Efficient Microarchitecture Policies for Accurately Adapting to Power Constraints

Juan M. Cebrián[1], Juan L. Aragón[1], José M. García[1], Pavlos Petoumenos[2] and Stefanos Kaxiras[2]
[1] *Dept. of Computer Engineering, University of Murcia, Murcia, 30100, Spain*
{jcebrian,jlaragon,jmgarcia}@ditec.um.es
[2] *Dept. of Electrical and Computer Engineering, University of Patras, 26500, Greece*
{kaxiras,ppetoumenos}@ee.upatras.gr

Abstract

In the past years Dynamic Voltage and Frequency Scaling (DVFS) has been an effective technique that allowed microprocessors to match a predefined power budget. However, as process technology shrinks, DVFS becomes less effective (because of the increasing leakage power) and it is getting closer to a point where DVFS won't be useful at all (when static power exceeds dynamic power). In this paper we propose the use of microarchitectural techniques to accurately match a power constraint while maximizing the energy efficiency of the processor. We will predict the processor power consumption at a basic block level, using the consumed power translated into tokens to select between different power-saving micro-architectural techniques. These techniques are orthogonal to DVFS so they can be simultaneously applied. We propose a two-level approach where DVFS acts as a coarse-grained technique to lower the average power while microarchitectural techniques remove all the power spikes efficiently. Experimental results show that the use of power-saving microarchitectural techniques in conjunction with DVFS is up to six times more precise, in terms of total energy consumed (area) over the power budget, than using DVFS alone for matching a predefined power budget. Furthermore, in a near future DVFS will become DFS because lowering the supply voltage will be too expensive in terms of leakage power. At that point, the use of power-saving microarchitectural techniques will become even more energy efficient.

1. Introduction

Modern microprocessors are already designed to be power efficient while increasing, or at least maintaining, current performance levels. However, individual core performance is saturating, and high performance processor designs are moving to multi-core approaches. Even if a low power processor is designed, it will be difficult to sell if it is slower than the previous generation, so designers have to look for a balance between consumption, thermal constraints and performance.

Regardless of the internal hardware implemented on a chip, it is extremely improbable that *all* of its resources will be simultaneously used due to program dependences and/or thread's synchronization. Moreover, designing the thermal envelop for this "worst case" is somewhat expensive and not very efficient. Computer architects, instead of designing the processor for the worst case scenario, look for the average case, and face these "special" cases by using both power saving and thermal management techniques. *Dynamic Thermal Management* is a mechanism that reduces the processor power consumption (and performance) during time intervals so it can cool down. One way to achieve this goal is to set a *power budget* to the processor.

This processor's power budget is not only useful to control power and temperature but also when there is an external limitation on the power consumption independent of the microprocessor (or even the system) that we need to satisfy, without shutting off the whole system. There are also situations where device power constraints are more restrictive than the power needs of a processor at full speed. In most of the cases we cannot afford to design a new processor to meet whatever power constraint because is too expensive. The problem increases when, as usual, power constraints are transitory and after some period of time we want all the processor's performance back. Imagine a computation cluster connected to one or more UPS units to protect from power failures. If there is a power cut, all processors will continue working at full speed consuming all of the UPS battery quickly, and then switching the computers off when the battery is close to run out and, consequently, losing all the

978-1-4244-3750-4/09/$25.00 ©2009 IEEE

work on fly. During the power failure (many times they are of limited duration), if the processors are not doing critical work, it might be more interesting to extend the UPS battery duration at the expense of degrading some performance, than to lose all the work done because the battery runs out.

Another example where setting a power budget could be useful is the case of a computing centre that shares a power supply among all kind of electric devices (i.e., computers, lights, air conditioning, etc.). In a worst case scenario (e.g., in summer at mid-day with all the computers working at full speed), if we integrate some kind of power budget management into the processors, during critical day hours or conditions when the air conditioning is consuming a big part of the total power of the computing centre, we could decrease the power of all processors, lowering the ambient temperature and having more power for the air conditioning. In this way, we could design the power capacity of the computing centre for the average case, reducing its cost.

We need processors able to accurately adapt to a given power budget in real time in an energy-efficient way. One approach to make the processor power go towards a power budget is Dynamic Voltage and Frequency Scaling (DVFS) [6][7][14][16]. DVFS decreases the processor voltage and frequency to reduce its dynamic power, as dynamic power depends on both voltage (quadratically) and frequency (linearly). DVFS is a well known technique already implemented in many commercial processors mainly used by DTM techniques [8]. The major advantage of DVFS is its precision for estimating the final power consumption and performance degradation associated with the voltage and frequency reduction. However, DVFS has some important drawbacks:

- Long transition times between different power modes.
- Long exploration and use windows (in order to amortize DVFS overheads), making it difficult to adapt precisely to the program behaviour.
- When activated, DVFS affects all instructions regardless of their usefulness in the program. Therefore, it cannot exploit situations such as instruction slack, instruction criticality, or low confidence on the predicted path.
- As process technology shrinks, reducing voltage (V_{DD} and V_T) to lower dynamic power consumption becomes impractical since leakage exponentially depends on V_T (see Section 2.1) which will turn DVFS into DFS (*Dynamic Frequency Scaling*) for deep submicron designs. However, DFS is not so energy-efficient since it seriously hurts performance.

In this paper we propose the use of power-saving microarchitectural techniques to accurately match an imposed processor power budget. Our approach will capture and store information about the processor power consumption, either at a cycle level (Power-Token Throttling) or at a basic block level (Basic Block Level Manager), in order to decide whether the next instruction/basic block can be executed based on its previous power consumption and how far the processor is from the power budget. We also introduce an efficient two-level approach that firstly applies DVFS as a coarse-grained approach to lower the average power and, secondly, uses microarchitectural techniques to remove local power spikes. One of the benefits of using microarchitectural techniques is that they can be applied at a cycle level, only in those cycles where the processor exceeds the power budget. The peculiarities of each microarchitectural technique will be detailed in Section 2.2 but, in general terms, these fine-grained techniques:

1. Locate non-critical instructions in cycles exceeding the power budget and delay them to cycles under the budget that are near enough not to make these instructions critical ones.
2. Previous studies have shown that 30% of the overall processor power comes from wrong path instructions [2][15]. Therefore, when the processor is exceeding the power budget, we can reduce the number of low-confident speculative instructions in the pipeline.
3. When the two previous policies are not enough to put the processor under the required power budget, we can throttle the pipeline at different stages in order to meet the given budget.

Experimental results show that the use of power-saving microarchitectural techniques is more energy efficient as well as more accurate (i.e., lower area over the power budget, see Section 4.2 for details) than DVFS alone for driving the processor under the required power budget. The rest of the paper is organized as follows. Section 2 provides some background on power-saving techniques. Section 3 shows a first analysis of the individual techniques and motivates the need for a hybrid approach to match the power budget. Section 4 describes our simulation methodology and reports the main experimental results. Finally, Section 5 shows our concluding remarks.

2. Background and related work

2.1. DVFS

Dynamic voltage and frequency scaling (DVFS) was introduced in the 90's [14], offering a great promise to reduce power consumption in microprocessors. Based on the fact that power consumption depends on both voltage and frequency ($P \propto V_{DD}^2 \times f$), DVFS dynamically scales these terms to save power [6][7][16]. Unfortunately, one of the major concerns about DVFS has been the slow off-chip voltage regulators that lack the ability to adjust to different voltages at small time scales (0.016mV / ns according to [7]). Modern real implementations are limited to temporary coarse-grained adjustments governed by runtime software (i.e., the OS).

In the recent years, researchers and designers have moved to chip multiprocessor architectures as a way of maintaining performance scaling while staying within tight power constraints [1][8]. This trend, coupled with diverse workloads found in modern systems, motivates the need for fast, per-core DVFS control. Kim *et al*. [17] very recently proposed the use of fast on-chip regulators to achieve transition speeds of 30-50mV / ns. This solves one of the major problems of DVFS, but still has some limitations. 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 $\alpha > 1$. That means that we can lower V_{DD} for DVFS as long as we keep 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 hugely increases as it exponentially depends on V_T, which makes leakage the major source of power consumption as the process technology scales below 45nm [3][4][5]; and b) processor reliability is further reduced.

2.2. Microarchitectural techniques

2.2.1. Critical path determination

Data dependencies are one of the main bottlenecks in high performance processors: dependency chains limit the performance of the machine leaving most of the processor structures and logic idle [11]. These chains of dependent instructions are known as the critical path of the code. A processor performance is determined by how fast it can execute the critical path, not by how fast it can execute all of the code.

If we were able to distinguish between instructions that belong to the critical path of the program, we could accelerate their execution to improve the machine performance, or slow down non-critical instructions to reduce power consumption and temperature. In [11] it is proposed a critical path predictor able to predict critical instructions, but the amount of cycles a non-critical instruction can be delayed without becoming critical is really small [13].

2.2.2. Pipeline throttling

Pipeline throttling is a technique that reduces the amount of in-flight instructions in the pipeline to reduce power consumption [2][10][15]. Pipeline throttling can be applied at different stages, producing different effects on both power and performance.

- Instruction flow control: These techniques try to estimate the amount of ILP in the processor by tracking the instructions traversing the pipeline. Authors in [10] propose the decode/commit ratio (DCR) heuristic for estimating the processor's current ILP. We can take advantage of this information and the front-end can be either stopped or slowed down for a small number of cycles to reduce power consumption.

- Confidence estimation: Confidence estimators try to add some additional information to branch predictors so the processor can check how good a prediction is and act accordingly. JRS [9] is one of the most cost-effective confidence estimators: it uses a direct-mapped table accessed by the program counter where it stores how many consecutive hits there are for a branch. When the counter exceeds a threshold, the branch is considered confident.

2.3. Hybrid approaches

There are several proposals that try to merge both DVFS and microarchitectural techniques in a two-level mechanism to benefit from both coarse and fine-grained mechanisms. Sasanka *et al*. propose the use of DVS and some microarchitectural techniques to specifically reduce the power consumption in real time video applications [18]. Their selected micro-architectural techniques try to reduce the power of

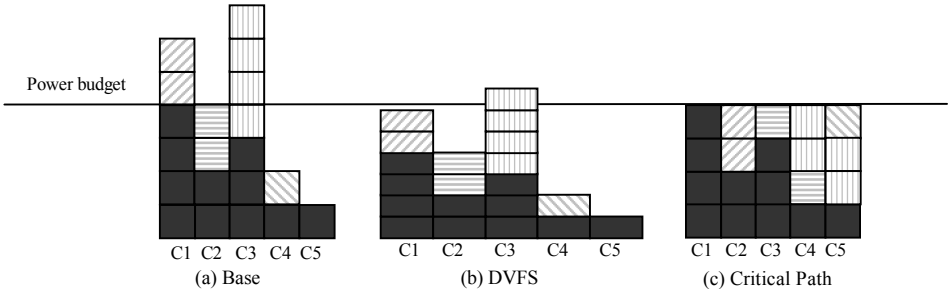

Figure 1. DVFS and critical path effects on power consumption at a cycle level.

functional units and the instruction window. Our proposal is more generic and adaptive, as we do not depend on profiling and the applications studied are SPECint benchmarks. Moreover, they don't use clock gating in their proposal, and the studied benchmarks have special properties that their selected microarchitectural techniques can take advantage of. Although we do not implement any specific micro-architectural technique to reduce the consumption of the instruction window and functional units, the use of clock gating prevents these structures to consume when they are underused. Winter *et al*. [19] also propose the use of a two-level approach that merges DVFS and thread migration to reduce temperature in SMT processors.

3. Power-saving microarchitectural techniques

Our goal is to reduce the processor power consumption to match an imposed power budget in an energy-efficient way. First we need to study how the different microarchitectural techniques behave independently, in order to design an adaptive mechanism that takes advantage of each of their peculiarities. As evaluation metrics, we will measure both the fraction of cycles exceeding the power budget over the total execution cycles as well as the induced performance degradation for the whole SPECint2000 (see Section 4.1 for details about the processor configuration).

3.1. Reactive techniques

Reactive techniques check the processor power consumption every cycle. If the current power exceeds the required budget, the processor applies a certain microarchitectural technique to reduce its power consumption. The major concern about reactive techniques is that they must be applied during enough cycles in order to achieve its low-power effect. If in a given cycle the processor goes over the power budget, we do not know a priori how long this situation will last. Furthermore, reactive techniques are not able to remove all the cycles the processor spends over the power budget since we activate them once the processor power consumption is over the budget, unless we use a predictive approach (as explained in Section 3.2).

3.1.1 Instruction criticality analysis

Instruction criticality analysis is an approach that tries to locate non-critical instructions in cycles where the power budget is exceeded, delaying them to cycles under the power budget. As in [11], we use an 8K-entry table indexed by PC. Each entry has a 6-bit saturating counter that is incremented by 8 if the instruction belongs to the critical path and is decremented by 1 otherwise. This 6KB table (340 times smaller than the L2 cache) introduces a power overhead of around 0.5% which is accounted for in our results. From the proposed policies in [11], we have chosen the "QOld" policy for our implementation because of its simplicity. Each cycle, QOld policy marks as critical the oldest instruction in the instruction queue, unless it is ready. When the instruction becomes ready it is marked as non-critical.

In order to gain some insight about what is happening inside the pipeline, Figure 1 shows the original, DVFS and critical path behaviour for several cycles (where Ci represents cycle i). Solid boxes represent critical instructions while the rest represent non-critical instructions. If we set a power budget like the one in Figure 1, DVFS will slow down all the instructions (actually, DVFS increases cycle length) to match the required budget at the expense of increasing the execution time. On the other hand, by using instruction criticality information, as shown in Figure

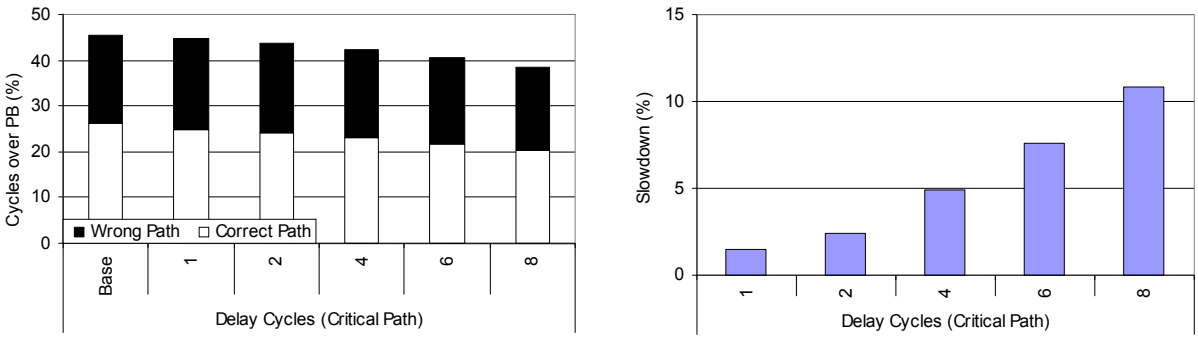

Figure 2. Instruction criticality analysis approach for a power budget of 50%.

1-(c), we locate non-critical instructions, so they can be efficiently delayed to cycles under the budget.

Our first implementation marks in the critical path predictor table, along with the information about the criticality of an instruction, if the last time it was executed the power budget was exceeded. Therefore, if the predictor detects a non-critical instruction previously executed in a cycle over the budget, we will delay its execution.

Figure 2 shows the result of using instruction criticality information alone for an aggressive power budget of 50%. We use this power budget to show the disadvantages of the critical path technique that does not work as expected for aggressive power budgets. The delay cycles (x axis) represents the maximum number of cycles a non-critical instruction can be delayed. First of all, we must notice that, for the base case almost half of the execution cycles are over the power budget. Our critical predictor relies on finding "holes" where it can delay instructions from cycles over the budget. The more cycles a program is under the power budget, the easier to balance instructions so they all execute under the budget. Second, in our current implementation, the critical path predictor has no information about the power cost of each instruction neither how much power over the budget was consumed in each cycle. And even if we had that information, it would be still needed some way of communication between instructions in order to know if enough instructions have been delayed to be under the power budget. To keep it simple, this first approach delays as many non-critical instructions as found when the power budget is exceeded.

3.1.2 Pipeline throttling analysis

This section shows how the pipeline throttling technique behaves individually in terms of both the number of cycles over the power budget the technique is able to reduce as well as the performance degradation for the JRS and DCR approaches.

The evaluated confidence estimator is a modified version of JRS with a 64K-entry table where each entry contains a 2-bit saturating counter (Figure 3). The confidence estimator has a size of 16KB with a power overhead of around 0.3%, also accounted for in our results. When a branch is labelled as non-confident, we either stop or slow down the front-end of the processor. For the DCR approach (Figure 4), we show results when the technique is always enabled (DCR); when we throttle the pipeline only if the power budget is exceeded (DCR + >PB); and when we throttle the pipeline ony if the power budget is exceeded (>PB). When pipeline throttling is activated, we divide both the fetch and decode bandwidth by 2 or 4 (note that we are evaluating a 4-wide issue processor). Figures 3 and 4 show the amount of throttling in the x axis as follows: the base case corresponds to bar 1; bars 2 and 4 correspond to a throttling of 1/2 and 1/4 respectively; finally, bar 0 means a "full-stop" of the fetch unit. All the techniques are applied as long as the trigger condition lasts (e.g., JRS labels a branch as non-confident; or the DCR condition is met[1] and the power budget is exceeded) plus 3 cycles.

First thing we can notice in Figures 3 and 4 is that the major reduction in the cycles over power budget comes from cycles that belong to a wrong path, as intended. However, none of these techniques alone is good enough to match the required power budget of 50% unless performance is highly degraded (e.g., in Figure 4, bars 4 and 0). Therefore, a predictive approach is needed in order to accurately match the required power budget in an energy-efficient way.

[1] Three times more committed than decoded instructions.

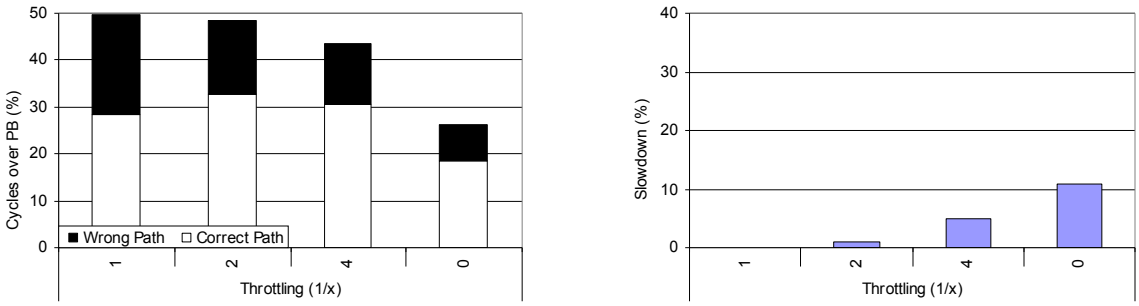

Figure 3. Cycles over PB and slowdown for JRS-based throttling (power budget = 50%).

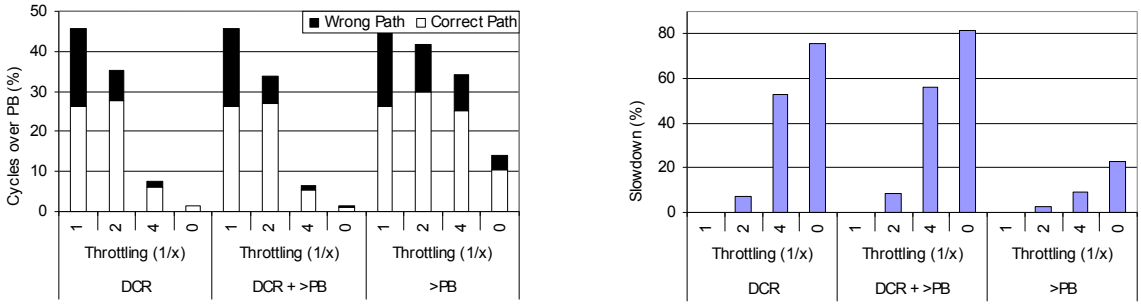

Figure 4. Cycles over PB and slowdown for DCR-based throttling (power budget = 50%).

3.2. Predictive techniques

Predictive techniques will capture and store information about the processor power consumption, either at a cycle level (Power-Token Throttling) or at a basic block level (Basic Block Level Manager), in order to decide whether the next instruction/basic block can be executed based on its previous power consumption. For predictive techniques to work we need a way to measure power consumption at a cycle level. Our proposal uses a novel Power-Token approach to dynamically estimate power consumption.

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 previous execution. This table introduces a power overhead of around 0.5% which is accounted for in our results. The power consumed by an instruction is calculated at commit stage by adding the base power consumption of the instruction (i.e., all the regular accesses to structures done by that instruction) plus a dynamic component that depends on the time it spends on the pipeline. This dynamic component is due to the combined RUU wakeup-matching logic power that we divide between all the active instructions in the RUU. Therefore, in order to work with power-token units in a simple way, we define a power-token unit as the joules consumed by one instruction staying in the RUU for one cycle. In this way, the number of power-tokens consumed by an instruction will be calculated as the addition of its base power-tokens plus the amount of cycles it spends in the RUU. The PTHT is updated with the 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 Section 4.1. 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 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).

Finally, the overall processor power consumption can be easily estimated in a given cycle based on the instructions that are traversing the pipeline without using performance counters by simply accumulating the power-tokens (as provided by the PTHT) of each instruction being fetched.

3.2.1 Power-Token Throttling (PTT)

This novel technique estimates the power consumption of the instructions inside the pipeline in a given cycle by means of accounting for the power-

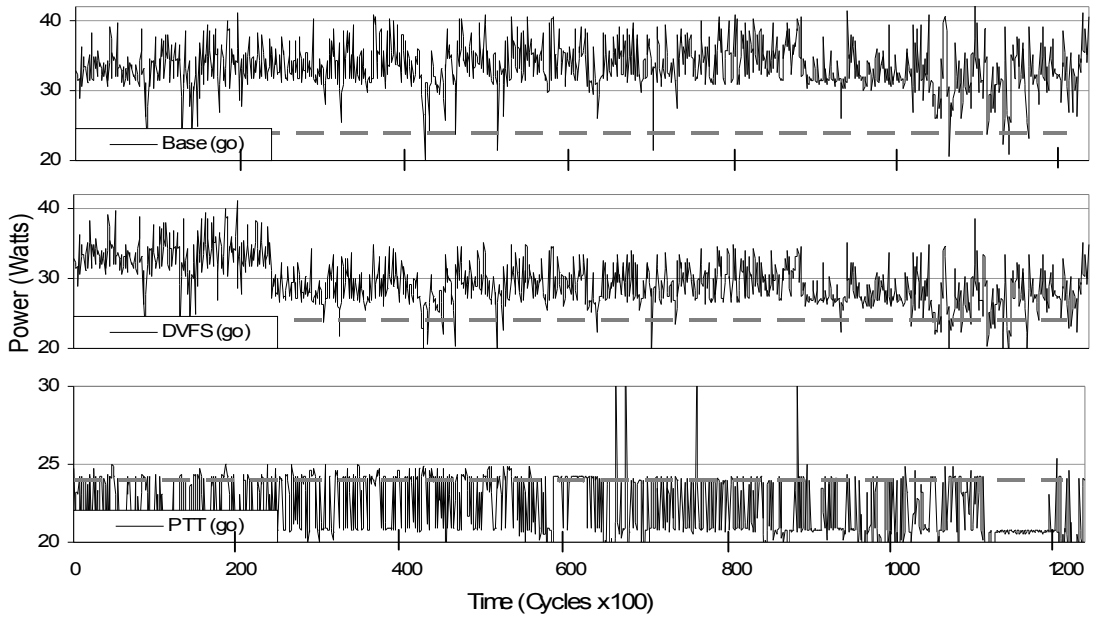

Figure 5. Detailed per-cycle power consumption for the "go" benchmark.

tokens they consume. When the power of the instructions inside the processor exceeds the budget, the fetch stage is stalled until enough committed instructions leave the pipeline, releasing their power-tokens, and eventually lowering the current total power, allowing the introduction of new instructions.

We have also evaluated a modified version of the Power-Token Throttling approach (labelled as Power-Token Throttling CP) that uses a Critical Path Predictor to detect critical instructions. In this way, even if we are over the power budget, we allow these instructions to continue their execution. This version is less aggressive than the original PTT.

In order to gain some insight on the power consumption behaviour of each technique, Figure 5 shows the average power consumption for the "go" benchmark. For the sake of visibility, we only plot the power information for a window of 120Kcycles. The dashed horizontal line represents the required power budget (set to 24W, which corresponds to a 50% power budget over the peak power of 48W, see Section 4.2). The current DVFS implementation (as in [1]) calculates the average power consumption over an exploration window of 500Kcycles, and modifies the voltage and frequency accordingly (from a set of pairs voltage/frequency modes; see Section 4.4) to match the desired power budget. For the window plotted in Figure 5-middle, the average power consumption of the exploration window is either under the power budget or the current working mode is the closest to the budget, therefore DVFS does nothing to force all cycles under the power budget. On the other hand, the Power-Token Throttling approach (Figure 5-bottom), accurately follows the power budget at a cycle level, as it knows in advance how much power (in tokens) the next instruction consumes. If we cannot afford to execute it, we wait until a committed instruction leaves the pipeline and there is enough power budget left to burn in the new instruction. Spikes in the PTT plot are due to branches that are left to enter into the pipeline in order to discover eventual mispredictions as soon as possible.

3.2.2. Basic Block Level Manager (BBLM)

This novel technique uses the last observed number of power-tokens consumed by a basic block (a stream of instructions between two branches) as well as the power budget left in order to decide what technique should be applied from the following: none, critical path, confidence estimation throttling, or decode-commit ratio throttling – from less to more aggressive.

The power consumed by a basic block is measured as the addition of the power-tokens of every instruction that belongs to that basic block. This power is stored in the branch predictor entry of the branch that points to the start of that basic block. This introduces 9 additional bits per entry in the branch predictor which corresponds to a negligible 0.3% power increase in the total processor power consumption, again accounted for in our results. When a branch is predicted, it is also obtained the last power

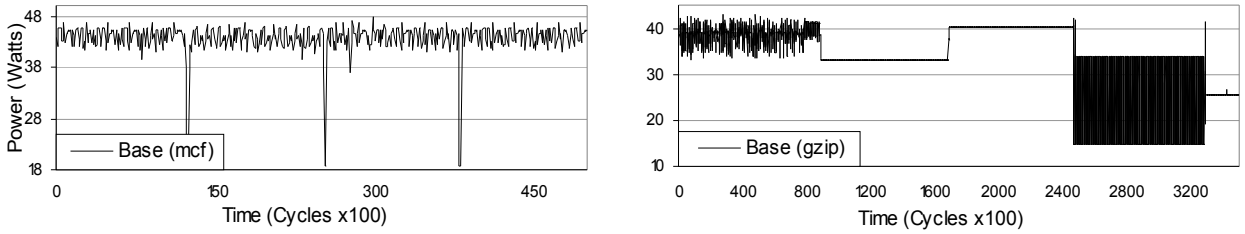

Figure 6. Power consumption for "mcf" and "gzip" benchmarks as a function of time.

history for the subsequent basic block, and we estimate how far from the power budget we will get if we execute that basic block. Depending on how far the estimated power is from the budget, we select a different power saving technique. In addition, when updating the branch predictor's saturating counter, we also update the power-tokens consumed by the subsequent basic block (that corresponds the next predicted path – we use a gshare predictor).

For the BBLM approach to decide what technique should be applied we define two threshold values, X and Y. Therefore, for a fraction of power over the budget lower than X we will use the first technique (i.e., critical path, which is the least aggressive), from X to Y we will use the second technique (i.e., confidence estimation throttling), and for a power over the budget greater than Y we will use the third technique (i.e., decode-commit ratio throttling, which is the most aggressive). Techniques are disabled progressively, once we get under the power budget, in reverse order. We performed an experimental study to determine the best values for these thresholds (best power budget matching with minimal performance degradation), and discovered that for the current processor configuration the best thresholds were X=15 and Y=65.

It is important to note that a very precise basic block power prediction is not really relevant, as long as the selected technique is the same for the same basic block (which is actually the common case).

3.2.3. Two-level approach

Microarchitectural techniques do not work efficiently when the processor spends a great number of cycles over the power budget, as we will see in Section 4, since there are few chances to reduce consumption without degrading performance. On the other hand, DVFS is extremely inaccurate when there are power spikes, because the influence of power spikes on the average search window power consumption is really low. As we can see in Figure 6-left, not all benchmarks have the same average power consumption (compare it with "go" in Figure 5-top). It can be noticed that "mcf" has an average power consumption close to the maximum power consumption (approx. 48W). If we only use microarchitectural techniques for these benchmarks we may not get close to the power budget, or the performance degradation will be quite high. Moreover, benchmarks exhibit different program phases with different average power consumptions (as shown in Figure 6-right). Phases that if we are able to detect we could use in our advantage, using only microarchitectural techniques in phases close to the power budget and a coarse-grained approach (DVFS) in phases far from the power budget.

Our proposal consists of a two-level approach: first we apply DVFS to take coarse-grained decisions about power consumption, and secondly we apply microarchitectural techniques for fine-grained decisions (mainly removing power spikes). This two level approach has a twofold benefit. We increase the DVFS accuracy for matching a power budget, while at the same time we do not need all the DVFS power modes to reach a power budget. Only the less aggressive power modes are enough in our simulations to accurately match the predefined power budget. It is important to note that, as discussed in Section 2.1, when the process technology goes below 65nm the reduction on V_T will be limited by both reliability and leakage power, therefore, it is not that DVFS does not use the extreme power modes, but having those power modes will be infeasible.

4. Experimental results

4.1. Simulation methology

To evaluate the energy-efficiency of both DVFS and microarchitectural techniques we have used the SPECint2000 benchmark suite. All benchmarks were compiled with maximum optimizations (-O4 -fast) by the Compaq Alpha compiler and they were run using a modified version of HotLeakage power-performance

Table I. Processor configuration.

Processor Core	
Process Technology:	70 nanometres
Frequency:	5600 MHz
Instruction Window	128 RUU, 64 LSQ
Decode Width:	4 inst/cycle
Issue Width:	4 inst/cycle
Functional Units:	4 Int Alu; 2 Int Mult 4 FP Alu; 2 FP Mult
Pipeline:	22 stages
Branch Predictor:	64KB, 16 bit history GSHARE
Memory Hierarchy	
L1 I-cache:	64KB, 2-way
L1 D-cache:	64KB, 2-way
L2 cache:	2MB, 4-way, unified

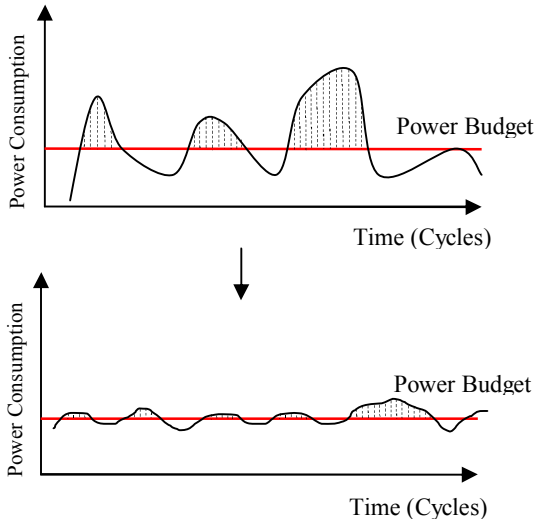

Figure 7. Shadowed areas show the exceeded power over the budget (Area over PB).

simulator [12] that includes the dynamic power model for the evaluated microarchitectural approaches as well as their associated power overhead. All benchmarks were run to completion using the reduced input data set (test). Table I shows the configuration of the simulated architecture.

4.2. A power budget of what? (100% usage ≠ 100% power consumption)

What we need to measure is whether the processor exceeds a preset power budget, usually represented as a percentage of the total power consumed by the processor. The worst case scenario (i.e., the peak consumption of a processor) would be achieved by a program that uses all the processor resources at once.

Our base processor has a peak power consumption of 75 Watts (73W on average). However, when a circuit-level technique such as clock gating is enabled, the average power consumption for the SPECint2000 falls to 25 Watts with a peak power consumption of 48 Watts. We will use that 48 Watts peak power consumption (using clock gating "cc3") as our reference power consumption (i.e., 100% power budget), which means the power consumption of the selected processor without any power saving technique.

Next sections show the simulation results for the SPECint2000 benchmark suite for: a) fraction of cycles over the power budget (Cycles over PB); b) total power exceeded over the budget (Area over PB); and c) normalized energy. The metric "area over the power budget" tells us how accurate a technique is for matching a power budget and how close we are from the budget (as illustrated in Figure 7). We use this metric instead of the average power since the standard deviation of the per-cycle power consumption is quite high. The processor has periods of high power consumption hidden by periods of low power consumption (branch mispredictions, cache misses, etc.), therefore, the average power is not a good metric for what is really happening inside the processor.

4.3. Cycles over PB and area distribution

Figure 8 shows the fraction of cycles over power budget whereas Figure 9 shows the area over the power budget for different budgets and benchmarks before applying any power saving technique. Evaluated PBs range from 95% of the original peak power to 40%.

As we can see in Figures 8 and 9, both the amount of cycles and the area the processor spends over the power budget is almost negligible for high budgets (95%-70%), due to the effects of clock gating on power consumption, but being highly noticeable for power budgets under 65%. As explained before, DVFS will be unable to find power spikes for low power budgets, as it works with the average power consumption of its long explore windows. On the

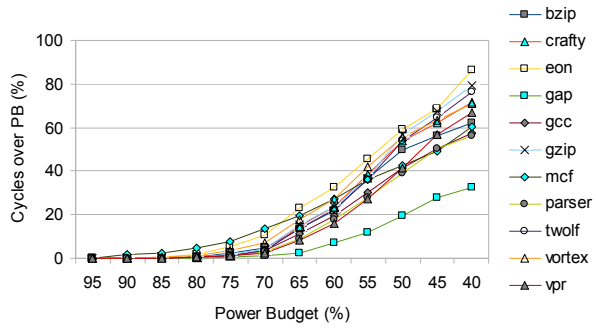

Figure 8. Cycles over PB distribution.

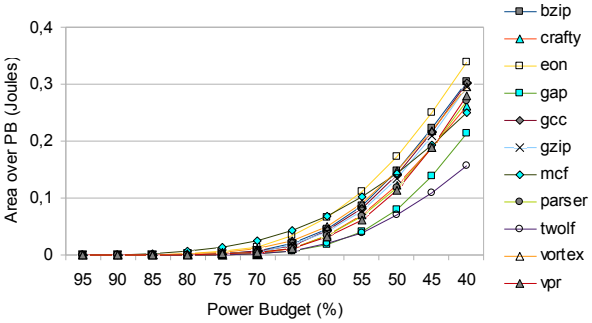

Figure 9. Area over PB distribution.

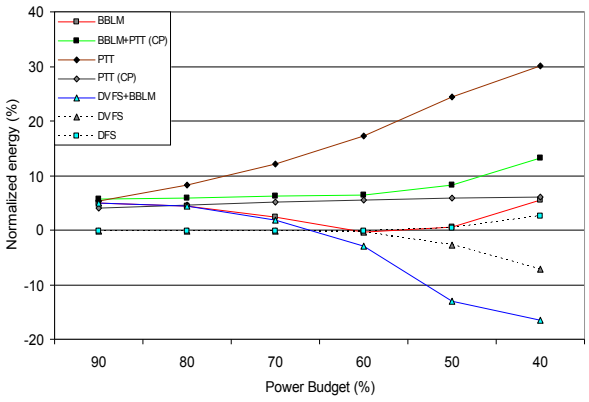

Figure 10. Normalized energy consumption.

other hand, microarchitectural techniques will detect these spikes and remove them whenever possible.

4.4. Efficiency on matching a power budget

This subsection evaluates the energy-efficiency of the proposed microarchitectural techniques as well as their accuracy for matching an imposed power budget. In order to reduce the extension of this section we only report results for the following techniques: DVFS, PTT and BBLM (with and without using Critical Path information), and the two-level approach (DVFS+BBLM).

The evaluated DVFS approach uses the implementation proposed in [1] and discussed in Section 2.1. DVFS calculates the average power consumption over a predefined amount of cycles (search interval). If the mechanism finds out that the average power consumption is over the budget, DVFS changes to a pair of voltage and frequency values from a set of predefined working modes in order to reduce the average power. DVFS uses a search interval of 500Kcycles with a transition time between modes set to 50 mV/ns (as in [17]), so it takes only 6 cycles to switch from one mode to another[2]. The evaluated working modes are the following:

- For DVFS: there are five modes (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).
- For the limited DVFS: this version of DVFS is used in the two-level approach. The power modes are limited to (100% V_{DD}, 100% f), (95% V_{DD}, 95% f), and (90% V_{DD}, 90% f).
- For DFS: only scales frequency as needed (V_{DD} remains unchanged).

The studied BBLM uses the configuration parameters and techniques proposed in Section 3.2.2. As explained before, the selected thresholds are X=15 and Y=65. Critical path (CP) is used as the first technique, JRS-throttling as the second technique, and DCR-throttling as the third one.

Figure 10 shows the normalized energy consumption for the different techniques and power budgets. As we can see, the two-level approach (DVFS+BBLM) is the most energy-efficient technique for all the studied power budgets, and especially for the very restrictive power budgets. The rest of microarchitectural techniques (except PTT) as well as DFS show not so high energy degradation (between 4% and 10%). BBLM alone shows a similar energy-efficiency as DFS up to a power budget of 50% while reducing four times more the area over the PB.

Moreover, as we are working under an imposed power budget, Figure 11 shows how accurate each technique is when trying to meet the power constraint as it plots the relative area over the power budget. It can be observed that all microarchitectural techniques are far more accurate than DVFS alone when adapting to the imposed power budget. For power budgets

[2] For HotLeakage V_{DD} at 70nm is 1V, so each 5% reduction in voltage translates into 50mV. That means it will take one ns to switch between modes. As the processor runs at 5.6Ghz (or 5.6cycles/ns), it will take 6 cycles to change between modes.

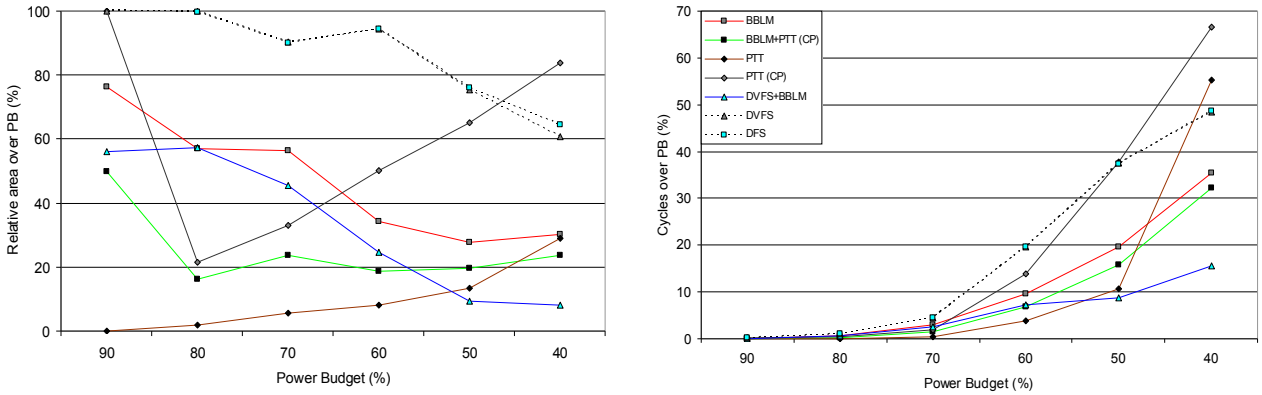

Figure 11. Relative area and fraction cycles over PB for different power budgets.

between 90% and 70%, the BBLM+PTT (CP) is the best approach: 25% of area over PB is left after applying this technique with a total energy increase of only 6% (Figure 10).

When we move to more restrictive power budgets of 60%, the two-level approach (DVFS+BBLM) is 4% more energy-efficient than DVFS alone while the accuracy of the former is three times better (25% of relative area over the power budget is left by the two-level approach whereas DVFS barely reduces the area to a 90% of the original). For an extreme power budget of 40%, the two-level approach gets even better: only 10% of relative area over the power budget is left, in contrast with the 60% that reports DVFS. In addition, the two-level approach for this extreme power budget is 11% more energy-efficient than DVFS and 20% more energy-efficient than DFS.

In general terms, DVFS and DFS are coarse-grained approaches unable to remove the huge amount of power spikes that appear when executing typical applications, which are even more apparent when considering low power budgets, making coarse-grained approaches far less accurate than microarchitectural techniques.

5. Conclusions

In this era of power-aware microprocessors, engineers look for general designs so that their processors can be used in different kinds of gadgets. These gadgets usually have different power requirements and it is needed some way to tell the processor the maximum power it can consume. The challenge is even more interesting if we think about external temporary power requirements that the processor can either match or simply shut-off. In many cases, the shut-off option is not even viable (e.g., for critical systems). In addition, the thermal envelop design in microprocessors cannot be done for the worst case scenario, because the production price highly increases, and the processor hardly ever reaches those temperatures. Designers look for the average case and develop techniques to treat the special cases where there are temporary thermal problems. All these challenges have a common solution: set a temporary power budget to the processor, limiting its power and temperature.

In the past years microprocessors matched this power budget by using dynamic voltage and frequency scaling (DVFS). However, with the increasing scaling technology DVFS gets less effective (because of the leakage power) and it is getting closer to a point where DVFS won't be useful at all. Moreover, DVFS is a coarse-grained approach and its accuracy when matching a power budget is far from appropriate. On the other hand, we can make use of power saving microarchitectural techniques. These techniques work at a cycle level instead of searching windows of thousands of cycles, so they are highly accurate.

This paper proposes the use of microarchitectural techniques to precisely match a predefined power budget while maximizing energy efficiency. We have introduced a novel adaptive technique, Basic Block Level Manager (BBLM), which uses basic block power consumption history (translated into power-tokens) in order to determine the best power saving technique for the current and near future processor power consumption. We have also proposed a two-level approach that combines both microarchitectural techniques and DVFS to take advantage of their best qualities. DVFS acts as a coarse-grained technique to lower the average power consumption while microarchitectural techniques remove all the power spikes efficiently. The two-level approach

(BBLM+DVFS) is able to beat DVFS alone in both energy efficiency (up to 11% more energy efficient) and area exceeded over the power budget (up to 6 times less area).

Moreover, as processor technologies advance towards deep submicron (<65nm), leakage power becomes the major source of power consumption. DVFS exponentially increases the leakage power consumption if we reduce the threshold voltage, in addition to seriously affecting the reliability of the chip. That will shortly transform DVFS into DFS, and any power reduction will come at the cost of a significant performance degradation. In that scenario, we have shown how microarchitectural techniques are even more energy efficient and accurate for adapting to a required power budget than DFS (the two-level approach is up to 20% more energy efficient and up to 6 times less area exceeded than DFS alone).

Acknowledgements

This work has been jointly supported by the Fundación Séneca (Agencia Regional de Ciencia y Tecnología, Región de Murcia) under grant 05831/PI/07, and also by the Spanish MEC and European Commission FEDER funds under grants "Consolider Ingenio-2010 CSD2006-00046" and "TIN2006-15516-C04-03', as well as by the EU FP7 NoE HiPEAC IST-217068.

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 *Proc. of the 39th Int. Symposium on Microarchitecture,* 2006.

[2] Aragon, J.L.; Gonzalez, J and Gonzalez, A. Power-aware control speculation thought selective throttling. In *Proc. of the 9th Int. Symp. on High-Performance Computer Architecture,* 2003.

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

[4] Kim, N.S.; Austin, T. *et al*. Leakage Current: Moore's Law Meets Static Power. In *IEEE Computer,* vol.36, issue.12 2003.

[5] Flynn, M.J. and Hung P. Microprocessor Design Issues: Thoughts on the Road Ahead. In *IEEE Micro*, vol.25, no. 3, pp. 16-31, May/Jun, 2005.

[6] Semeraro, G.; Magklis, G. *et al*. Energy-Efficient Processor Design Using Multiple Clock Domains with Dynamic Voltage and Frequency Scaling. In *Proc. of the 8th Int.Symp. on High-Performance Computer Architecture*, 2002.

[7] Wu, Q.; Juang, P.; Martonosi, M. and Clark, D. W. Formal Online Methods for Voltage/Frequency Control in Multiple Clock Domain Microprocessors. In *Proc. of the 11th Int. Conf. on Arch. Support for Programming Lang. and Operating Systems* (ASPLOS-XI), 2004.

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

[9] Jacobsen, E.; Rotenberg, E. and Smith, J. E.. Assigning Confidence to Conditional Branch Predictions. In *Proc.of the Int. Symp. on Microarchitecture*, December 1996.

[10] Baniasadi, A. and Moshovos, A. Instruction Flow-Based Front-end Throttling for Power-Aware High-Performance Processors. In *Proc. of the Int. Symp. on Low Power Elect. and Design*. 2001.

[11] Tune, E.; Liang, D.; Tullsen, D.M. and Calder, B. Dynamic Prediction of Critical Path Instructions. In *Proc. of the 7th Int. Symp. on High-Performance Computer Arch.* 2001

[12] Zhang, Y.; Paritkh, D.; *et al*. HotLeakage: a temperature-aware model of subthreshold and gate leakage for architects. *Technical Report, Dept. Comp. Science, U. Virginia*, 2003.

[13] Casmira, J. and Grunwald, D. Dynamic Instruction Scheduling Slack. *In Proc. of the KoolChips Workshop*, 2000.

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

[15] Manne, S.; Klauser, A. and Grunwald, D. Pipeline Gating: Speculation Control For Energy Reduction. In *Proc. of the Int. Symp. on Computer Architecture*, 1998.

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

[17] Kim, W.; Gupta, M. S. *et al*. System Level Analysis of Fast, Per-Core DVFS using On-Chip Switching Regulators. In *Proc. of the 14th Int. Symp. on High-Perf. Computer Architecture*. 2008.

[18] Sasanka, R.; Hughes, C. J. and Adve, S.V. Joint Local and Global Hardware Adaptations for Energy. In *Proc. of the 10th Intl. Conf. on Arch Support for Programming Languages and Operating Systems*, 2002.

[19] 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.