policy)

optimizations without any specific mechanism: (1) indirect spinning detection, and (2) an automatic priority system for non-spinning threads.

Figure 6 help us to illustrate how PTB could be used to detect spinning. When a core enters a spinning state, the dynamic power follows the behavior shown in Fig. 6. In this figure we can see an initial power peak due to useful computation. If the spinning state lasts enough, the pipeline empties and power goes down and stabilizes (cycles over 35 in Fig. 6) to an amount that is usually under the budget. We can assume then that the core is spinning. Note that spinning is just a particular case of power imbalance, so our mechanism will benefit from it but that is not the only case. Remember that PTB knows nothing about locks, barriers, etc, it just balances power.

For illustrating purposes and continuing with the spinning example, Fig. 7 shows how PTB works in the case of a barrier (using the *ToAll* policy). For this example let us assume there are four cores (C1 to C4) with local power budgets set to 10 tokens and that when spinning a core consumes 4 tokens. As cited before, a spinning core gives its spare tokens (TotalAvailableTokens - UsedTokens) to the PTB load-balancer. Figure 7a shows that core 2 reaches the barrier and transfers 6 tokens to the load-balancer. Now the rest of cores have more available power left to burn until they get to the synchronization point (in our example, cores 1, 3, 4 receive 2 extra tokens each from the load-balancer, raising their local budget to 12 tokens). When any other core (e.g., core 3 in Fig. 7b) reaches to the barrier it also gives 6 spare tokens to the PTB load-balancer which allows cores 1 and 4 to use the 6 + 6 extra tokens from cores 2 and 3, raising their local budget to 16. Finally, Fig. 7c shows core 1 spinning in the barrier and giving its six spare tokens to the PTB load-balancer which prevents the last core (C4) to be slowed down as it can use all the spare tokens. Again, note that PTB does not explicitly distinguish between barrier- or lock-spinning. It is important to recall that PTB basically detects power imbalance among cores and also benefits from any misprediction event (e.g., a cache miss or a mispredicted branch).

3.5.2 PTB implementation details

The implementation of PTB is based on a centralized structure called PTB load-balancer. This structure receives the number of spare power tokens from all cores under their local power budget and splits them among the cores exceeding it (intending not to trigger any power-saving mechanism for the exceeding cores which would result in a performance degradation). Power balancing is performed every cycle, so tokens from previous cycles are not stored in the balancer. However, token exchange takes some cycles. Cores are not stalled during this time, but the core that gives away tokens sets a more restrictive power budget (its original power budget minus the given tokens) until tokens reach their destination. This is usually not a major issue as power trends don't change that quickly, and if a core gets under the power budget it will remain there for tens of cycles. To exchange token information we need to build communication wires between the cores and the PTB load-balancer. We will use four wires for sending and four wires for receiving the number of tokens per core; this limits the amount of given/received tokens but makes the mechanism more power-efficient. Note that these wires are used to send the amount of spare tokens (tokens are used as a currency to account for power). All these wires will be placed on a different layer of that of the interconnection network.

To estimate latency delays of the communication wires we used Xilinx ISE for a processor running at 3 GHz without buffers as a reference to calculate the logic delay of the circuit. We removed the delay caused by both pins and routing, making the logic delay almost equivalent to the delay of a circuit in an ASIC implementation. For a 4-core CMP delays are: one cycle for sending the number of spare tokens, one for processing tokens and one for sending the number of spare tokens back to the cores over the power budget. For an 8-core processor, wire delay increases to 2 cycles, so it will take a total of 5 cycles to send and receive the number of spare tokens to/from the PTB load-balancer. For a 16-core CMP the mechanism needs 4 cycles for receiving the tokens, 2 cycles for processing and 4 cycles for sending the tokens to the cores over the power budget, according to Xilinx ISE. When a core gives away tokens it sets a more restrictive power budget to ensure it won't consume additional power until tokens reach its destination (i.e. 10 cycles for a 16-core CMP). The power dissipation of the PTB mechanism plus the communication wires has been estimated using Xilinx XPower Analyzer with the same configuration as the delay latency, increasing the average application energy consumption by just 1 %, which is also accounted in the experimental results presented in the next section.

Problems might arise as we increase the number of processing cores, and thus, the PTB load-balancer communication and processing latencies. For the analyzed number of cores and latencies, experimental results show significant improvements in terms of temperature, energy and accuracy on matching the power budget, even with a pessimistic 10-cycle delay for sending/receiving the number of spare tokens from other cores. However, when the number of cores increases latencies over 20 cycles, power variations affect performance of PTB negatively. Nevertheless, one approach to make PTB more scalable (over 32 cores) consists of clustering the PTB load-balancer into groups of 8 or 16 cores and replicate the structure as needed. Results in next section will show that such a group of cores (8 or 16) is enough for PTB to efficiently balance power and accurately match the imposed power budget.

3.6 Reusing power to reduce energy: *Nitro*

As mentioned before, when running parallel workloads on a CMP, speeding up or slowing down a specific core may not vary the final program execution time due to synchronization points. For example, overclocking a core may not lead to any performance improvement. The key point in a CMP running a parallel application is that, in general, it is more crucial *when* you apply the mechanism than the mechanism itself. The idea behind *Nitro* is quite simple: save power when a core does not need it (e.g., while spinning) and reuse it when it becomes really useful (e.g., critical threads/sections). *Nitro* differs from other spinning-based mechanisms [2,12,14] in two things. First, all these mechanisms are meant to reduce energy consumption before/while spinning, whereas *Nitro* tries to reuse this energy somewhere else. Second, all these mechanisms usually try to exploit the barrier synchronization mechanism, while *Nitro* can benefit from locks and barriers. *Nitro* is designed for parallel workloads, looking for code sections that will benefit from local overclocking.

In Sect. 2.2.1 we mentioned that spinning states can be detected either by hardware or by static instructions introduced by the programmer (the later is out of the scope of this paper). In any case, those sections can be identified and, by using the power-token approach, we can approximate the amount of energy wasted in spinning (depending on how long a core stays spinning). However, a small structure to account for total power-tokens saved is still needed. *Nitro* works as follows: once a lock is detected, the processor that gets access to the lock after the contention period (lock acquisition) is overclocked, as long as we have power-tokens left to overclock (from other spinning cores) and for as long as the critical section lasts. The use of power from spinning cores ensures that the overall power of the processor remains under it's TDP, and temperature is kept under safe limits (similar to Intel's Turbo Boost). We will assume the proposed DVFS by Kim et al. [11] which is able to quickly switch between power modes at speeds of 30–50 mV/ns. The overclocked core will run at a 15 % faster rate than the base frequency. Of course, sometimes there are not enough cycles in the critical section to take advantage of *Nitro*, so we need some kind of mechanism to estimate the duration of the critical section and only apply *Nitro* if it lasts for long enough. For this purpose we use the spinning predictor proposed in [12], based on the PC to predict a critical section duration. The overclocking will also last for a short period of time, so the processor has time to recover after the overclocking. *Nitro* can also be applied to speed up the remaining threads while others wait in a barrier, but we did not implement this feature as there were many studies about overclocking/underclocking threads in barriers to optimize energy. Please note that *Nitro* tries to reuse unused power from spinning cores, so it does not violate the power budget of the whole CMP, only individual cores for a short period of time. The key difference with Intel's Turbo Boost is that the later disables idle cores to overclock one core, for the benefit of sequential applications. *Nitro* is a more fine-grain approach and can act at a critical section level or even when cores are waiting in a barrier, thanks to the cycle level accurate power information provided by power tokens.

3.6.1 Nitro outside locks

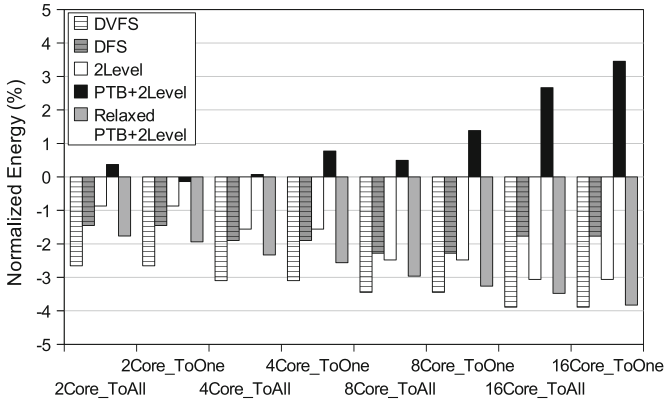
Nitro was designed to benefit from wasted power from spinning cores, saving that power and reusing it somewhere else. We decided to focus our analysis in lock-delimited code sections, because there were many previous works that focused on reusing or balancing power from spinning cores in barriers [2,12,14,19]. However, with minimal modifications on the PTB mechanism, *Nitro* can be used to speed up execution in both barriers and locks. If we take a look at the example in Fig. 7c, once cores 1, 2 and 3 get to the barrier, the PTB load-balancer receives all their spare tokens, and gives them to core 4. As there is only one core left, it probably won't even use any power saving mechanism, either because the total power dissipated by the whole CMP will be under the global budget (one of the requirements from Sect. 3.3) or because the extra spare tokens from the PTB load-balancer prevent the usage of local power saving mechanisms. Under these conditions we could overclock core 4, the last one to get to the barrier, in order to reach the synchronization point faster, and thus reduce global execution time. The overclocking should only be done as long as we have power tokens left from the PTB load-balancer.

More specifically, if PTB is using the *ToOne* policy, *Nitro* will overclock the most power hungry core (note that this core is not necessarily the latest to get to the synchronization point), speeding it up as long as we have tokens left from spinning cores. On the other hand, if PTB is using the *ToAll* policy, power tokens will be equally divided between all the cores over the budget, and, eventually, all these cores will receive enough tokens to overclock themselves, speeding up more and more as cores reach the synchronization point. We did not implement *Nitro* outside locks in the present work, but we will evaluate the theoretical potential of this mechanism.

4 Experimental results

4.1 Efficiency of power token balancing (PTB)

In this section we perform an analysis of the PTB mechanism with the previously defined power-token distribution policies: *ToAll* (that shares the power-tokens among all the cores over their local power budget) and *ToOne* (that gives all the spare power-tokens to the core that needs them the most). The selected global power budget will



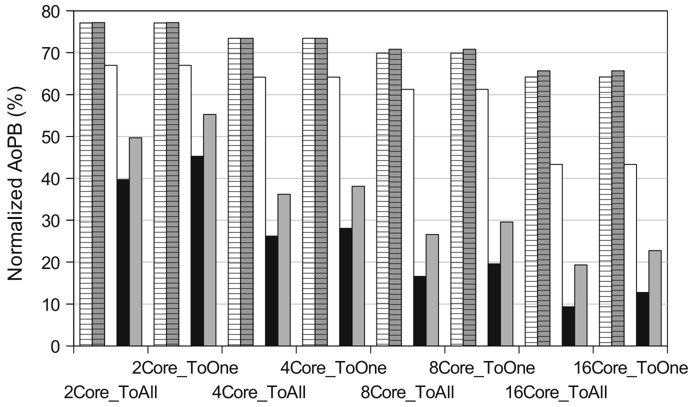


Fig. 8 Normalized energy (*top*) and AoPB (*bottom*) for a varying number of cores and PTB policies

be 50 % of the peak power consumption[6] with clock gating and a varying number of processing cores in the CMP (2–16 cores). This restrictive power budget is a worst case scenario that forces a high usage of power saving mechanisms. Results are normalized to a base case without power-control mechanisms.

Figure 8-top shows the energy consumption for the evaluated techniques (enumerated in Sect. 3.3) using different combinations of core number/policy whereas Fig. 8-bottom shows the AoPB metric. We can observe that when using the proposed PTB mechanism, area numbers go back to the reported numbers in the previous work for the single-core scenario: average 8 % of AoPB for a 16-core CMP when the PTB + 2-level technique is used (although energy numbers are not as good as for the single-core scenario). In a 16-core CMP, DVFS and DFS[7] are unable to lower the AoPB below 65 % while PTB + 2-level reduces the average area to just 8 %, getting close to the ideal AoPB of zero. It can also be observed that the

[6] We only report results for a 50 % power budget due to space limitations for the sake of visibility. For less restrictive power budgets PTB also works properly.

[7] DVFS and DFS are applied at a core-level to increase its accuracy when matching the power budget.

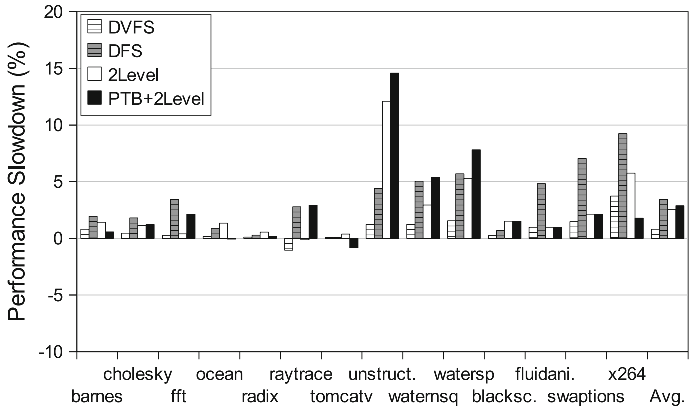


Fig. 9 Detailed performance for a 16-core CMP using the dynamic policy selector

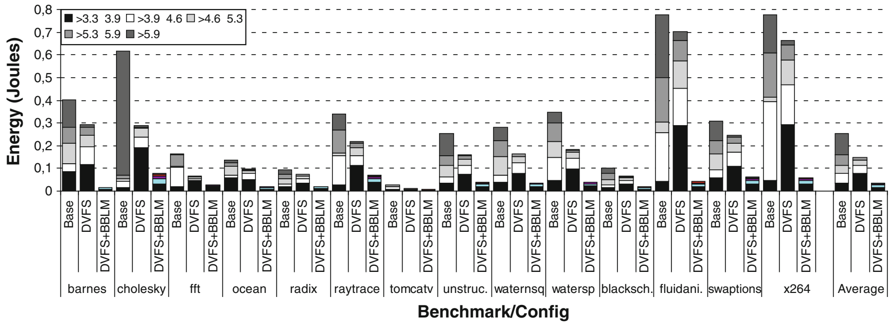


Fig. 10 Energy distribution on cycles over the power budget for different mechanisms

accuracy on matching the power budget increases (i.e., AoPB decreases) with the number of cores, because we have more chances of receiving tokens from other cores.

For a 16 Core CMP, DVFS shows an average energy reduction of 6 % whereas PTB increases the energy in 3 %. Note, however, that this energy increase can be turned into energy savings if we relax the accuracy constraint of PTB, as we will show in Sect. 4.3. In terms of performance (Fig. 9), Unstructured is the application that is more affected by the microarchitectural power-control mechanisms. Finally, Fig. 10 shows the energy (AoPB) distribution (population chart) of different power intervals when we request to match a power budget of 50 %. We can clearly see how PTB plus a 2-Level mechanism is able to almost completely eliminate the exceeding energy (area) over the power budget, ensuring that almost no cycle exceeds 4.6 W when a 3.3 W power budget is set. Meanwhile DVFS, due to its coarse grain nature, is unable to properly adapt to the power variations, and still shows some cycles with power peaks over to 5.9 W. Note that the analyzed dynamic policy selector, for the presented results, is assisted by actual application-specific information although pure indirect dynamic detection of the type of spinning is possible (and practical) via heuristics similar to those described in [14].

4.2 The importance of accuracy

Performance degradation that results from using our proposed PTB mechanism if the application is parallel enough to use these extra cores.

In this section we will show an example that illustrates the importance of the accuracy on matching a predefined power budget when moving from a homogeneous CMP design into a heterogeneous CMP maintaining the same TDP (Thermal Design Power). Imagine we have a 16-core CMP design that we want to reuse with a 100 W TDP. In this design each core would use 6.25 W (for simplicity let us ignore the interconnection network). If we set a power budget of 50 % we could ideally use 50 W for other types of accelerators, such as GPU, FPGA, etc. But for this ideal case a perfect accuracy on matching the power budget is needed.

According to the previous results, DVFS incurs in an energy deviation of 65 % from the target power budget (the AoPB metric). As the average performance penalty of the studied mechanisms is minimal, we can extrapolate similar per cycle power deviation. Therefore, with such 65 % deviation each core dissipation may raise up to $3.125 \times 1.65 = 5.15$ W, meaning that for a 100 W TDP we can reuse a maximum of 17.5 W for the accelerators ensuring that the total per cycle power will remain under the TDP, and that we can provide enough current to all components. Using a regular 2-level approach (without PTB) the deviation is reduced to 40 % that gives us an average power dissipation of $3.125 \times 1.40 = 4.375$ W per core, so we can have 30W available for accelerators with the same TDP. Finally, when using the non-relaxed PTB approach the error is reduced below 10 %, that gives us a potential average power dissipation of $3.125 \times 1.1 = 3.4375$ W per core, so we could reuse 45 W for accelerators. Therefore, thanks to the extra hardware we can perfectly overcome the 3 % performance degradation that results from using our proposed PTB mechanism if the application is able to use these extra resources.

4.3 Relaxing PTB to be more energy-efficient

Up to this point we have focused on a PTB mechanism that maximizes the accuracy on matching the given power budget. As explained before, this kind of optimization hurts individual core performance and, therefore, increases overall energy consumption. If the conditions are not so restrictive (if we want to control temperature for example) or if we cannot reuse this saved power (obtained by setting a power budget) to overcome the energy penalty we may be interested in reducing the energy penalty. If we relax the accuracy constraint, PTB can also achieve positive energy savings since power-saving mechanisms would be applied in a less restrictive way, not affecting performance that much. In order to analyze this new focus, Fig. 8 (rightmost bar) shows how PTB behaves when optimizing for energy-efficiency instead of just accuracy for a power budget of 50 %. For this experiment we are assuming and AoPB threshold of +20 %. This relaxed threshold is used to delay triggering a power-saving mechanism when the global/local power budgets are exceeded. Note that the original PTB may trigger the power-saving mechanisms *immediately* after detecting that the power budget was exceeded, if the second level power prediction suggested so (basic block power prediction).

If we take for example a 16-core CMP and relax the AoPB metric allowing it to be 20 % above the power budget, PTB obtains an average energy reduction of 4 % (Fig. 8-top) similar to that obtained by DVFS, and still being far more accurate than per-core DVFS on matching the power budget (as seen in Fig. 8-bottom). Of course, better energy savings could be achieved for the same 16-core CMP if we relax the area constraint even more. Finally, if we use PTB as a spinlock detector to disable the spinning cores we could further increase the energy savings. similar to those described in [14].

4.4 Temperature analysis

Thermal hotspots increase cooling costs and have a negative impact on reliability and performance. The significant increase in cooling costs requires designs for temperature margins lower than the worst-case. When we reduce the per-cycle power consumption of an application we can consequently reduce the CMP temperature over time. Moreover, leakage power is exponentially dependent on temperature and an incremental feedback loop exists between temperature and leakage, which may turn small structures into hotspots and potentially damage the circuit. High temperatures also adversely affect performance, as the effective operating speed of transistors decreases as they heat up. In this section we will analyze the per-structure and per-benchmark temperature effects of PTB and DVFS when comparing to the base case. Please note that temperature reductions are simply the result of the combined power reduction from the selected power saving mechanisms, what we try to illustrate in this section is the temperature variability from two different mechanisms that work at different power granularity, both across cores inside the die and structures inside the same core. In addition we will give some insight of how much temperature reduction can we expect from setting a power budget to the processor.

Temperature numbers were obtained by introducing the HotSpot 5.0 [23] thermal models into Opal and building our tiled CMP by replicating N times our custom floorplan (depicted in Fig. 11), where N is the number of cores. We simulate a 14-stage, out-of-order core. Power and area numbers for this core configuration were obtained through the McPAT [13] framework. We then input these power numbers into GEMS-Opal to obtain preliminary average power numbers of each structure using a modified version of the Wattch implementation included in GEMS-Opal (minor bugfixes). Using these average power numbers we build the input files for HotSpot—HotFloorplanner that will generate the core floorplan. In addition to the power numbers, HotSpot also needs per-structure area information (provided by McPAT) and structure communication needs (we used Alpha 21264 structure dependences as our input for HotFloorplanner). More specifically, we have modeled both leakage (through McPAT) and the leakage/temperature loop in Opal, so leakage will be updated on every Hotspot exploration window (10 K cycles). Leakage power is translated into power tokens and updated according to the formula $L_{new} = L_{Base} \cdot e^{Leak_{\beta} \cdot (T_{current} - T_{base})}$ where $Leak_{\beta}$ depends on technology scaling factor and is provided by HotSpot 5.0, L_{new} is the updated leakage, L_{Base} is the base leakage (obtained using McPAT thermal models), $T_{current}$ is the current temperature and T_{base} is the base temperature. Once

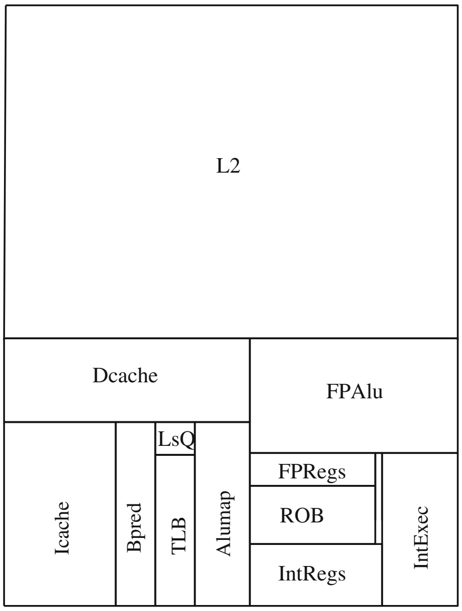


Fig. 11 Core floorplan

leakage is updated, it is translated back to *power tokens*. Another important parameter is the cooling system. The regular thermal resistance of a cooling system ranges from 0.25 K/W for the all-copper fan model at the highest speed setting, to 0.33 K/W for the copper/aluminum variety at the lowest setting. We model a real-world Zalman CNPS7700-Cu heatsink with 0.25 K/W thermal resistance and 3.268 cm^2 area (136 mm side).

Figure 12 shows both average and peak (maximum) temperatures before using PTB for the studied benchmarks running on a 16-core CMP along with their corresponding standard deviation. We define "idle" temperature as the temperature of the whole CMP in idle state (i.e., only the operating system is running). Therefore, the maximum temperature reduction any power saving mechanism can achieve will vary between the base peak/average temperature of the CMP and the idle temperature. For the studied 16-core CMP the idle temperature reported by McPAT is around 60 °C (red line in Fig. 12). In Fig. 12 we also see that the average temperature is 72 °C for the evaluated benchmarks, therefore, the temperature reduction we could aspire is 12 °C on average.

Figure 13 shows a comparison between the minimum temperature of the CMP without PTB (coolest core) against the peak temperature of the CMP when using PTB (hottest core). This initial study shows how PTB is able to balance temperature between the cores, lowering the peak temperature of the hottest core of the CMP to almost equal the temperature of the coolest core in the base CMP (without PTB). This is the major benefit of the PTB mechanism that ensures minimal deviation from the target power budget and, therefore, temperature.

For a more detailed analysis, Fig. 14 shows the per-structure peak and average temperatures for a 16-core CMP. Temperatures are normalized against the maximum

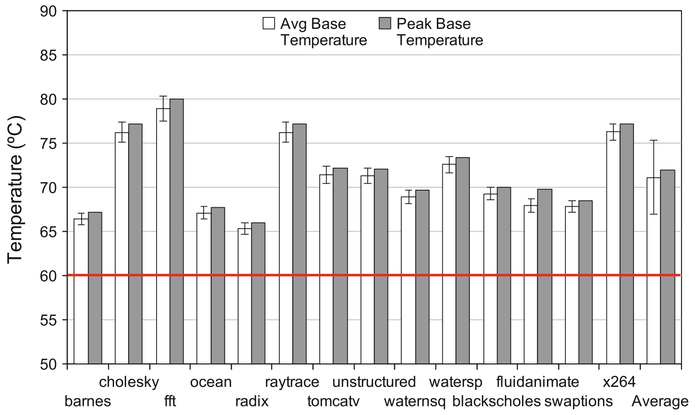


Fig. 12 Average and peak temp. of a 16-core CMP

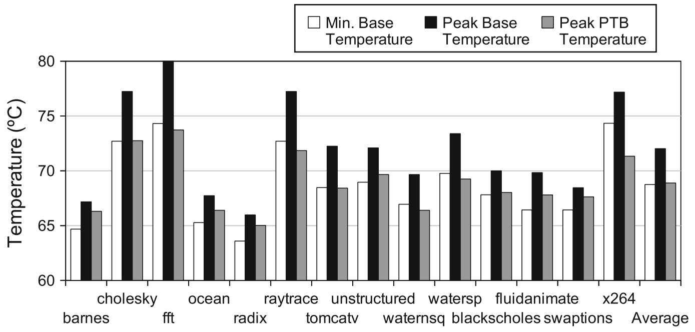


Fig. 13 Minimum base versus peak base versus peak PTB temperatures of a 16-core CMP

temperature gain: peak (or average) temperature *minus* idle temperature. We can see how PTB and DVFS are able to reduce the temperature of all the internal structures of the core. However, PTB almost doubles the temperature reduction of DVFS, due to the extra accuracy when matching the target power budget. In particular, PTB obtains an average reduction of the peak temperature of all structures of 35 % in addition to an average reduction of the average temperature of 30 % for the evaluated 16-core CMP. On the other hand, the per-benchmark temperature reduction achieved by PTB follows the same trend, as it can be observed in Fig. 15. We can see that, except for fluidanimate, there exists a temperature reduction in all of the studied benchmarks, for both peak and average temperature during the benchmark execution. The peak and average temperature reductions provided by the use of PTB are again close to 27 % on average, doubling the temperature reduction of DVFS.

Please note that, although we are not trying to match a temperature budget, but a power budget, the effects on temperature from this accurate power budget matching are extremely good for temperature (both average and peak) and temperature gradient. If we were trying to match a temperature budget, using PTB + 2-Level mechanism would

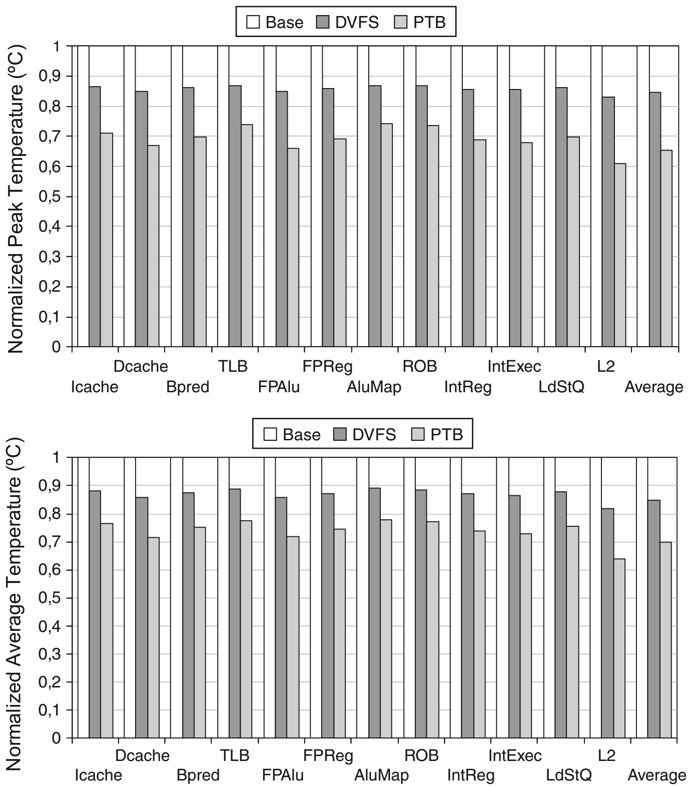
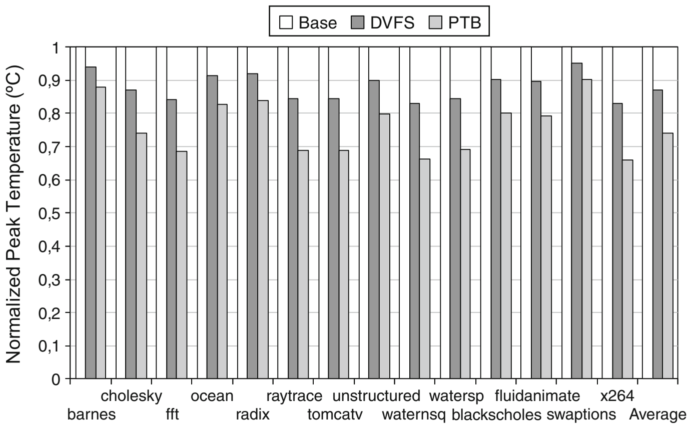


Fig. 14 Normalized per-structure peak (*top*) and average (*bottom*) temperature analysis

increase energy efficiency, reduce the time it takes to get to the target temperature and the temperature gradient over DVFS.

4.5 *Nitro* energy and performance analysis

Finally, we have evaluated *Nitro* for a varying number of cores (from 2 to 16) running the SPLASH-2 benchmark suite and some benchmarks from the PARSEC 2.1 suite. We will assume the proposed DVFS by Kim et al. [11] which is able to quickly switch between power modes at speeds of 30–50 mV/ns. Figure 16 shows the performance improvement (execution time) and the normalized energy reduction over the base case without power restrictions running at full speed for the different benchmarks. As expected, the benchmark that benefits the most from *Nitro* is Unstructured, which is the one that has the most lock contention from the set of studied benchmarks. Note, however, that this mechanism does not cause a heavy impact on energy in the rest of studied benchmarks. For Unstructured, the number of overclocked cycles represents about 22 % of the total simulation cycles, and that is enough to reduce the energy consumption of this benchmark by 3–3.5 %. We also obtain 3.5–4.5 speedups from a potential 3.5–4.5 % according to Amdahl's law, 15 % frequency increase on a



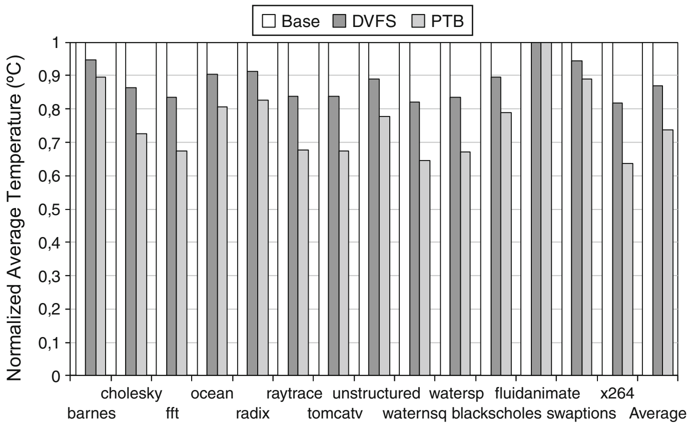


Fig. 15 Normalized per-benchmark peak (*top*) and average (*bottom*) temperature analysis

22–30 % of the code. However, both the SPLASH-2 benchmark suite and the studied PARSEC benchmarks are quite optimized and contention periods are kept as low as possible, especially the ones related to locks, but this is not always the case in parallel applications such as for commercial and server workloads.

Nevertheless, results obtained by the *Nitro* approach are encouraging since they show that, when contention for lock acquisition and release exists, our proposal can improve both energy and performance. Moreover, the extension of *Nitro* to work with both barrier and lock information could achieve even further energy savings. The fraction of time cores spend in spinning increases with the number of cores. For a 16-core CMP we saw (Fig. 4) an average 16 % barrier time, that will exponentially increase with the number of cores. The potential speedup for this scenario is limited by Amdahl's law to 2.4 % (15 % frequency increase on 16 % of the execution time). However, for 32 or 64 core processors with barrier times greater than 40 % this potential speedup increases to >6 %. This also translates into energy savings as Nitro is reducing execution time reusing wasted power, without additional power overhead.

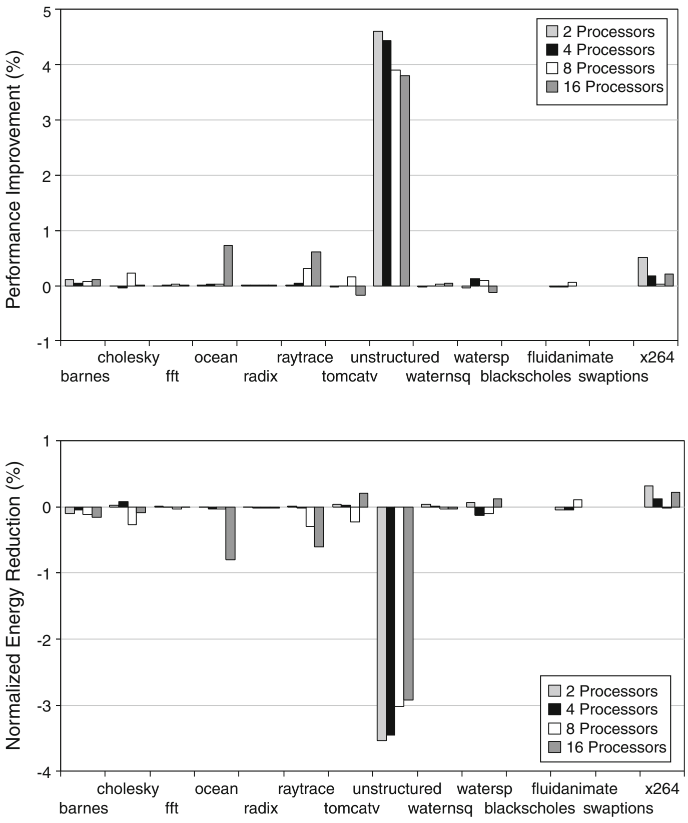


Fig. 16 Performance improvement and energy reduction for a 16-core CMP using *Nitro*

5 Conclusions

We are inevitably heading to an era of dark silicon with fast and extremely dense chips that we cannot afford to power up. The so-called utilization wall will limit the fraction of the chip we can use at full speed at once. In such context processors will run within a so tight power budget that only a small fraction of transistors could be activated simultaneously. In this paper we have proposed PTB, a mechanism that dynamically balances power between the cores to ensure that the processor accurately matches a global power budget, even if there is a performance/energy penalty. Remember that PTB knows nothing about locks, barriers, etc, it just balances power. Experimental results show that PTB is able to accurately match the global power budget with an AoPB of just 8 % for a 16-core CMP with a negligible energy increase (3 %) while DVFS fails to match the power budget precisely, resulting in a high AoPB of around 65 % when setting a power budget of 50 % on the base processor peak power.

As a side effect of this accurate power budget matching, the use of PTB provides another interesting benefit: a more stable temperature over execution time. We obtain a 27–30 % peak and average temperature reduction when setting a power budget of 50 % on the base processor peak power for the studied benchmarks, that also applies to the individual structures. PTB is also able to balance temperature between cores, reducing the peak temperature of the hottest core making it equal to the temperature of the coldest core in the base CMP design. This temperature gradient reduction not only reduces leakage power but also increases reliability and reduces packing costs. Moreover, if we tried to match a temperature budget, a relaxed PTB can beat DVFS, at both accuracy, energy degradation and time to get to the target temperature.

Finally, we have proposed *Nitro*. This technique reuses power from spinning cores to overclock cores that are executing critical sections of the program (e.g., code delimited by locks). This mechanism is suited for applications with contended locks and does not degrade performance for the rest of applications. Note that *Nitro* tries to reuse unused power from idle or spinning cores, so it does not violate the power budget of the CMP, only individual cores for a short period of time. Unfortunately, both the studied benchmark suites are optimized to minimize contention, and only one benchmark exhibit high lock contention periods. However, other regular programs with more coarse-grain locks will provide higher improvements. Moreover, we can expect greater energy savings if we extend *Nitro* to work with barriers, speeding up the working cores using the energy from the spinning cores.

Acknowledgments This work was supported by the Spanish MEC, MICINN and EU Commission FEDER funds under Grants CSD2006-00046 and TIN2009-14475-C04. Also by the EU-FP7 ICT Project "Embedded Reconfigurable Architecture (ERA)", contract No. 249059.

References

1. Bhattacharjee A, Martonosi M (2009) Thread criticality predictors for dynamic performance, power, and resource management in chip multiprocessors. In: Proceedings of the 36th annual international symposium on computer architecture, ISCA '09, pp 290–301. ACM, New York, NY, USA. http://doi.acm.org/10.1145/1555754.1555792
2. Cai Q, Gonzalez J, Rakvic R, Magklis G, Chaparro P, Gonzalez A (2008) Meeting points: Using thread criticality to adapt multicore hardware to parallel regions. In: Proceedings of the international conference on parallel architectures and compilation techniques, pp 240–249
3. Cebrian JM, Aragon JL, Garcia JM, Petoumenos P, Kaxiras S (2009) Efficient microarchitecture policies for accurately adapting to power constraints. In: Proceedings of the IEEE international parallel and distributed processing, symposium, pp 1–12. doi:10.1109/IPDPS.2009.5161022
4. Cebrian JM, Aragon JL, Kaxiras S (2011) Power token balancing: adapting CMPS to power constraints for parallel multithreaded workloads. In: Proceedings of the IEEE international parallel and distributed processing symposium
5. Donald J, Martonosi M (2006) Techniques for multicore thermal management: Classification and new exploration. In: Proceedings of the 33rd international symposium on computer, architecture, pp 78–88. doi:10.1109/ISCA.2006.39
6. Esmaeilzadeh H, Blem E, St. Amant R, Sankaralingam K, Burger D (2011) Dark silicon and the end of multicore scaling. In: Proceedings of the 38th annual international symposium on Computer architecture, ISCA '11, pp 365–376. ACM, New York, NY, USA. doi:10.1145/2000064.2000108. http://doi.acm.org/10.1145/2000064.2000108
7. Flynn MJ, Hung P (2005) Microprocessor design issues: thoughts on the road ahead 25(3):16–31. doi:10.1109/MM.2005.56
8. Isci C, Buyuktosunoglu A, Cher CY, Bose P, Martonosi M (2006) An analysis of efficient multi-core global power management policies: maximizing performance for a given power budget. In: Proceedings of the 39th annual IEEE/ACM international symposium on microarchitecture, pp 347–358. doi:10.1109/MICRO.2006.8
9. Keshavarzi, A. (1997) Intrinsic iddq: origins, reduction, and applications in deep sub- low-power cmos ic's. In: Proceedings of the IEEE international test conference

10. Kim NS, Austin T, Baauw D, Mudge T, Flautner K, Hu JS, Irwin MJ, Kandemir M, Narayanan V (2003) Leakage current: Moore's law meets static power. Computer 36(12):68–75. doi:10.1109/MC.2003.1250885
11. Kim W, Gupta MS, Wei GY, Brooks D (2008) System level analysis of fast, per-core DVFS using on-chip switching regulators. In: Proceedings of the IEEE 14th international symposium on high performance computer, architecture, pp 123–134. doi:10.1109/HPCA.2008.4658633
12. Li J, Martinez JF, Huang MC (2004) The thrifty barrier: energy-aware synchronization in shared-memory multiprocessors. In: Proceedings of the 10th international symposium on high performance computer, architecture, pp 14–23. doi:10.1109/HPCA.2004.10018
13. Li S, Ahn JH, Strong RD, Brockman JB, Tullsen DM, Jouppi NP (2009) Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In: Proceedings of the 42th international symposium on microarchitecture, pp 469–480
14. Li T, Lebeck AR, Sorin DJ (2006) Spin detection hardware for improved management of multithreaded systems 17(6):508–521. doi:10.1109/TPDS.2006.78
15. Macken P, Degrauwe M, Van Paemel M, Oguey H (1990) A voltage reduction technique for digital systems. In: Proceedings of the 37th IEEE international solid-state circuits conference, digest of technical papers, pp 238–239. doi:10.1109/ISSCC.1990.110213
16. Magnusson PS, Christensson M, Eskilson J, Forsgren D, Hallberg G, Hogberg J, Larsson F, Moestedt A, Werner B (2002) Simics: a full system simulation platform. Computer 35(2):50–58. doi:10.1109/2.982916
17. Martin MMK, Sorin DJ, Beckmann BM, Marty MR, Xu M, Alameldeen AR, Moore KE, Hill MD, Wood DA (2005) Multifacet's general execution-driven multiprocessor simulator (gems) toolset. SIGARCH Comput Archit News 33:2005
18. Meng K, Joseph R, Dick RP, Shang L (2008) Multi-optimization power management for chip multiprocessors. In: Proceedings of the 17th international conference on parallel architectures and compilation techniques, PACT '08, pp 177–186. ACM, New York, NY, USA. http://doi.acm.org/10.1145/1454115.1454141
19. Sartori J, Kumar R (2009) Distributed peak power management for many-core architectures. In: Proceedings of the design, automation and test in Europe conference and Exhibition, pp 1556–1559
20. Sasanka R, Hughes CJ, Adve SV (2002) Joint local and global hardware adaptations for energy. In: Proceedings of the 10th international conference on architectural support for programming languages and operating systems, ASPLOS-X, pp 144–155. ACM, New York, NY, USA. http://doi.acm.org/10.1145/605397.605413
21. Semeraro G, Magklis G, Balasubramonian R, Albonesi DH, Dwarkadas S, Scott ML (2002) Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling. In: Proceedings of the 8th international high-performance computer architecture, symposium, pp 29–40 doi:10.1109/HPCA.2002.995696
22. Simunic T, Benini L, Acquaviva A, Glynn P, de Micheli G (2001) Dynamic voltage scaling and power management for portable systems. In: Proceedings on design automation conference, pp 524–529. doi:10.1109/DAC.2001.156195
23. Skadron K, Stan MR, Huang W, Velusamy S, Sankaranarayanan K, Tarjan D (2003) Temperature-aware microarchitecture. In: Proceedings of the 30th annual international computer architecture, symposium, pp 2–13. doi:10.1109/ISCA.2003.1206984
24. Winter JA, Albonesi DH (2008) Addressing thermal nonuniformity in smt workloads. ACM Trans Archit Code Optim 5:4:1–4:28. http://doi.acm.org/10.1145/1369396.1369400
25. Wu Q, Juang P, Martonosi M, Clark DW (2005) Voltage and frequency control with adaptive reaction time in multiple-clock-domain processors. In: Proceedings of the 11th international symposium on high-performance computer, architecture, pp 178–189. doi:10.1109/HPCA.2005.43