Power Token Balancing: Adapting CMPs to Power Constraints for Parallel Multithreaded Workloads

Juan M. Cebrián, Juan L. Aragón
Dept. of Computer Engineering,
University of Murcia, Murcia, Spain
{jcebrian,jlaragon}@ditec.um.es

Stefanos Kaxiras
Dept. of Information Technology,
University of Uppsala, Uppsala, Sweden
stefanos.kaxiras@it.uu.se

***Abstract*—In the recent years virtually all processor architectures employ multiple cores per chip (CMPs). It is possible to use legacy (i.e., single-core) power saving techniques in CMPs which run either sequential applications or independent multithreaded workloads. However, new challenges arise when running parallel shared-memory applications. In the later case, sacrificing some performance in a single core (thread) in order to be more energy-efficient might unintentionally delay the rest of cores (threads) due to synchronization points (locks/barriers), therefore, harming the performance of the whole application.**

CMPs increasingly face thermal and power-related problems during their typical use. Such problems can be solved by setting a power budget to the processor/core. This paper initially studies the behavior of different techniques to match a predefined power budget in a CMP processor. While legacy techniques properly work for thread independent/multi-programmed workloads, parallel workloads exhibit the problem of independently adapting the power of each core in a thread dependent scenario. In order to solve this problem we propose a novel mechanism, *Power Token Balancing* (PTB), aimed at accurately matching an external power constraint by balancing the power consumed among the different cores using a power token-based approach while optimizing the energy efficiency. We can use power (seen as tokens or coupons) from non-critical threads for the benefit of critical threads. PTB runs transparent for thread independent / multiprogrammed workloads and can be also used as a spinlock detector based on power patterns. Results show that PTB matches more accurately a predefined power budget (total energy consumed over the budget is reduced to 8% for a 16-core CMP) than DVFS with only a 3% energy increase. Finally, we can trade accuracy on matching the power budget for energy-efficiency reducing the energy a 4% with a 20% of accuracy.

Keywords: power consumption; power budget; DVFS; power tokens; power balancing.

I. Introduction

CMP architectures exhibit some peculiarities in terms of power and performance compared to a single-core processor. In terms of power consumption, whenever the number of processing cores is doubled, power consumption is almost multiplied by two. Although technology scaling reduces dynamic power consumption, the complexity of the interconnection network and caches increases when more cores are incorporated into the die resulting in higher power consumption. Furthermore, when a CMP is running parallel multithreaded workloads many interdependent threads are usually executed in different cores. In this scenario, if a power-saving mechanism is independently applied to a single core, it can affect the rest of threads in the next synchronization point, slowing down the whole program execution and increasing the overall energy consumption. Global information is required to reduce power in threads that are not in the critical path of execution (e.g., threads that arrive earlier to synchronization points) [1][11][12][13][17].

The more cores in the processor, the more thermal problems. A relevant technique to reduce temperature is to set a power budget to the processor [1][2]. This processor's power budget is not only useful to control power and temperature but also to satisfy external power constraints (i.e., power cuts or shared power supply). It could also be used to increase the number of cores in a CMP maintaining the same TDP (Thermal Design Power), or to reuse an existent processor design with a cheaper thermal package. A well-known approach to make the processor's power converge to a power budget is *Dynamic Voltage and Frequency Scaling* (DVFS) [5][6][7]. DVFS relies on modifications in voltage and frequency to reduce the processor's dynamic power, as dynamic power depends on both voltage (quadratically) and frequency (linearly). However, DVFS has some important drawbacks: a) long transition times between power modes [8]; b) long exploration and use windows in order to amortize DVFS overheads, making it difficult to adapt precisely to the program behaviour; and c) when activated DVFS affects all instructions within a thread regardless of their usefulness in the program.

In order to overcome those DVFS limitations, in [2] Cebrián *et al.* proposed the use of fine-grained microarchitectural power-saving techniques to accurately match a predefined power budget in a single-core scenario. However, in a CMP scenario these techniques have a great impact on power and performance due to synchronization points which make them not good candidates to lower the power under the budget. In the literature it can be found other specific CMP proposals to match a predefined power budget such as [1][18] or [19], but they are only suitable for CMPs running multiple single-threaded (or multiprogrammed) applications but not parallel workloads and they do not perform any accuracy analysis on the budget requirements. There are other works aimed at reducing the power wasted when cores wait at synchronization points, either putting cores to sleep [12] or trying to make all cores reach the synchronization point simultaneously (e.g., *meeting points* [11] or *thrifty barriers* [13]), however, these mechanisms are not suitable for matching a power budget on their own, the main goal of this work.

To address the shortcomings of previous works we propose *Power Token Balancing* (PTB), a mechanism that balances the CMP power consumption by means of efficiently distributing power tokens among the cores. Whenever the CMP exceeds a predefined global power budget, local power budgets are applied to all running cores. Without any global mechanism the power would be just equally split between the cores. However, PTB globally manages power consumption so cores that are under the 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 ensuring that the global power budget is not exceeded. PTB benefits from any power unbalance among cores (cache misses, ROB stalls, pipeline stalls, etc.) but has another important feature: it transparently benefits from thread's busy-waiting synchronization in a very "lightweight" way. Since a core waiting in a barrier naturally reduces its power consumption, PTB allows its spare power tokens to be given to other cores doing useful work (i.e., critical threads). The same applies to locks: a core that enters into a critical section receives extra power tokens from other cores waiting on spinlocks. Due to the additional power tokens its local power budget is less restrictive and the core can leave the critical section faster.

The main advantages of PTB over previous approaches focused on low-power spinning (such as *meeting points* [11], *thrifty barriers* [13] or [19]) are:

- PTB is designed (but not limited) to work in a CMP scenario running parallel workloads. Unlike other proposals, PTB is not limited to multiple instances of applications and also works properly with multiprogrammed workloads.
- PTB's main goal is to make the per-cycle power consumption go under a certain power budget while maximizing accuracy, not to reduce overall energy. Previous approaches on low-power spinning are not suitable for a power-constrained scenario, as power saving mechanisms must be applied outside the synchronization points.
- PTB takes advantage of any power unbalance including both barriers/locks in a spinning state (busy-wait) plus thread criticality by just relying on power token information, making it more generic than previous approaches.
- PTB can identify critical threads faster than [11][12][13] (the critical thread can change during execution) since it relies on cycle-level information, increasing its adaptability to the application behavior.
- PTB is a fine-grained approach, unlike [11][12][13][19], because it relies on actual real-time information and not time/power estimations, increasing accuracy when matching the budget and minimizing the standard power deviation from the selected power budget.
- PTB is able to reduce both the average power consumption and the average chip temperature with minimal standard deviation due to its precision on matching the predefined power budget.

The rest of the paper is organized as follows. Section II provides some background on power-saving techniques for both single-core processors and CMPs. Section III 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 IV reports the main experimental results. Finally, Section V shows our concluding remarks.

II. Background and Related Work

A. *Dynamic Voltage Frequency Scaling (DVFS)*

DVFS, introduced in the 90's [6], is based on the fact that the per-cycle power consumption of a transistor depends quadratically on the supply voltage and linearly on its frequency ($P \propto V_{DD}^2 \times f$) and downscales both voltage and frequency to save power [7]. But as the process technology goes into deep submicron, the margin between V_{DD} (supply voltage) and V_T (threshold voltage) is reduced. As this margin decreases, the processor's reliability is reduced, among other undesirable effects.

Furthermore, the transistor's delay (or switching speed) depends on $\delta \approx 1/(V_{DD} - V_T)^{\alpha}$, with $\alpha > 1$. That means that we can lower V_{DD} for DVFS as long as we keep constant the margin between V_{DD} and V_T (i.e., V_T must be lowered accordingly) so we can obtain the desired speed increase derived from technology scaling. However, the counterpart of reducing V_T is twofold: a) leakage power increases as it exponentially depends on V_T [3][4]; and b) processor reliability is further reduced.

B. *Hybrid Power Control Approaches*

Recently, in [2] we introduced the concept of power tokens in a single-core scenario along with a two-level approach that firstly applies DVFS as a coarse-grained approach to reduce power consumption towards a predefined power budget, and secondly chooses 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 on matching the power budget for a single-core scenario. Sasanka *et al.* propose the use of DVFS and some micro-architectural techniques to specifically reduce the power consumption in real time video programs [9]. Their selected microarchitectural techniques try to reduce the power of functional units and the instruction window by using profiling. Winter *et al.* also propose the use of a two-level approach that merges DVFS and thread migration to reduce temperature in SMT processors [10].

C. CMP-specific Power Control Mechanisms

As mentioned before, CMP processors have some peculiarities when managing power and performance in parallel workloads. 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.

1) Spin-lock Detection to Reduce Power

When a processor is waiting in either a lock or a barrier it enters in a "spinning" state that may become an important source of useless power consumption if it remains there for too long. In order to detect spinning, first approaches used source code or binary instrumentation but that requires recompilation and might be infeasible in certain situations. In [12] Li *et al.* 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 in 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) DVFS in CMPs

As cited in the introduction, Li *et al.* proposed *thrifty barriers* [13], a DVFS-based mechanism to reduce power consumption in CMPs. Basically, they calculate the per-core time interval between synchronization points and once they know how long it takes the different cores to get to the next synchronization point, they disable or DVFS cores that get to the barrier. They approximate the wakeup time by the time the slowest thread takes to get to the barrier. If the sleep/wakeup takes more time than they can save, the technique is not used. In [1] Isci *et al.* proposed a chip-level dynamic power management for CMPs but just focusing on single-threaded programs while Sartori *et al.* [19] extends this work to reduce peak power in a distributed way. These mechanisms selectively change between several DVFS power modes for the different cores maximizing throughput under certain power constraints. Unfortunately, as they rely on the use of performance counters and/or time estimation, these proposals only work properly for multiprogrammed or single-threaded applications simultaneously running on the different cores of the CMP, because for parallel workloads performance counters (and time estimations) are almost useless for relating performance and power (due to synchronization points). A spinning core may have a high IPC, but doing nothing but spinning. In other words, synchronization points may increase global execution time although local core performance counters show a performance increase. Moreover, none of the mentioned techniques provides any precision on matching an imposed power budget.

In [11] Cai *et al.* proposed *meeting points*. This mechanism locates critical threads in parallel regions and uses DVFS to reduce power consumption of non-critical threads. They propose two approaches: thread delaying, that slows down the fastest thread to ensure that all threads get to the synchronization point (meeting point) at the same time; and thread balancing, that gives priority to the critical thread when accessing resources in a 2-way SMT processor. 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 on producing 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. Again, 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.

III. Enforcing a Power Budget in CMPs

There are certain circumstances, usually related to power or thermal constraints, where precise power/energy control of the processor is required. Our goal is to reduce power consumption to accurately match an imposed power budget in an energy-efficient way, always having in mind the peculiarities of CMP processors running parallel workloads. To achieve this goal we need: first to 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, to balance the power between the cores; and finally (when nothing else can be done), to reduce the power locally even at the cost of degrading performance (by means of DVFS and/or microarchitectural techniques).

A. Simulation Environment

For evaluating the proposed approaches we have used the Virtutech Simics platform [14] extended with Wisconsin GEMS v2.1 [15]. 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 next. 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 32nm technology were obtained from Cacti's v5.1 [16]. To evaluate the performance and power consumption 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 less than 3 days in our cluster). Results have been extracted from the parallel phase of each benchmark. Benchmark sizes are specified in Table 2.

As simulation results we will provide both overall CMP energy consumption along with the *accuracy* of each evaluated technique on matching a predefined global power budget. 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 the power budget and each core dynamic power curve (represented by shadowed areas in Figure 1). The lower the area (energy) the more accurate the technique (the ideal AoPB is zero). Performance results are only shown for the dynamic approach (see section IV.B) due to space limitations and because performance is implicitly accounted in the reported energy results.

B. Measuring Power in Real-time

The concept of *Power-Tokens* (introduced in [2]) basically consists of calculating the dynamic power consumed by an instruction at commit stage by adding, to the base power 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 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 [2], the implementation of the *Power-Token* approach is done by means of an 8K-entry history table (Power-Token History Table – PTHT), accessed by PC, which stores the power cost (in tokens) of each instruction's last execution. 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 by running the SPECint2000 benchmark suite with the processor configuration shown in Table 1. Once we had the base power for all the possible instructions, we used a K-mean algorithm to group instructions with similar base power consumption. Our simulated results show 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 accounting for the actual power consumption in joules as provided by HotLeakage).

Hence, the overall processor power consumption 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 power-tokens (provided by the PTHT) of each instruction being fetched. Note that the extra power consumption of the PTHT structure is also accounted in our results.

C. Matching a Power Budget in a CMP Running Parallel Workloads

This section discusses different approaches for managing power consumption under power constraints. Initially, we will adapt and tune the best proposed techniques in [2] to a

Table 1. Simulated CMP configuration.

Processor Core	
Process Technology:	32 nanometres
Frequency:	3000 MHz
V_{DD}:	0.9 V
Instruction Window:	128 entries + 64 Load Store Queue
Decode Width:	4 inst/cycle
Issue Width:	4 inst/cycle
Functional Units:	6 Int Alu; 2 Int Mult 4 FP Alu; 4 FP Mult
Pipeline:	14 stages
Branch Predictor:	64KB, 16 bit Gshare
Memory Hierarchy	
Coherence Protocol	MOESI
Memory Latency	300 Cycles
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 cycles latency
Network Parameters	
Topology	2D mesh
Link Latency	4 cycles
Flit size:	4 bytes
Link Bandwidth:	1 flit / cycle

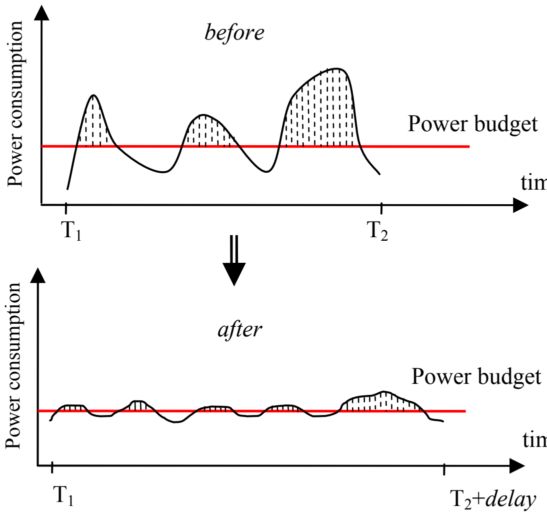
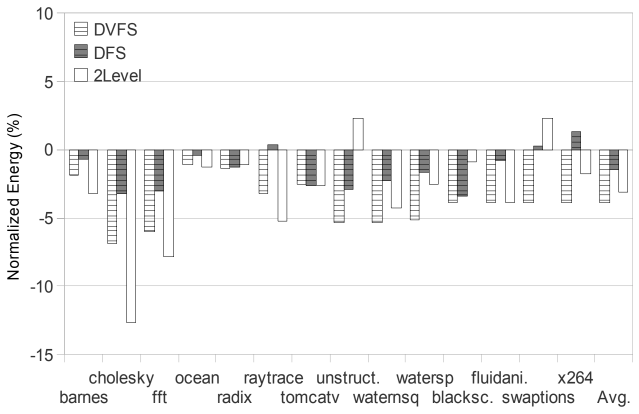


Figure 1- Area over the Power Budget (AoPB) metric example – shadowed areas.

Table 2. Evaluated benchmarks and input 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	258x258 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



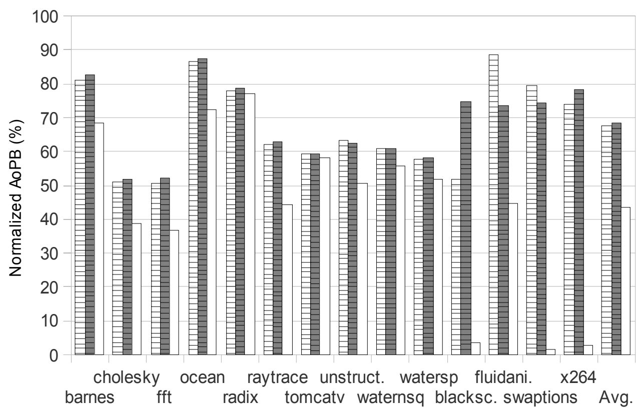


Figure 2. Normalized Energy (left) and AoPB (right) for a 16-core CMP with a power budget of 50%.

CMP scenario. These evaluated techniques are:

a) 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).

b) DFS: similar to a) but only scaling down frequency (V_{DD} remains 100% in all cases).

c) Hybrid: as in [2], a 2-level approach that uses DVFS to lower the average power consumption towards the power budget and then uses different microarchitectural techniques to remove power spikes (2level in the graphs).

Note that techniques from [2] were designed for the single-core scenario and, therefore, they are applied at the core-level instead of at the CMP-level, so the first step is to decide how to split the power available for the whole CMP (as determined by the global power budget) among the different cores. A naive and straightforward initial implementation consists of assigning to each core the same amount of available power. In this case, power budget techniques will be locally applied to a particular core if: 1) the whole CMP is over the global power budget (Σ core$_i$ power > global power budget); and 2) a particular core is over its local power budget (core$_i$ power > global power budget/number of cores).

Initially, we will show simulation results when applying the above power matching techniques (DVFS, DFS, 2level) to a 16-core CMP (results for 2, 4 and 8 cores have been omitted due to space limitations) 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 and using clock gating. It is important to note that we have selected Kim's implementation [8] 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 in the 2-level experiment) will become even more accurate and energy-efficient than DVFS.

Figure 2 shows normalized energy and area over the power budget (AoPB) with respect to a base case where no power-control mechanisms are used to meet the global power budget. In terms of energy, all the evaluated techniques behave accordingly with the numbers we reported in [2] for the single-core scenario. In some benchmarks, like Cholesky, the 2-level approach reduces energy by almost 13%. In terms of performance, the average degradation is under 1% for the studied benchmarks. 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, around 45%, which is far from the average 10% AoPB we obtained for the single-core scenario [2]. 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 well respected.

What is happening in the CMP scenario? The main difference are synchronization points found in parallel workloads and thus, the optimal execution configuration for each individual core may not be the optimal configuration for the whole CMP. Applications with low AoPB have no lock/barrier contention as we will see in Figure 3.

This initial analysis shows that previous mechanisms for managing power under temporary power constraints are not suitable for a CMP scenario when using this naive distribution policy that equally splits the power among cores.

D. *Analysis on the Power Consumed in Spinning*

Differently, our major goal is now to accurately match an imposed power budget in an energy-efficient way while having in mind the peculiarities of CMP processors running parallel workloads. In this case, it is important to focus on places where power can be saved without harming performance such as synchronization points.

Figure 3 shows a first analysis on the time spent by a CMP with a varying number of cores (from 2 to 16) either spinning or 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

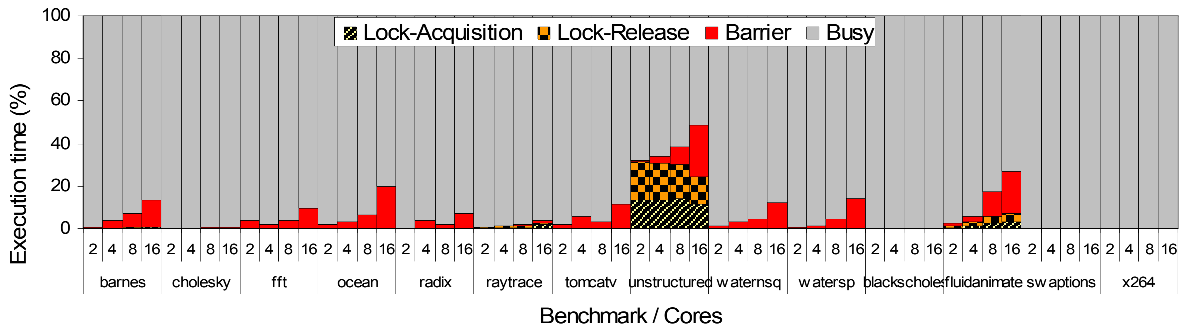


Figure 3. Execution time breakdown for a varying number of cores.

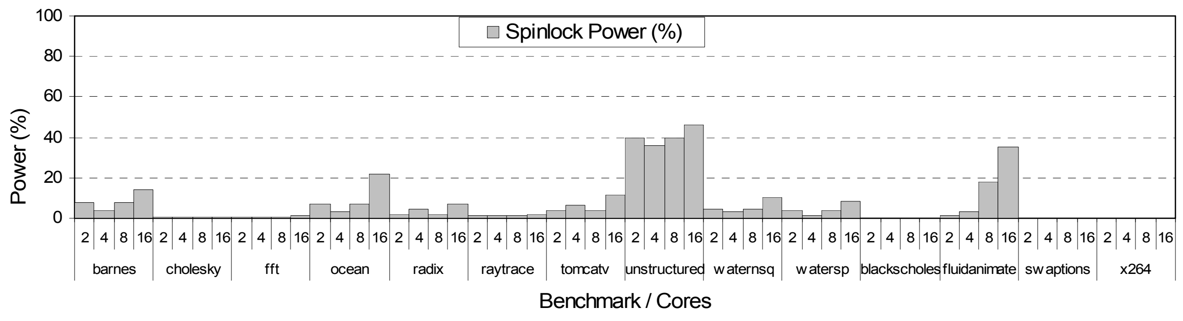


Figure 4. Normalized spinlock power for a varying number of cores.

Lock-Rel states (contended locks) while others (Cholesky/ Blackscholes/Swaptions/x264), in contrast, have no lock/ barrier contention. While Cholesly's behavior is due to a well balanced code, the other three benchmarks only synchronize at the end of the code.

More interestingly, Figure 4 shows the power wasted while spinning normalized to the total power consumed by the original processor. As explained in Section II.C, this wasted power can be reduced by detecting spinning and slowing down/stalling the cores. Specifically, improving the previously proposed spin-detection techniques is out of the scope of this paper, however, the use of *Power-Tokens* can indirectly be used to detect spinning states as we will describe in the next section. This spinlock power is close to a 10% on average for a 16-core processor running all the studied benchmarks. In any case, this 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 the whole time it lasts. Therefore, we need a more generic approach that could benefit from other wasteful situations such as mispredictions events.

E. Power Token Balancing (PTB)

1) PTB Basics

If we take a look at the power consumed by each individual core at a cycle level, even if the total power consumed by the CMP is over the global power budget, there may be cores under their power budget share. For example, assuming a 4-core CMP and global power budget of 40W, the naive implementation cited before assigns to each core a local power budget of 10W. Looking at Figure 5, cycles 1, 2 and 4 are over the global power budget (40W), so local power-savings should be enabled. In cycle 1, no power-control mechanisms are applied to cores 1 and 2 since they are under their local power budget (10W). Differently, cores 3 and 4 apply their local power-saving mechanisms. However, slowing down those cores could be more harmful than it seems since it may cause the whole application to slow down in a future synchronization point if they are critical execution threads. The same happens in cycle 2 with core 3. In cycle 3, however, although there are cores exceeding their local power budget, no mechanism is applied as the global CMP power is under the budget. Finally, in cycle 4, all cores exceed their local power budget (so does the CMP which exceeds the global one), and hence, local mechanisms are applied to all cores. Note however, that in cycles 1 and 2, if we were able to tell cores 3&4 that their respective local power budgets are less restrictive than 10W (since cores 1&2 have some power left, 4+2 W; 2+1 W), the effects on the performance should be less harmful.

The PTB approach is based on *Power-Token* accounting. Each individual core will count, at a cycle level, how many 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, the core offers its spare tokens to the PTB load-balancer. Tokens are used as a currency to account for power, so it is important to note that they are neither sent nor received,

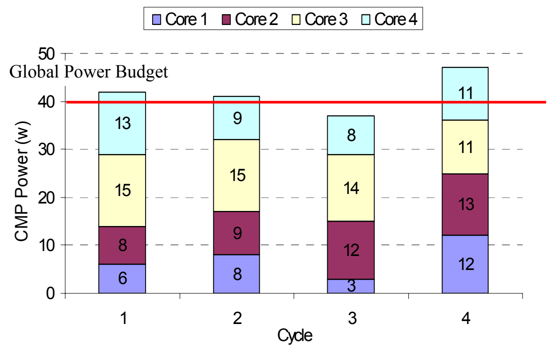


Figure 5. *Power Token Balancing* motivation (not real numbers).

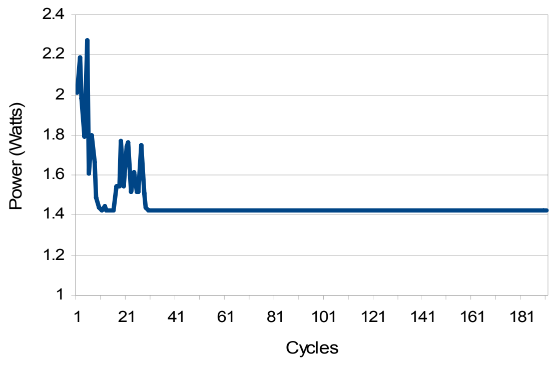


Figure 6. Per-cycle power behavior of a spinning core.

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 that cycle. Therefore, PTB is not a loan/refund mechanism, a core can reuse power from others but there is no need to give it back.

PTB has two power distribution policies: 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). PTB also exhibits two inherent features that allow "transparent" optimizations without any specific mechanism: 1) indirect spinning detection, and 2) an automatic priority system for non-spinning threads.

Figures 6 help us to illustrate how PTB could be used to detect spinning. When a core enters a spinning state, the consumed power follows the behavior shown in Figure 6. After the initial power peak due to useful computation, if the spinning state lasts enough, power lowers and stabilizes (cycle >35 in Figure 6) to an amount that is usually under the budget that presumably means the core is spinning. Note, however, that spinning is just a particular case of power unbalance, so our mechanism will benefit from spinning but that is not the only case. Actually, PTB knows nothing about locks, barriers, mispredictions, etc, it just balances power.

For illustrating purposes, and continuing with the spinning example, Figure 7 shows how PTB works in the case of a barrier. For this example let us suppose 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 to the PTB load-balancer. Figure 7-a shows that core 2 reaches to the barrier and so it will transfer 6 tokens to the load-balancer. Now the rest of cores have more available power 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 Figure 7-b) reaches to the barrier it also gives its 6 spare tokens to the PTB load-balancer which allows cores 1&4 to use the 6+6 extra tokens from cores 2&3, raising their local budget to 16. Finally, Figure 7-c also shows core 1 spinning in the barrier and giving its 6 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. As PTB basically detects power unbalance among cores it will benefit from any misprediction event (e.g., a cache miss or a mispredicted branch), not only from spinning states.

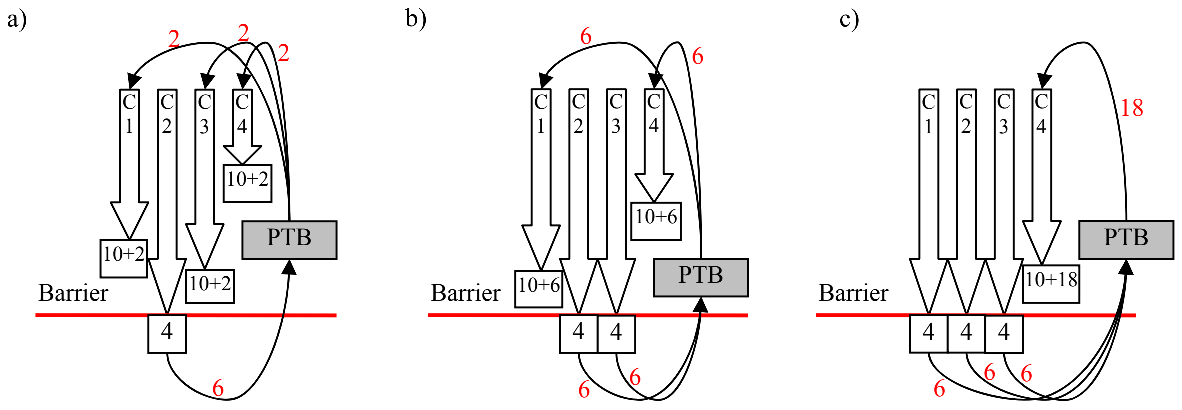


Figure 7. *Power Token Balancing* example in the case of a barrier.

2) *PTB Implementation*

To implement the *Power Token Balancing* mechanism we need a centralized structure, the PTB load-balancer, that receives the number of spare power tokens from all cores *under* their local power budget and splits them among the cores *exceeding* it (with the aim of not triggering any power-saving mechanism for the exceeding cores which would result in a performance degradation).

Balancing is done every cycle, so tokens from previous cycles are not stored in the balancer. We need to build the interconnection wires to send the number of tokens from/to the cores to/from the PTB load-balancer as depicted in Figure 8. We will use 4 wires for sending and 4 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, not the tokens themselves (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.

We have estimated the delays using Xilinx ISE for a processor running at 3GHz without buffers as a reference to calculate the logic delay of the circuit and eliminated the delay caused by the pins and router delay, 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 tokens, one for processing tokens and one for sending 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 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 power until tokens reach its destination. The power consumption of the PTB mechanism plus the interconnection wires has been estimated using Xilinx XPower Analyzer with the same configuration as the delay latency, increasing the average application power 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 interconnection and processing latencies. However, for the analyzed number of cores and latencies the experimental results show significant improvements in terms of accuracy on matching the power budget, temperature and energy, even with a pessimistic 10-cycle delay for sending/receiving tokens from other cores. Nevertheless, one approach to make PTB more scalable (>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.

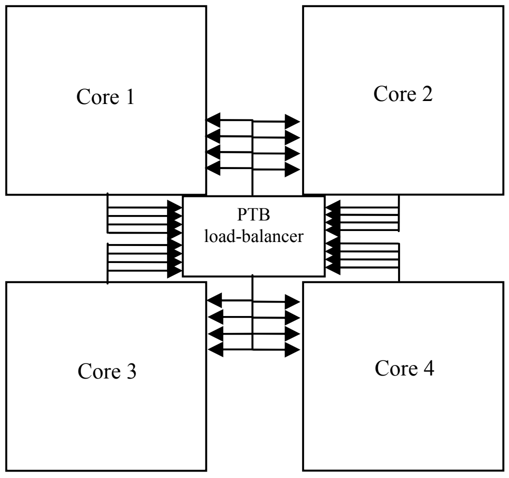


Figure 8. PTB implementation diagram for a 4-core CMP.

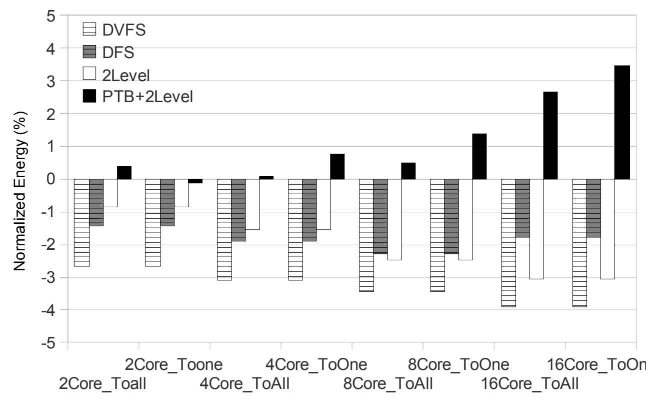
IV. EXPERIMENTAL RESULTS

A. *Efficiency of Power Token Balancing (PTB)*

This section reports simulation results for a global power budget of 50% of the peak power consumption[1] with clock gating and a varying number of processing cores in the CMP (2 to 16 cores). We evaluate the proposed 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). All results are normalized to a base case where no power-control mechanisms are used to match the global power budget. Figure 9 (left) shows the energy consumption for different pairs of {core number/policy} for the evaluated techniques (enumerated in Section III.C) whereas Figure 9 (right) shows the AoPB metric. It can be observed that, when using the proposed PTB mechanism, area numbers go back to the reported numbers in our previous work [2] for the single-core scenario: average 10% of AoPB for a 16-core CMP when the PTB+2level technique is used (although energy numbers are not as good as in the single-core scenario). In a 16-core CMP, DVFS and DFS are unable to lower the AoPB below 65% while PTB+2level reduces the average area to just 8%, getting close to the ideal AoPB of zero, with only 3% more energy consumed. It can also be observed that accuracy on matching the power budget increases (AoPB decreases) with the number of cores, because we have more chances of receiving tokens from other cores.

A more detailed analysis (Figures 10 and 11) shows that there are benchmarks, like Unstructured, where energy increases when using power saving techniques (mainly due to sync points - see Figures 3&4). Unstructured has many thread dependences and slowing down a core causes a great impact on performance. On the other hand, benchmarks like

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



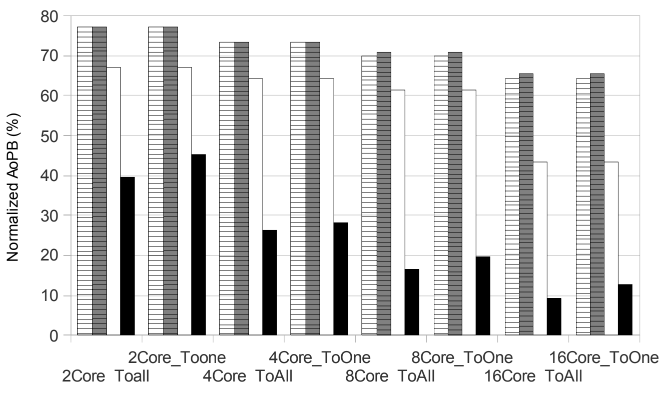
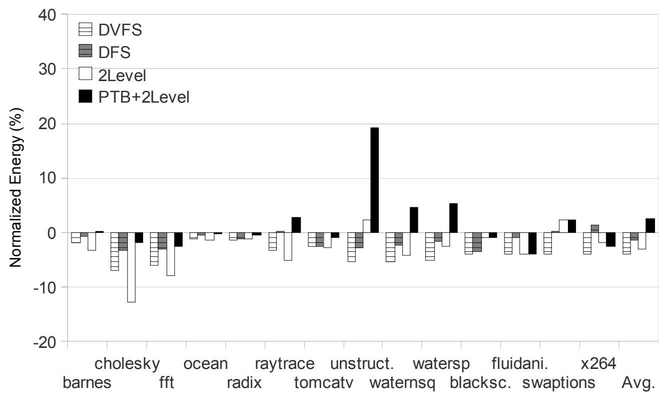


Figure 9. Normalized energy (left) and area over the power budget (right) for a varying number of cores and PTB policies.



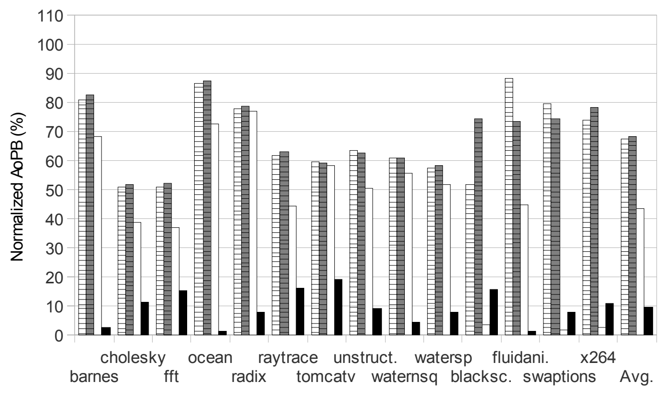
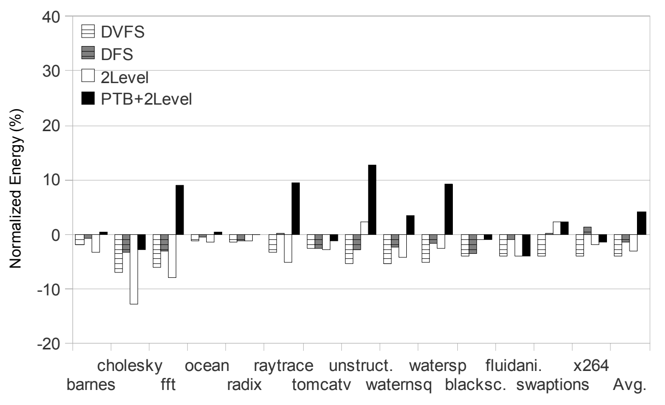


Figure 10. Detailed energy (left) and AoPB (right) for a 16-core CMP with the *ToAll* PTB policy.



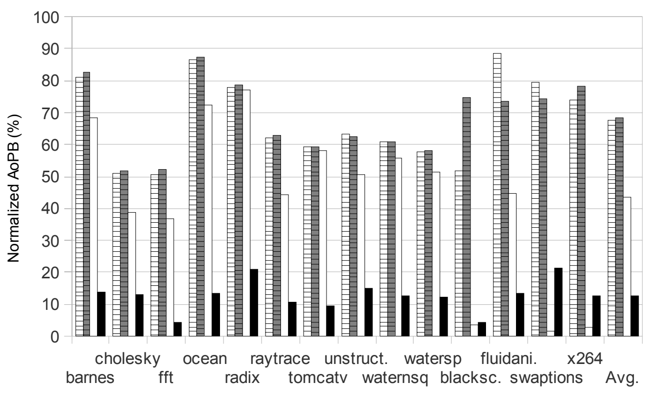
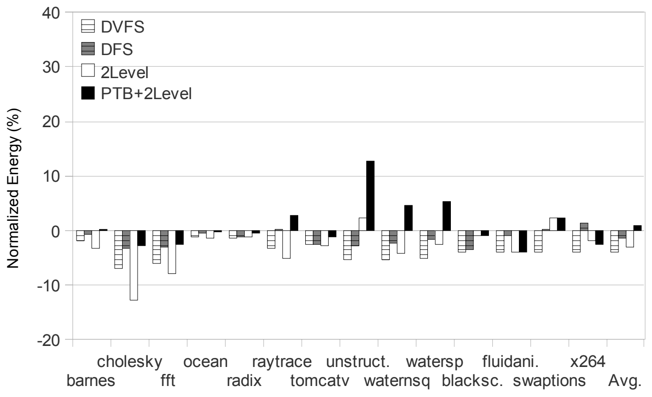


Figure 11. Detailed energy (left) and AoPB (right) for a 16-core CMP with the *ToOne* PTB policy.

Barnes and Ocean, that reported very high AoPB of 70% for the naive power-distribution implementation discussed in Section III.C (Figure 2), now offer an AoPB of just 2% thanks to the efficient power distribution among cores performed by PTB. However, in some benchmarks the extra accuracy on matching the global power budget comes at the cost of higher energy consumption than that of DVFS alone. 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 section IV.C.

When comparing both the *ToAll* and *ToOne* power-token distribution policies, on average, the former works better than the later. However, benchmarks like Unstructured and Waternsq work better when the extra power is given to a single core rather than to all cores. In these benchmarks, threads have an unbalanced workload and spend a significant



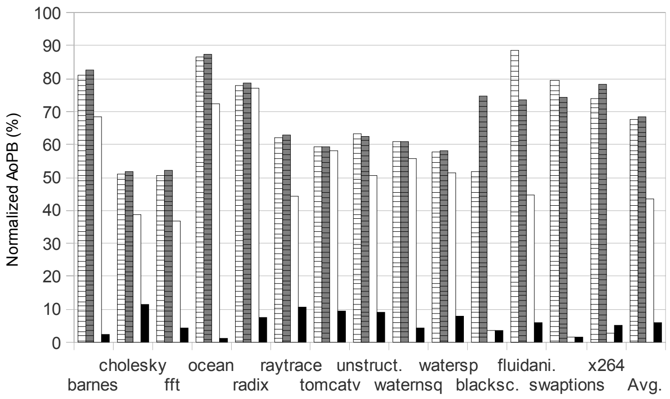


Figure 12. Detailed energy (left) and AoPB (right) for a 16-core CMP using the dynamic policy selector.

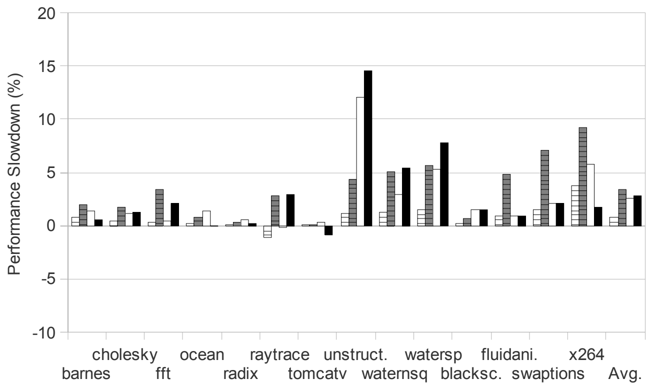


Figure 13. Detailed Performance for a 16-core CMP using the dynamic policy selector.

fraction of their time spinning on locks. They benefit, therefore, from giving the extra power (priority) to threads that enter in a critical section (i.e., the *ToOne* policy).

B. Dynamic Policy Selector

In a barrier scenario the *ToAll* policy will split power tokens from cores already waiting in the barrier among the remaining cores, speeding them all in order to get to the barrier as soon as possible whereas the *ToOne* policy will only benefit one core, that will get faster to the barrier, but we still have to wait for the rest of cores. On the other hand, when a core is spinning in a lock and gets access to a critical section, giving all the tokens to this core will benefit the overall program execution.

Results in the previous section showed that the *ToAll* policy is best suited for applications with many barriers in the code whereas the *ToOne* policy works better for applications with high lock contention. Therefore, in order to enhance the PTB power balancing mechanism we have included a dynamic selector for the power sharing policy (either *ToOne* or *ToAll*). This selector will change the policy depending on the current state of the spinning cores. If the spinning is taking place to access a lock, the mechanism will use the *ToOne* policy. If the spinning is taking place in a barrier (or there is no spinning) the PTB mechanism will use the *ToAll* policy.

Figure 12 shows how this dynamic policy selection approach obtains the best results for the evaluated techniques in terms of both area and energy metrics and making PTB really close to DVFS (around 2%) in terms of performance, as shown in Figure 13, but with the added benefit of PTB being far more accurate on matching the imposed power budget than DVFS or DFS approaches. Normalized energy goes down to 2%, 1% less than the static *ToAll* and 3% less than the static *ToOne*. Accuracy is improved in 3% compared with the static *ToAll* and 5% compared with the static *ToOne* policy. In terms of performance, Unstructured is the application that is more affected by the micro-architectural power-control mechanisms.

Note that the 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 (e.g., monitoring the number of cores that stop spinning simultaneously via their power token consumption, run-time instruction analysis or techniques similar to those described in [12]). For the sake of clarity, we only report on the first approach which does not entail any additional energy cost for the classification of spinning to barrier- or lock-spinning.

C. Relaxing PTB to be More Energy-Efficient

Our initially proposed PTB mechanism is optimized for accuracy on matching the power budget. As explained before, this kind of focus hurts performance and, therefore, slightly increases overall energy consumption. However, if we relax the accuracy constraint, PTB could also achieve energy savings since power-saving mechanisms would be applied in a less restrictive way, therefore, not affecting performance that much. In order to analyze this new focus, Figure 14 shows how PTB behaves when optimizing for energy-efficiency instead of just accuracy for several relaxed area thresholds (i.e., +10%, +20%, +30%, etc). These relaxed thresholds are used to delay triggering a power-saving mechanisms when the global/local power budgets are exceeded. Note that the original PTB triggers the power-

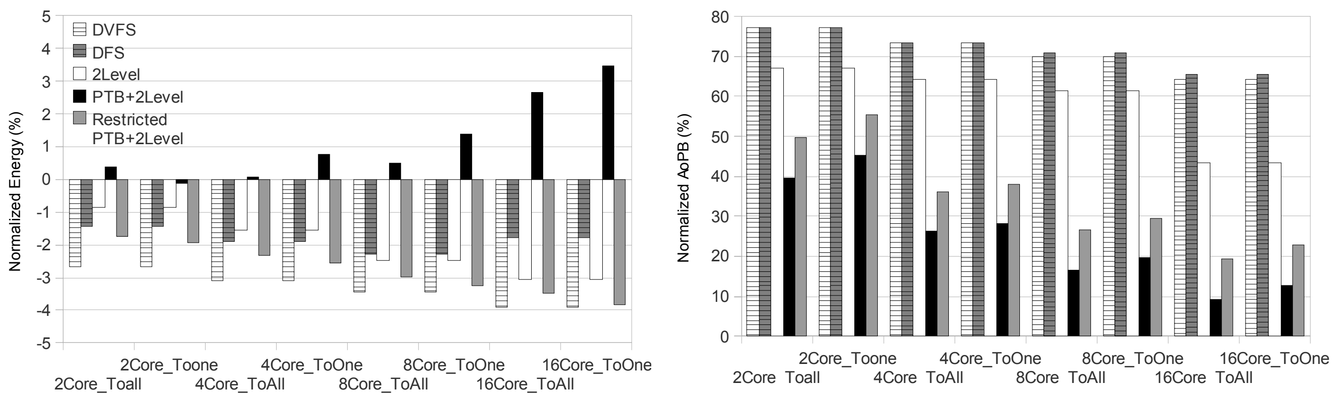


Figure 14. Normalized energy (left) and area over the power budget (right) for a varying number of cores and PTB policies.

saving mechanisms immediately after detecting the current power budget is exceeded.

It can be seen that, for a 16-core CMP with a relaxed AoPB metric allowed to be 20% above the power budget, PTB obtains an average energy reduction of 4% similar to that obtained by DVFS, however, still being more accurate than DVFS that obtains a huge AoPB of 65%. Of course, better energy savings can be achieved for the same 16-core CMP if we relax more the area constraint. Furthermore, higher energy savings could be achieved if we use PTB as a spinlock detector and we disable the spinning cores to save power. But the later is out of the scope of the current paper and part of our future work.

D. *The Importance of Accuracy*

Finally, we will provide an example that tries to illustrate the importance of the accuracy on matching a predefined power budget, which is closely related to what authors try to achieve in [19], where they increase the number of cores of a CMP maintaining the same TDP (thermal design power).

Let us suppose that we want to increase the number of cores in a CMP maintaining the same TDP. For a 16-core CMP with a 100W TDP there are 6.25W per core (for simplicity let us ignore the interconnection network). If we set a power budget of 50% we could ideally duplicate the number of cores in that CMP with the same TDP (up to 32 cores each one consuming an average of 3.125W). But for this ideal case it is needed a perfect accuracy on matching the power budget.

According to our previous results, DVFS has an error of 65% energy left over the power budget. Therefore, with a 65% error each core consumption raises to 3.125*1.65=5.15W, and for a 100W TDP we can put a maximum of 100/5.15=19 cores inside the CMP. For a regular 2level approach (without PTB) the error is reduced to 40% that gives us a potential average power consumption of 3.125*1.40=4.375W per core, so we can put 100/4.375=22 cores in the CMP 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 consumption of 3.125*1.1=3.4375W per core, so we can put 100/3.4375=29 cores inside our CMP. If our application is parallel enough to use these cores it can perfectly overcome the 5% performance degradation of using the non-relaxed PTB approach.

V. CONCLUSIONS

Design complexity and verification of microprocessors is increasingly costly. Some companies cannot afford the design of custom processors for their products (especially for cost-sensitive consumer handheld devices and gadgets) and have to rely on existing processors that may not meet their power requirements. In other scenarios it might be useful to increase the number of cores on a CMP maintaining the same thermal envelopment. Furthermore, thermal envelop design in processors cannot be done for the worst case, because production costs are raised. Being able to set a power budget to the processor can be helpful in these cases.

In a CMP running parallel workloads, previously proposed power managing mechanisms fail to accurately adapt to temporary power constraints, due to thread dependences and synchronization points. However, power saved just from spinlocks is not enough to match an aggressive power budget because is too local and too low. A global control mechanism is needed to match these design peculiarities. In this paper we propose *Power Token Balancing* (PTB), a novel mechanism that dynamically balances power among the different cores to ensure that the whole processor accurately matches a predefined and global power budget. Our proposed mechanism accounts for unused power from cores that are under the power budget (translated into power-tokens) and passes that power to cores over the power budget, hence, not having to slow them down to match their local power budget constraint.

PTB is a fine-grained mechanism that ensures maximum accuracy with minimal standard deviation from the power budget, which is crucial if you want to optimize packaging costs or to increase the number of cores in a CMP with the same TDP. However, the accuracy constraint can be relaxed in order for PTB to be more energy-efficient by means of applying the power-saving mechanisms in a less restrictive way, and therefore, not affecting performance that much.

Experimental results have shown 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%) and with a more stable temperature over execution time (due to the increased accuracy when matching the power budget), as opposed to DVFS which fails to match the power budget precisely, resulting in a high AoPB of around 65%. Furthermore, when considering a relaxed PTB approach that allows being 20% above the power budget (still far from the 65% AoPB obtained by DFVS), PTB obtains the same energy reduction as DVFS for the 16-core CMP. Of course, better energy savings can be achieved if the area constraint is relaxed still more.

ACKNOWLEDGEMENTS

This work was supported by the Spanish MEC and MICINN, as well as European Commission FEDER funds, under Grants CSD2006-00046 and TIN2009-14475-C04. The authors would like to thank the anonymous reviewers for their insightful suggestions and comments. We also want to thank Varadan Savulimedu for his support on estimating the power consumption and interconnection delay of the PTB mechanism.

REFERENCES

[1] Isci, C.; Buyuktosunoglu, A.; Cher, C.; Bose, P. and Martonosi, M. An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget. In *Proceedings of the 39th International Symposium on Microarchitecture,* 2006.

[2] Cebrián, J.M.; Aragón, J.L.; García, J.M.; Petoumenos, P.; and Kaxiras, S. Efficient Microarchitecture Policies for Accurately Adapting to Power Constraints. In *Proceedings of the 23rd International Parallel and Distributed Processing Symposium (IPDPS)*, Rome, Italy, 2009.

[3] Kesharvarzi, A. Intrinsic iddq: Origins, reduction, and applications in deep sub-micron low-power CMOSIC's. In *Proceedings of the IEEE International Test Conference,* 1997.

[4] Flynn, M.J. and Hung P. Microprocessor Design Issues: Thoughts on the Road Ahead. In *IEEE Micro*, vol. 25, no. 3, 2005.

[5] Donald, J. and Martonosi, M. Techniques for Multicore Thermal Management: Classification and New Exploration. In *Proceedings of the 33th International Symposium on Computer Architecture* (ISCA-33), 2006.

[6] Macken, P.; Degrauwe, M.; Paemel, V. and Oguey, H. A voltage reduction technique for digital systems. In *Proceedings of the IEEE Int. Solid-State Circuits Conference*, 238–239, February 1990.

[7] Simunic, T.; Benini, L.; Acquaviva, A. and Glynn, P. Dynamic Voltage Scaling and Power Management for Portable Systems. In *Proceedings of the Design Automation Conference*, 2001.

[8] Kim, W.; Gupta, M. S. *et al*. System Level Analysis of Fast, Per-Core DVFS using On-Chip Switching Regulators. In *Proceedings of the 14th International Symposium on High-Perf. Computer Architecture (HPCA)*, 2008.

[9] Sasanka, R.; Hughes, C. J. and Adve, S.V. Joint Local and Global Hardware Adaptations for Energy. In *Proceedings of the 10th International Conference on Arch Support for Programming Languages and Operating Systems (ASPLOS)*, 2002.

[10] Winter, J.A. and Albonesi, D.H. Addressing Thermal Non-Uniformity in SMT Workloads. In *ACM Transactions on Architecture and Code Optimization*, Vol. 5, No. 1, May 2008.

[11] Cai, Q.; González, J.; Rakvic, R.; Magklis, G.; Chaparro, P.; González, A. Meeting Points: Using Thread Criticality to Adapt Multicore Hardware to Parallel Regions. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, 2008.

[12] Li, T.; Lebeck, A.R.; Sorin, D.J. Spin Detection Hardware for Improved Management of Multithreaded Systems. In *IEEE Transactions on Parallel and Distributed Systems*, vol 17, 2006.

[13] Li, J.; Martínez, J.F.; Huang, M.C. The Thrifty Barrier: Energy-Aware Synchronization in Shared-Memory Multiprocessors. In *Proceedings of the 10th International Symposium on High-Performance Computer Architecture,* 2004.

[14] Magnusson, P. S.; Christensson, M.; Eskilson, J.; Forsgren, D.; Hallberg, G.; Hogberg, J.; Larsson, F.; Moestedt, A. Simics: A full system simulation platform. *Computer, 35(2):50–58*, 2002.

[15] Martin, M. M. K.; Sorin, D. J.; Beckmann, B. M.; Marty, M. R.; Xu, M.; Alameldeen, A. R.; Moore, K. E.; Hill, M. D. and Wood, D. A. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset. *SIGARCH Comput. Archit. News, 33(4):92–99*, 2005.

[16] Thoziyoor, S.; Muralimanohar, N.; Ahn, J. H.; Jouppi, N. P. Cacti 5.1. *HP-Labs technical report HPL-2008-20*, 2008.

[17] Bhattacharjee, A. and Martonosi, M. Thread criticality predictors for dynamic performance, power, and resource management in chip multiprocessors. In *Proceedings of the 36th International Symposium on Computer Architecture (ISCA),* 2009.

[18] Meng, K; Joseph, R; Dick, P.D.; Shang, L. Multi-optimization power management for chip multiprocessors. In *Proceedings of the 17th International conference on Parallel Architectures and Compilation Techniques (PACT),* 2008.

[19] Sartori, J; Kumar, R. Distributed Peak Power Management for Many-core Architectures. In *Proceedings of the international conference on Design, Automation and Test in Europe,* 2009.

[20] Skadron, K; Stan, M; Huang, W; Velusamy, S; Sankaranarayanan, K and Tarjan, D. Temperature-aware microarchitecture. *In Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2003.