Interval-Based Models for Run-Time DVFS Orchestration in SuperScalar Processors

Georgios Keramidas
Industrial Systems Institute
Patras, Greece
keramidas@isi.gr

Vasileios Spiliopoulos
Department of Electrical and Computer Engineering
University of Patras, Greece
vspil@ece.upatras.gr

Stefanos Kaxiras
Department of Electrical and Computer Engineering
University of Patras, Greece
kaxiras@ece.upatras.gr

ABSTRACT

We develop two simple interval-based models for dynamic superscalar processors. These models allow us to: i) predict with great accuracy performance and power consumption under various frequency and voltage combinations and ii) implement targeted DVFS policies at run-time. The models analyze program execution in intervals —steady-state and miss-event intervals. Intervals are signalled by miss events (L2-misses in our case) that upset the "steady state" execution of the program and are ended when the pipeline reaches again a steady state. The first model is fed by an approximation of the stall cycles (the time the processor instruction window is blocked) due to long-latency L2-misses. The second model improves on this approximation using as input the occupancy of the L2's miss-handling registers (MSHRs). Despite their simplicity these models prove to be accurate in predicting the performance (and energy) for any target frequency/voltage setting, yielding average errors of 2.1% and 0.2% respectively.

Besides modelling, we show that the methodology we propose is powerful enough to implement (at run-time) various DVFS policies: "operate at optimal EDP" or "ED^2P," or even "reduce ED^2P within specific performance constraints." Approaches based on the two models require minimal hardware cost: two counters for measuring the duration of the steady state and the miss-event intervals and some comparison logic. To validate our methodology we use a cycle-accurate simulator and the benchmarks provided by the SPEC2K suite. Our results indicate that our proposed run-time mechanism is able to orchestrate different DVFS policies with great success yielding negligible errors —bellow 1.5% on average.

Categories and Subject Descriptors

C.0 **[Genenal]:** Modeling of computer architecture; C.1.3 **[Processor Architectures]:** Other Architecture Styles—*Adaptable architectures*.

General Terms

Algorithms, Management, Design.

CF'10, May 17–19, 2010, Bertinoro, Italy.
Copyright 2010 ACM 978-1-4503-0044-5/10/05...$10.00.

Keywords

Dynamic Voltage and Frequency Scaling, Performance and Power Modeling, Superscalar Out-of-Order Processors.

1. INTRODUCTION

The power-aware architecture landscape has been dominated by techniques based on supply voltage and clock frequency scaling. Dynamic Voltage and Frequency Scaling (DVFS) offers great opportunities to dramatically reduce energy/power consumption by adjusting both voltage and frequency levels of a system according to the changing characteristics of its workloads. The great potential of the DVFS in energy/power savings has been widely studied in a variety of research communities (from circuit designers to system designers) and has been also used in commercial systems. Intel XScale and AMD Mobile K6 Plus are typical low-power processors that feature DVFS management capabilities. Example processors from the high-performance area are the AMD Opteron Quad-Core and the Intel Core i7 processor.

In general, the heart of the DVFS techniques is the exploitation of the system *slack* or "*idleness*." Their objective is to take advantage of slack so that performance is affected little by frequency scaling while at the same time a cubic benefit in power consumption (see Section 2) —with the help of voltage scaling— is achieved. Slack can appear at different levels and various approaches have been proposed for each level. According to [11], there are three major levels where DVFS decisions can be taken: i) the system level based on system slack, ii) the program or program-phase level based on instruction slack, and iii) hardware level based on hardware slack. More details can be found in [11]. In this work, we concern ourselves with the instruction slack due to the long latency memory operations (off-chip accesses).

There is an abundance of prior work which aims to exploit instruction slack at run-time [8,9,13,18] as well as at compile-time [14,19]. In addition, the current trend from scaling the processor frequency to scaling the number of cores that can reside in a single die enables new opportunities for the DVFS i.e., *per-core* DVFS in CMPs [3,7]. In this paper, we explore the issue of run-time DVFS management of an out-of-order superscalar processor at the microarchitectural level.

We provide two simple analytical models that are able to drive run-time DVFS decisions. These models allow us to predict with great success performance and power consumption under various frequency/voltage combinations. Our modeling methodology is inspired by the interval-based performance model presented initially by Karkhanis and Smith [10] and further refined by Eyerman et. al. [6]. Similarly to the performance model, we examine power consumption in dynamically scheduled superscalar processors by dividing their execution in intervals. Intervals are marked by miss-events that upset the "steady state" execution of the program. A *miss-interval* starts with a miss-event (off-chip load accesses in our case), and lasts until the pipeline reaches again a steady state (a period related to the memory latency). Periods between miss-intervals are steady–state intervals. *The realization that drives this work is that core frequency scaling in these models is nothing more than changing the memory latency in cycles.*

The first model, called *stall-based model*, relies on the number of cycles during which the processor's instruction window is blocked due to a long latency miss i.e., when the head of the reorder buffer (ROB) is occupied by a performance-critical L2 load miss. The second model, called *miss-based model*, is a refinement of the first model and uses as input the occupancy of the L2's MSHRs (a common structure used in all modern processors). *Both models require minimal input and allow us to accurately predict processor performance and energy consumption under any target DVFS setting (V/f points) relative to the situation where measurements are taken. This, in turn, allows us to derive directly (using calculations) the V/f points to optimize, for example, EDP or ED^{2}P without having to go through a step-by-step search pattern of such points.*

The miss-based model is considerably more accurate than the stall-based model. This is because the stall-based model assumes that the stall time due to a long-latency miss is equivalent to the memory latency (measure in cycles). Although this is a good approximation, it is a source of error too. This is because, *the stall-based model disregards the time the processor continues to issue useful instructions after an L2-miss*. This time is not subject to frequency scaling or it does not depend on the latency of the main memory (Section 4). Recall that our conceptual view of the core frequency scaling is just a change in the memory latency measured in cycles. The miss-based model addresses this weakness. *The miss-based model divides the miss-interval (the time upon the face of a new L2-miss till the program reaches again the steady state) into two distinct regions (Section 4).* Of course the addition of the two regions equals the memory latency. The first region includes the amount of time (in cycles) it takes to fill the entire ROB (called ROB-fill) which is not vulnerable to clock frequency scaling, *aka inelastic to frequency scaling*. The second region accounts for the time the processor is stalled due to the L2-miss (no instructions are issued). This time is susceptible to clock frequency, *aka fully elastic to frequency scaling.* The end result is that the stall-based model produces performance estimates under various clock frequencies with an error of 2.1% (average), while the estimates of the miss-based model improve to less than 0.2% error (average).

In addition to modeling, we show that our methodology is powerful enough to enforce various DVFS policies at run-time. The models allow us to directly calculate the effects on performance and energy of any possible DVFS setting of interest. We show how we can derive at run-time such DVFS points for three different policies: "operate at optimal EDP," "operate at optimal ED^{2}P", and "minimize ED^{2}P within specific performance constraints." Using detailed cycle-accurate simulations and all the benchmarks of the SPEC2K suite, we show that the proposed run-time mechanism is able to drive the above DVFS policies with great success and achieves that with minimal hardware cost: just two counters for measuring the duration of the steady state and the miss-event intervals and some comparison logic.

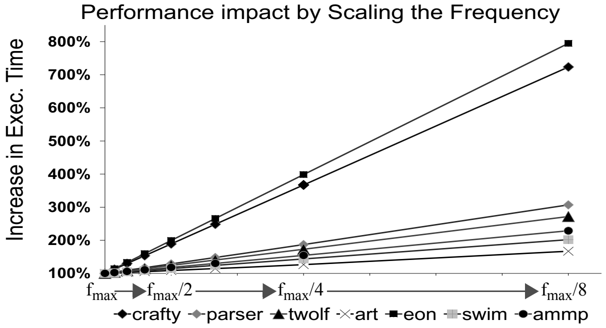

Figure 1. Performance impact by scaling the frequency in a subset of the SPEC2K benchmarks.

Structure of the paper. The remainder of this paper is organized as follows: Section 2 motivates this work and discusses related work. Section 3 describes the experimental framework used in this paper. In Section 4 we present and evaluate the two proposed models. Section 5 shows how the proposed models can be used to calculate optimal EDP and ED^{2}P V/f points. Section 6 provides the evaluation of our run-time DVFS management mechanisms. Finally, Section 7 offers our conclusions.

2. BACKGROUND, MOTIVATION AND RELATED WORK

Dynamic power is proportional to f x C_{load} x V^2 where f is the system clock frequency, C_{load} is the effective capacity, and V is the supply voltage [11]. If the voltage is reduced by some factor, then the dynamic power will be reduced by the square of that factor. Scaling down the voltage, however, requires a commensurate reduction in clock frequency, since transistor speed is reduced. The benefit of this is that within a given system, scaling supply voltage down offers the potential of a cubic reduction in power dissipation. This cubic relation explains the effectiveness of DVFS techniques in power/energy savings. The downside is that it may also linearly degrade performance. In addition to dynamic power benefits, reducing voltage, reduces also static power [11].

The great opportunities for power/energy savings offered by DVFS were a key incentive for designers to devise techniques for DVFS optimizations. Various techniques have been proposed that can be applied either offline, with analysis performed by the compiler, or online. As we have already mentioned, the exploitation of the system slack is at the heart of every DVFS technique. In this work, we target the instruction slack caused by long latency memory operations (off-chip accesses) i.e., we explore the use of the "memory boundeness" of a program as the key metric that should drive the DVFS decisions. Figure 1, which serves as a proof-of-concept of our work, depicts the performance impact of scaling the frequency in seven benchmarks of the SPEC2K suite. As we see, different benchmarks exhibit different DVFS behavior. It can be easily proven that the culprit is the number of off-chip memory operations. For example *eon* experiences a proportional increase in its execution time which means that *eon* represents a compute-bound application. At the other end is *art*, which is one of the most memory intensive benchmarks of the SPEC2K suite and as a result its performance is hardly affected by scaling the frequency. The slope is almost flat. The rest of the benchmarks stand between those two extremes. The results presented in Figure 1 reveal that DVFS should be carefully applied to programs, otherwise system performance may be seriously hurt. Isci et. al. [7] utilize this linear relationship between the frequency and corresponding increase in execution time in order to guard the power consumed in a multiprocessor system, but as the authors point out accurate results can only be taken if the frequency is scaled within 15% of its nominal operating point.

One of the first approaches for DVFS management was by Li et. al. [13]. The authors propose a heuristic mechanism, called VSV, for fine-grain DVFS management which employs an on-chip power network with two discrete supply voltages. Li et. al. employ two empirical FSMs —down and up FSM— to scale the frequency according to the amount of parallelism in the instruction stream (ILP). In the absence of ILP under an L2-miss, the voltage is scaled down and when the L2-miss is serviced the voltage is scaled up again. Although being a successful approach, Li et. al. take DVFS decisions based on heuristics and their proposal is not able to get benefit from the multiple available frequency settings that exist in modern contemporary microprocessors (e.g., 32 frequency steps in Transmeta Crusoe and 320 in the Intel XScale). In other words, Li et. al. provide a solution only for a single point of the design space.

Wu et. al. [18] propose an analytical model by studying heuristic methods using dynamic compilation techniques. Their goal is to dynamically insert DVFS adjustments at locations determined to be most fruitful. To decide on the optimization, the analytic model determines whether a frequently-executing program region is memory or CPU bound. The categorization is done based on the number of the existing memory operations in each program region. If the region is clearly memory bound, it is instrumented with DVFS instructions. If the code is CPU–bound, it is left alone since slowing it down could seriously degrade performance [18]. While this model is good enough to determine the "memory-boundeness" of a program phase, it cannot predict the resultant DVFS benefit and as a consequence fails to give accurate instruction on how much to DVFS. In addition, Wu et. al. do not account for the impact of Memory-Level-Parallelism (whether a miss is isolated or occurs in parallel with other misses) and therefore their proposal leads to over-estimation of the DVFS potential.

Analytical models for DVFS adjustments at compile-time have also been proposed by Xie et. al. [19] and by Maglis et. al. [14]. Xie et. al. produce an analytical model in order to explore the opportunities of compiler techniques, while Maglis et. al. use the compiler to insert reconfiguration instructions into applications using profile-driven binary rewriting, targeting mainly multiple clock-domain processors. However, compile-time techniques have limitations because they fail to capture the dynamic behavior of the applications, they require time and memory-consuming static analysis, and their results are usually input-dependent.

Techniques for online DVFS management include the work of Isci et. al. [8,9]. Since most general-purpose processors include a suite of user-readable hardware performance counters, it is possible to record event counts to build up a history of program behavior. In particular, Isci et. al. demonstrate how these event counts can be viewed as identifying "fingerprints" of the program's power behavior [9]. More recently, Isci et. al. elaborated on their technique by including a predictor that predicts future power behavior based on recently observed values [8].

To the best of our knowledge Eyerman and Eeckhout, independently and concurrently to our work (including the work appearing in [15]), have derived similar to ours analytical models to assess the profitability of DVFS at run-time [5].

Finally, DVFS is also used to improve the reliability of a uniprocessor system [16], to mitigate the impact of process variation in CMPs [17] and to eliminate localized thermal problems in modern microprocessors [4].

3. EXPERIMENTAL FRAMEWORK

Simulator and Benchmarks. The experiments are performed using a detailed, cycle-accurate simulator of superscalar processor. The execution core is a 4-way out-of-order processor. We simulate a 32K, 64 byte block, 4-way, dual-ported, 1-cycle latency, L1 data cache and a 256K, 8-way, 10-cycle, unified, on-chip L2 cache. The main memory has 200 cycles latency. The baseline configuration also includes a 64-entry ROB with a 32-entry instruction queue — smaller and larger instruction windows have also been explored and they show similar results. In this work, our intention is to assess the DVFS behavior of the applications based on their off-chip data accesses, hence we use a relatively large instruction L1 to preclude instruction misses from polluting the L2. Energy estimates are based on the power models of Wattch [2].

The benchmark suite for this study consists of all the SPEC2K benchmarks. All benchmarks are run with their reference inputs. For each program, we skip the first 1B committed instructions. Finally, for the results in Section 4 (model validation), we simulate 200M committed instructions (to limit the simulation time in our experiments), while the experiments performed in Section 6 (run-time policies) are for 3B committed instructions (after skipping).

Technology Parameters and DVFS Settings. We use process parameters for a 70nm CMOS technology and we assume that the highest supply voltage (V_{max}) is 1V and the highest clock frequency (f_{max}) is 4 GHz. During frequency scaling, the voltage is scaled according to the equation: $V = 0{,}12 \times f + 0{,}5$. As we show later in this paper, our methodology does not depend on the exact relationship between *V-f* (this relationship is simply an input to our models). Finally, we assume that the processor's effective capacity (C_{load}) remains constant across different *V/f* settings.

Static Power Consumption. In this work, we consider only dynamic power/energy as our target for optimization. However, our methodology can be easily extended to include leakage power; typically known during manufacturing for a range of *V,f* settings.

On-chip and Off-chip Voltage Regulators. Traditionally, the full promise of the DVFS has been hindered by slow off-chip voltage regulators. The time overhead to switch the supply voltage using off-chip voltage regulators is in the order of microseconds and is also accompanied by a significant energy overhead [12]. Recently, significant work has been performed on integrating voltage regulators on-chip. On-chip voltage regulators offer fast voltage transitions at nanosecond time-scales and are much smaller in size compared to the off-chip voltage regulators. Unfortunately, their benefits are hampered by lower energy-conversion efficiencies. Thus, there is a trade-off between the size of the voltage regulators, the transition time they offer, and their conversion efficiencies. This trade-off is explored by Kim et. al. [12] in the context of a CMP system where per-core DVFS (using on-chip regulators) as well as chip-level DVFS (using off-chip regulators) were utilized.

Our methodology does not depend on the time-scale in which DVFS decisions are taken. The two analytical models presented in the next section are able to drive informed DVFS decisions by

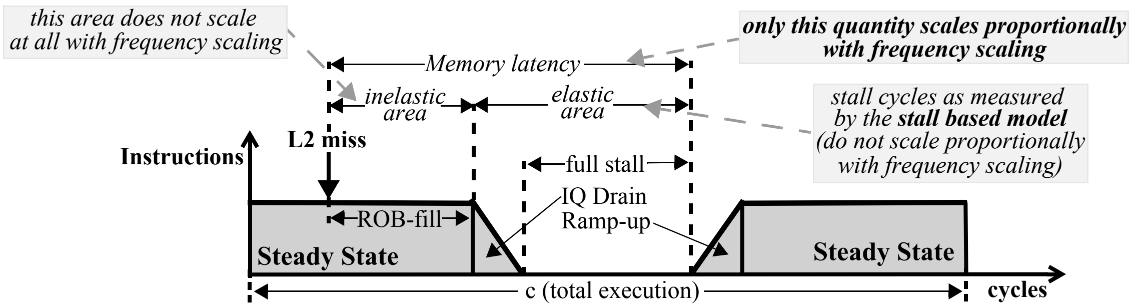
this area does not scale at all with frequency scaling
only this quantity scales proportionally with frequency scaling
Memory latency
inelastic area
elastic area
stall cycles as measured by the stall based model (do not scale proportionally with frequency scaling)
L2 miss
Instructions
full stall
IQ Drain
Ramp-up
ROB-fill
Steady State
Steady State
c (total execution)
cycles

Figure 2. Useful instructions issued per cycle in the case of an isolated L2 load miss.

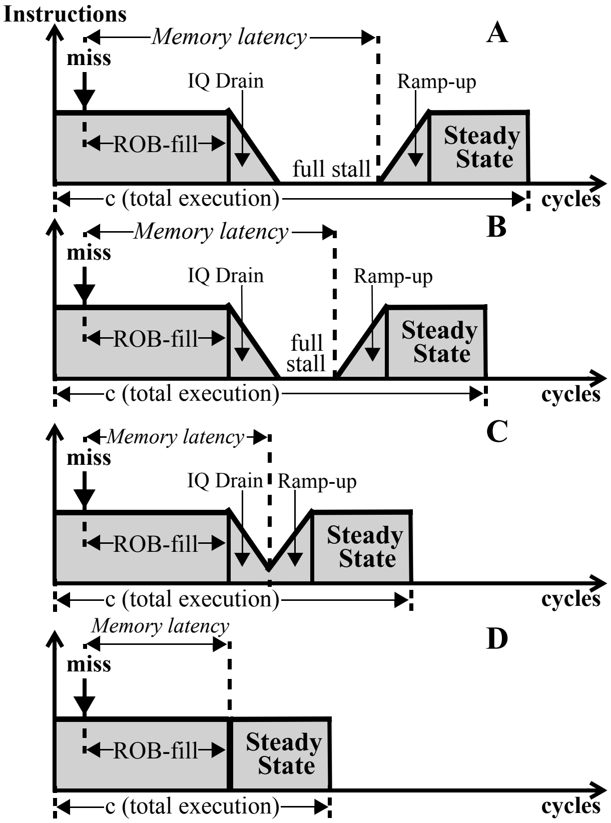

Figure 3. Useful instructions issued per cycle: case studies under different frequencies.

exploring the slack due to long-latency, off-chip, memory operations, but over user-selected portions of the program. Thus, either fine-grained DVFS policies, using on-chip voltage regulators (at the micro-architectural level), or coarse-grain DVFS policies, using off-chip voltage regulators (at the OS level), can be orchestrated using these models.

4. INTERVAL-BASED MODELS

Interval-based models, introduced by Karkhanis and Smith [10] and later refined by Eyerman et. al. [6], break the execution of a program to intervals. Steady-state intervals —where the issue rate is constrained by the width of the machine and the program's ILP— are punctuated by miss-intervals which introduce stall cycles in the machine. Miss-intervals are due to branch mispredictions, L1 instruction and data misses, and L2-misses. By far, the L2 miss-events are the most destructive for performance and the ones bound to introduce the most stalls.

The contribution of our work in relation to previous models is twofold: 1) we use very simple models that are geared towards easy input capture at runtime; 2) we use them to model the effects of DVFS on performance and energy: performance-wise *we model frequency scaling as a change in the memory latency* (measured in clock cycles) and energy-wise we account for both frequency scaling and voltage scaling.

The premise of our models is simple: *since DVFS only affects memory latency in cycles, the only miss-intervals that are affected from frequency scaling are the ones due to the L2-misses (from another point of view, the core, the L1 and the L2 cache are scaled as a single quantity during DVFS management).* The duration of no other miss-intervals (i.e., branch misprediction, L1 miss-intervals) scale, ***in terms of clock cycles***, as the frequency changes. Thus, we only need to account for the L2 miss-intervals. We discard any other miss-event since our goal is to model the ***relative*** *performance/energy changes brought by DVFS.*

The main issue that we need to address in order to derive practical models is how to account for the cycles which are related to an L2-miss (L2-stalls). We present two models that are based on successively more accurate approximations of the L2-stalls. The stall-based model is a rough approximation that assumes that the number of stall cycles due to an L2-miss is equal to the memory latency thus proportional to frequency. This is not an accurate assumption since this way the amount of useful work gets done under an L2-miss is ignored (the stall cycles equal memory latency minus the cycles needed until the head of the ROB will be occupied by the L2-miss). *In other words, the cycles in which the processor is blocked due to an L2-miss do not scale proportionally with frequency*. The second model, called miss-based model, is a refinement of the first in that it ackowledges that the number of stall cycles is not proportional to frequency; *the whole miss-interval scales proportionally with frequency since this interval equals memory latency.* Despite their approximations, these models prove to be valuable and accurate for modeling DVFS; because of their approximations, the run-time information these models need is exceedingly easy to capture.

Figure 2 shows a typical L2 miss-interval of an isolated miss. The y-axis represents the useful instructions issued and the x-axis represents core cycles. Upon the occurrence of a performance critical load L2-miss, the processor continues to issue instructions at the steady-state rate for a few cycles until the ROB fills up (ROB-fill). When the miss reaches the ROB head, no more instructions can enter the issue window and the issue rate starts to drop. From this point on we have stall cycles in the processor.

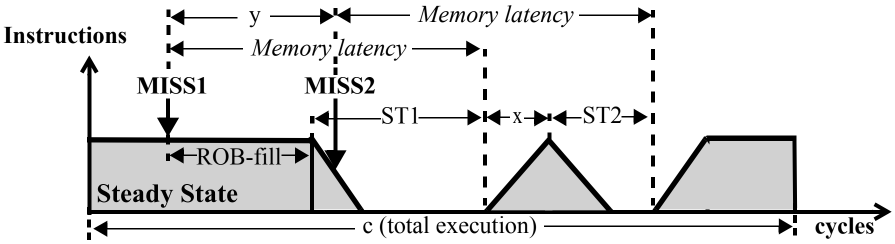

Figure 4. Useful instructions issued per cycle in the case of overlapping L2 load misses.

After a few cycles more the issue rate drops to zero —no further instruction can be issued because all the remaining instructions are dependent on the L2-miss which is still outstanding. When the requested data arrive from main memory (after Mem_{lat} cycles), the miss retires, new instructions can enter the issue window and the issue rate ramps-up until it reaches again the steady-state rate.

To reason about the effects of frequency scaling we must understand how the miss-interval is affected when varying memory latency in cycles. From Figure 2 we can discern four areas in the miss-interval (starting at the L2-miss and ending when the issue rate reaches the steady state again): ROB-fill, independent instruction drain, full stall, and ramp-up (Figure 3.A). These areas are affected differently as the memory latency is reduced. The area that shrinks immediately is the *full stall* (Figure 3.B). When the memory latency drops to the point where the full stall area is eliminated then the drain and ramp-up areas start to shrink simultaneously (Figure 3.C). Finally, when the memory latency becomes as low as the ROB-fill time then we have eliminated all possible stalls due to the miss (Figure 3.D); further reducing memory latency cannot benefit architectural performance (IPC).

We say that the ROB-fill is *inelastic* (Figure 2) to frequency scaling since it does not change with memory latency. In contrast, the drain and full stall areas are *elastic* to frequency scaling, that is, change in relation to memory latency (but not proportionally; *the only quantity that changes proportionally to frequency is the memory latency*). We can safely ignore the ramp-up since it only changes in conjunction with *drain* (when the one piggy-backs the other) and this happens only for their duration.

4.1 Stall-based Model

In the simplest model we assume that ROB-fill is negligible and therefore the bulk of the L2 miss-interval is elastic to frequency scaling. As can be seen from Figure 2:

$$Mem_{lat} = Stall_{cycles} + ROB_{fill}$$

Assuming that ROB-fill is negligible:.

$$Stall_{cycles} \approx Mem_{lat} \quad (1)$$

Up to now, we have been concerned with the case of an isolated L2-miss. When multiple L2-misses overlap (Figure 4), the issue rate begins to rise after the first miss has serviced but drops again when the second miss reaches the ROB head. When the data of the second miss arrive from the main memory, the issue rate rises until it reaches the steady-state rate. In Figure 4 we assume that the second miss occurs *y* cycles after the occurance of the first miss and also that the second miss reaches the ROB head *x* cycles after the first miss has been serviced. As a result, two separate stall intervals appear: *ST1* and *ST2*. Observe that:

$$ST1 + ST2 = y + Mem_{lat} - ROB_{fill} - x$$

Again if we disregard the term $y\text{-}ROB_{fill}\text{-}x$ we conclude that:

$$ST1 + ST2 \approx Mem_{lat} \quad (2)$$

In the stall-based model we assume that (i) the stalls generated by an isolated miss and (ii) the sum of stalls generated by overlapping misses are both approximately equal to the memory latency (in cycles). *Because memory latency is proportional to core frequency, these quantities are approximately proportional to frequency as well.* Consequently, the total number of stall cycles is approximately proportional to frequency, while the total number of non-stall cycles (steady state) is independent of frequency (measured in cycles).

Let *c* be the total cycles of a program execution and *ST* be the total number of stall cycles in max frequency f_{max}. In core frequency f_{max}/k the total number of cycles would be:

$$c_{new} = c - ST + \frac{ST}{k} \quad (3)$$

If the clock period under frequency f_{max} is T_{fmax}, then under frequency f_{max}/k the clock period is $k \times T_{fmax}$. The execution time under frequency f_{max}/k is:

$$t_{new} = c_{new} \times kT_{fmax} = (c \times k - ST \times k + ST) \times T_{fmax} \quad (4)$$

Using equation (4) we are able to predict the execution time under different frequencies by counting the total cycles and the stall cycles under frequency f_{max} (two counters and some comparison logic are the only hardware needed to capture the required functionality). On the other side, if the starting point is a frequency *f* where $f = f_{max}/l$, we can predict the corresponding values for the max frequency $c_{fmax} = c - ST + ST \times l$ and $ST_{fmax} = ST \times l$ and then use in equation 4 c_{fmax} and ST_{fmax} instead of *c* and *ST*.

The advantage of this model is that only in-core information is used to predict performance under various frequencies. However, the model assumes that ROB-fill is negligible, which is a source of error especially in benchmarks characterized by little dependence between instructions and thus large ROB-fill time.

4.2 Miss-based Model

While in the stall-based model we assume that the number of stall cycles is approximately equivalent to the memory latency for both isolated and overlapping misses, in the second model we recognize that there is an *inelastic area* in the miss-intervals (the ROB-fill time) that does not scale with frequency (in cycles). While we do not attempt to model or measure this directly, we model its implications, especially with respect to overlapping misses. *In the second model we assume that the stall cycles of only the first miss of a group of overlapping misses or the stall cycles of an isolated miss should be taken into account for DVFS.*

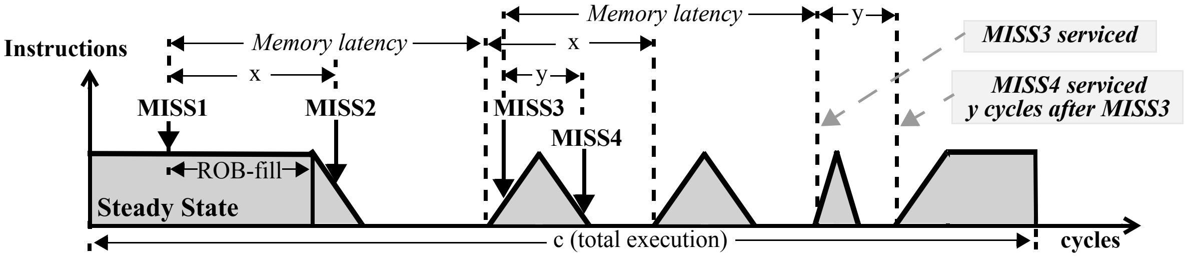

Figure 5. A complex policy indicating the category of misses that have to be counted by the miss-based model.

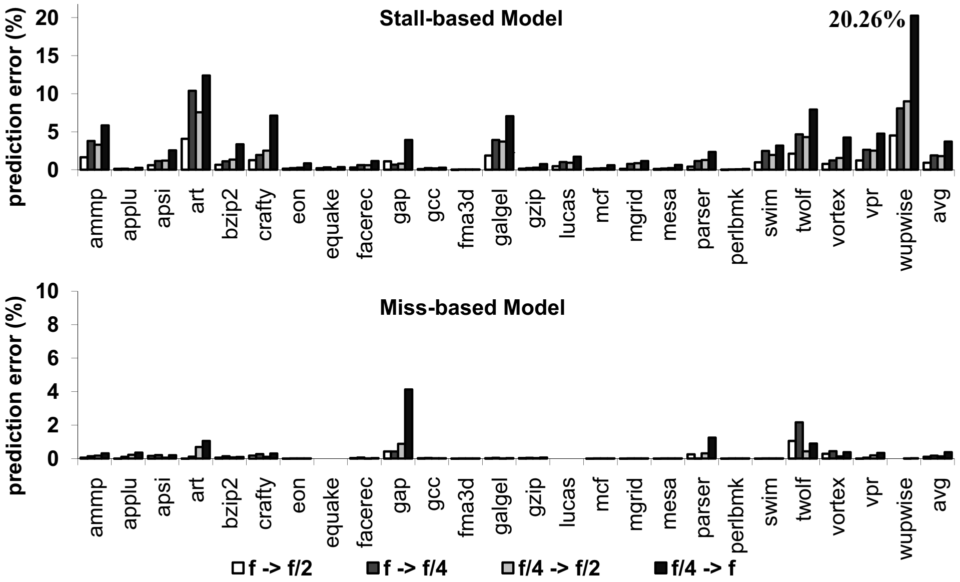

Figure 6. Error for predicting the execution time using the stall-based model (top) and the miss-based model (bottom).

We consider this approach much more viable as we only have to count misses that do not happen in the presence of other misses. Since this can be easily determined even on the memory system side, we call the second model *miss-based*. The information needed can be easily derived from the miss-handling registers (MSHRs) present in most modern processors.

Let us investigate the following scenario shown in Figure 5. A miss occurs and before the issue rate drops to zero a second (overlapping) miss also occurs. The issue rate then drops to zero. After the completion of the first miss, the issue rate rises and then drops again because of the forwarding of the second miss to the ROB head. A third and a fourth miss occur as the issue rate rises and falls. When the first miss occurs, no other miss is in progress. This miss will be serviced after Mem_{lat} cycles. Any miss that occurs x cycles after that miss ($x<Mem_{lat}$) will be serviced x cycles after the first miss, *no matter what the frequency is*. This is due to the *inelasticity* of x with respect to memory latency scaling. This means that only the miss-interval of the first miss scale proportionally to frequency, while the additional stall cycles for the completion of the second miss remain intact. If a miss occurs after the first miss has completed, its service time (Mem_{lat}) will be scaled proportionally to frequency. Similarly, any miss that occurs y cycles after MISS3 ($y<Mem_{lat}$) will finish y cycles after MISS3 has finished *irrespective of the frequency*.

Under the above scenario, only time intervals that are Mem_{lat} long scale proportionally to frequency. All other intervals remain the same when the frequency changes. This means that only MISS1 and MISS3 should be taken into account for DVFS. This leads us to the following method for predicting the execution cycles for a program:

- a miss counter is used to account for the intervals that will be scaled proportionally to frequency.
- the counter counts L2-misses that occur at least Mem_{lat} cycles after the last L2-miss that was taken into account.

Assume now that we counted m misses, the total number of cycles is c and we attempt a prediction for frequency f/k (which means the latency of the memory is Mem_{lat}/k cycles), the total cycles under the new frequency are predicted to be:

$$c_{new} = c - m \times Mem_{lat} + m \times \frac{Mem_{lat}}{k} \qquad (5)$$

If the initial frequency was f_{max} and the corresponding clock period was T_{fmax}, in frequency f_{max}/k the period will be $k\text{x}T_{fmax}$. The execution time is then predicted to be:

$$t_{new} = \left(c - m \times Mem_{lat} + m \times \frac{Mem_{lat}}{k}\right) \times kT_{fmax}$$

4.3 Evaluation

Figure 6 quantify the effectiveness of our models in predicting the execution time across a wide range of frequencies. Both graphs show the resulting absolute error in predicting the execution time for different frequencies. There are four bars per benchmark. The two leftmost bars show the error for predicting the execution time at $f_{max}/2$ and $f_{max}/4$ frequency from the maximum available frequency (f_{max}), while the two rightmost bars represent the reverse scenario (going from the $f_{max}/4$ to $f_{max}/2$ and f_{max}).

The miss-based model (bottom graph) is far more accurate than the stall-based model (top graph). The miss-based model yields prediction errors less than 0.2% on average and up to 4.13% across all the aforementioned frequency transitions. This is a result of correctly taking into account the ROB-fill area. However, the stall-based model is also fairly accurate with an average error of 2.1% (for all frequency transitions). This is remarkable on its own for such a simple model, especially in light of the performance prediction across such large frequency swings. Errors, of course, increase with larger frequency transitions (e.g., from f_{max} to $f_{max}/4$). This is not an issue for the miss-based model where even the largest frequency swing does not increase the error by more than 2.33%. On the other hand in the stall-based model, 11.26% additional error is experienced in *wupwise*, 6.31% in *twolf*, and 4.62% in *crafty*. Those benchmarks are characterized by small dependency chains, so the effect of ignoring the ROB-fill time is more pronounced.

4.4 Assumptions

To keep our models simple, we have made three assumptions. We have verified that these assumptions have a marginal impact in the accuracy of our models since they can only be false on rare occasions.

First, according to [6], upon a load L2-miss, the processor continues to issue instructions for a few cycles until: (i) the ROB fills up (the case that we consider in this work), (ii) the instruction queue fills-up with instructions that are dependent on the L2-miss, and (iii) available rename registers are exhausted. We find that the two latter cases rarely occur (as pointed out in [6]).

Second, we assume that the off-chip store misses have a negligible impact on the system performance (thus are ignored by our models). This is equivalent to assuming that the processor's store buffers never get exhausted, which almost always holds true.

Third, we assume that the access latency of the main memory is constant, even though in reality it might oscillate due to bus contention issues. The hardware cost of calculating the actual access latency of the memory is trivial, but still we believe that a constant value is a good enough approximation.

5. IMPLEMENTING INFORMED DVFS POLICIES

The presented methodology allows us to estimate the execution time of an application —compute or memory bound— under different DVFS settings, which in turn can be used to estimate energy. In this section, we show how our models can be utilized in order to implement various energy-saving policies. We consider the three following policies in this work: EDP and ED^{2}P minimization (both EDP and ED^{2}P are lower-is-better-metrics [1]), and energy minimization within specific performance constraints i.e. assign a specific maximum penalty (d) in execution time trying at the same time to reduce energy to the extent possible. The latter policy is very interesting since it enables new opportunities for our models. For example, in memory-bound applications it is possible to reduce the core frequency (exploiting the slack provided by the long-latency off-chip accesses), until the application becomes compute-bound. If we carefully apply this policy, a small penalty in execution time will be experienced. In the rest of this section, we show how our models can be used to implement those informed DVFS policies.

Assuming a fully clock-gated processor[1], the energy consumed by the core is proportional to the square of voltage: $E=A\text{x}V^2$ (for a given number of executed instructions). In this work, we consider a linear relationship between voltage and frequency: $V=af+b$, where a,b are technology-dependent parameters. In addition, we set f_{max} as the maximum available core clock frequency, T_{fmax} the corresponding clock period, c the total number of cycles needed for the program execution and ST as the total number of stall cycles. Using those parameters, the frequency for the optimal EDP, ED^{2}P as well as the frequency in which the execution time will suffer a specific penalty can be predicted as follows.

Stall-based Model. According to equation (4) at frequency $f=f_{max}/k$, the execution time is predicted to be:

$$t_{new} = (c \times k - ST \times k + ST) \times T_{fmax}$$

Then:

$$EDP = A(af+b)^2 \times t_{new}$$

Since our target is to find the global minimum EDP (for a frequency scaling factor k), we differentiate the above expression and after doing the necessary substitutions (and simplifications) we derive the following expression:

$$(bk + af_{max})\left((c-ST)b \times k^2 - (c-ST)a \times f_{max} \times k - 2af_{max} \times ST\right) = 0$$

Given that the technology-dependent parameters a, b are always positive quantities, the solution $k=-(a\text{x}f_{max})/b$ is rejected. Thus, the optimal frequency scaling factor is given by the second term of the above equation, which is a typical quadratic equation. The roots of the equation are:

$$k = \frac{\frac{a \times f_{max}}{b} \mp \sqrt{\Delta}}{2} \quad \text{where:}$$

$$\Delta = \frac{a^2 \times f_{max}^2 (c-ST) + 8 \times a \times b \times f_{max} \times ST}{(c-ST)b^2}$$

The root that results from using the negative value of Δ (discriminant) can be easily proven to be always less than or equal to zero, so it is rejected, too. This leaves at our disposal the following optimal scaling factor k which minimizes EDP in the stall-based model:

$$k = \frac{\frac{a \times f_{max}}{b} + \sqrt{\Delta}}{2}$$

Of course, if the optimal frequency ratio is less than 1, the frequency $f = f_{max}/k$ is greater than f_{max}, which is not applicable in our case (although this is actually the optimal EDP point). In this case we choose f_{max} as the optimal frequency[2].

[1] Our methodology can be easily extended to include non clock-gated processors. In this case, the resulting energy under DVFS is also proportional to the frequency and the execution time.

[2] This is not a limitation of our model. It is a corner case which appears when the frequency which minimizes the corresponding metric (EDP or ED^{2}P) is greater than the max frequency f_{max}.

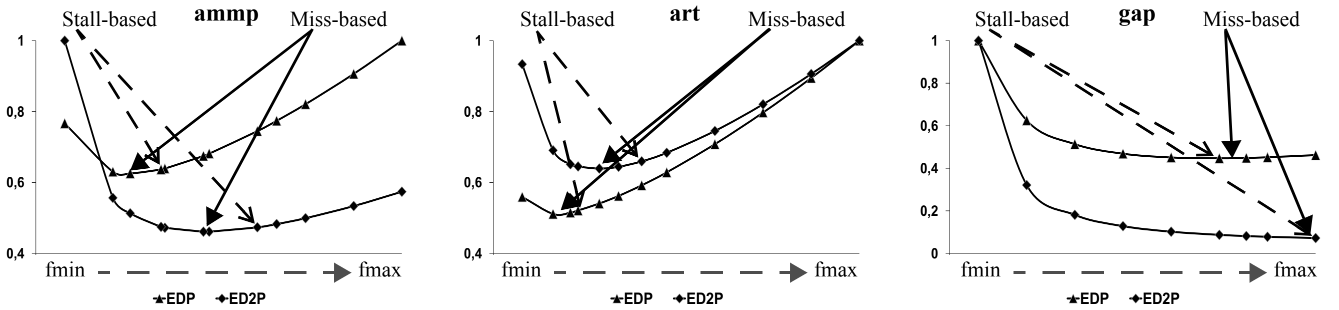

Figure 7. Normalized EDP-ED^{2}P curves for 3 benchmarks. Our models manage to accurately predict optimal frequencies.

In a similar way, we calculate the frequency scaling factor that yields the optimal ED^{2}P:

$$k = \sqrt{\frac{a \times f_{max} \times ST}{b(c - ST)}}$$

Again if k is less than 1, we automatically revert to f_{max}.

Finally, in order to model the last policy, which aims to bound the execution time of an application within a performance penalty *of at most d*, we define as $t_{fmax}=c\times T_{fmax}$ the execution time when the core operates at its nominal frequency and t_{new} as the execution time in the new frequency f_{new} which is calculated using equation (4). As a result, the performance penalty is defined as follows:

$$d = \frac{t_{new} - t_{fmax}}{t_{fmax}} = \frac{(c \times k - ST \times k + ST)T_{fmax} - c \times T_{fmax}}{c \times T_{fmax}} \Rightarrow$$

$$(c - ST)k + ST = (d+1)c \Rightarrow k = \frac{(d+1)c - ST}{c - ST}$$

Miss-based model. A closer inspection of the equations (4) and (6) —the equations which calculate the predicted execution time for the new frequency for the stall-based and miss-based models respectively— reveals that equation (6) can be easily derived from equation (4) by substituting ST with $m\times Mem_{lat}$ (where m is the number of misses accounted in the miss-based model and Mem_{lat} is the memory latency in cycles). By applying this substitution in the equations used to calculate the targeted DVFS policies in the stall-based model, we obtain the corresponding equations for the miss-based model. As a result, the appropriate frequency scaling factor k for the optimal EDP in the miss-based model is:

$$k = \frac{\frac{a \times f_{max}}{b} + \sqrt{\Delta}}{2} \text{ where:}$$

$$\Delta = \frac{a^2 \times f_{max}^2(c - m \times Mem_{lat}) + 8ab \times m \times Mem_{lat} \times f_{max}}{(c - m \times Mem_{lat})b^2}$$

Respectively, the k factor for the optimal ED^{2}P is:

$$k = \sqrt{\frac{a \times m \times Mem_{lat} \times f_{max}}{b(c - m \times Mem_{lat})}}$$

while the predicted scaling factor k for the policy which aims to bound the performance penalty within a constant d is:

$$k = \frac{(d+1)c - m \times Mem_{lat}}{c - m \times Mem_{lat}}$$

Finally, Figure 7 shows the effectiveness of our models in predicting the optimal EDP and ED^{2}P for three representative benchmarks (*ammp*, *art* —memory intensive benchmarks,— and *gap* —compute intensive). As we can see, our models accurately manage to predict the frequencies in which the EDP and ED^{2}P metrics are minimized with the miss-based model being more accurate. Our experimental findings reveal that across all the SPEC2K benchmarks our models manage to predict EDP and ED^{2}P frequencies within 1.5% of the optimal with the deviation in ED^{2}P to be marginally bigger.

6. RUN-TIME DVFS MANAGEMENT

In this section, we show how our models can be used to enforce various informed DVFS policies at run-time. The presented models afford us to directly calculate the effects on performance (performance loss) and on energy (energy benefits) of any possible DVFS setting. Having this information it is straightforward to apply any DVFS policy of interest. In this work, we showcase three different policies: "operate at optimal EDP," "operate at optimal ED^{2}P", and "reduce ED^{2}P within specific performance constraints." Recall, that our approach requires minimal hardware cost which makes it suitable for run-time optimizations.

The methodology we follow for the run-time optimizations is: we divide program execution into windows and we start executing the program at the highest available *V/f* setting. During the first window, we collect the statistics needed by our models. These statistics are used to predict the *V/f* settings for the next window. In other words, we use a "last value" predictor to set the next window's *V/f* point. While more complicated predictors can be employed (such as the Global Phase History Table [8]), we found that they produce marginally better predictions. On the other hand, the window size is a critical parameter which really affects the quality of our predictions. Our experiments reveal that relatively large windows are needed in order to obtain energy/power benefits. Therefore, we split up the execution into windows of 150M cycles. All the presented results are normalized to the base case of an un-managed processor (running at the V_{max}/f_{max} point).

Figure 8 illustrates the results for the first two DVFS policies (operate at optimal EDP and ED^{2}P respectively) that we consider in this paper and for all SPEC2K. In each graph, there are two bars per benchmark. The first bar depicts the resulting benefits when the stall-based model drives the DVFS decisions, while the second bar shows the results for the miss-based model. As we can see, both models manage to achieve significant improvements. In terms of EDP (top graph), the stall-based model achieves a reduction in EDP (both EDP and ED^{2}P are lower-is-better-metrics [1]) by 17% on average (up to 47%), while the miss-based model reduces EDP

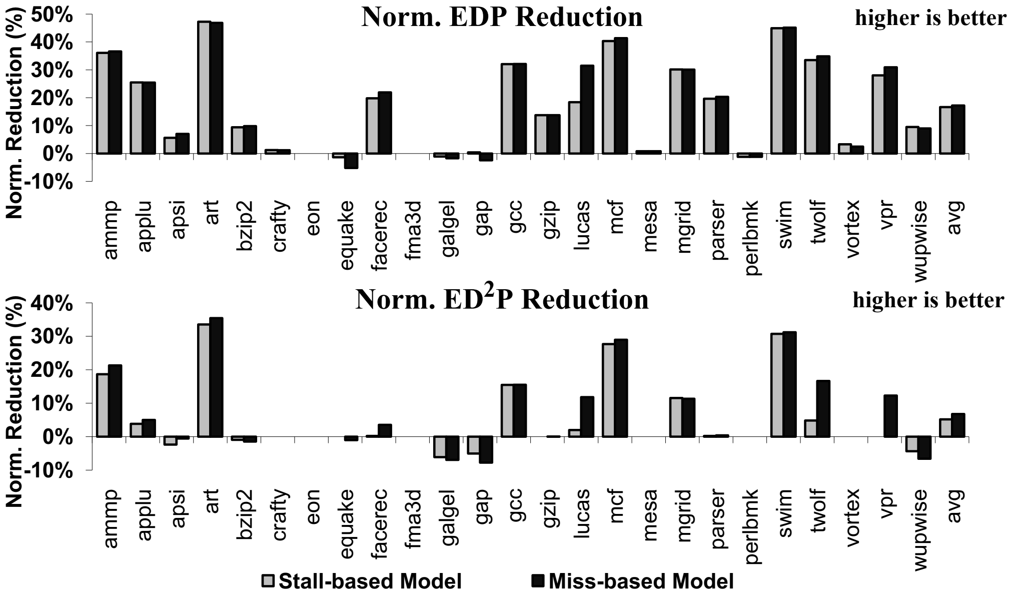

Figure 8. Normalized EDP (top graph) and ED^{2}P (bottom graph) reduction achieved by the stall-based and the miss-based model.

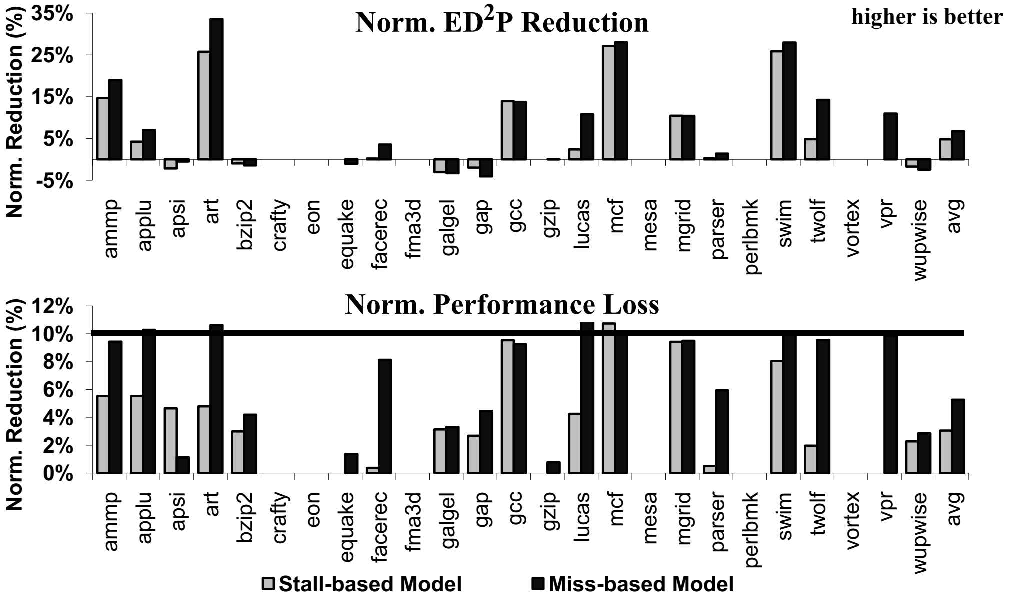

Figure 9. Normalized ED^{2}P reduction (top graph) and Performance loss (bottom graph) for the third policy.

by 18% on average (up to 47%). Of course the results are more pronounced for the memory-intensive workloads (*ammp*, *applu*, *art*, *mcf*, *swim*, *gcc, twolf*, and *vpr*), in contrast to the compute-intensive workloads where our mechanism correctly decides to operate at the maximum available frequency (*crafty*, *eon*, *equake*, f*ma3d*, *galgel, gap*, *mesa*, *perlbmk*, and *vortex*). While the miss-based model shows only a marginal improvement (on average) in terms of EDP compared to stall-based model, there are three cases where it manages to further increase the potential benefits (*lucas*, *vpr*, and *facerec*). Finally, across all benchmarks, four experience a marginal increase in EDP (5% in *equake*, 2% in *gap*, 1% in *galgel*, and 1% in *perlbmk*). This is the result of using a simple "last value" predictor; it is also related to the size of the windows we choose. In other words, our naive predictor assumes that the next window exhibits the same characteristics as the current window. Our models correctly manage to predict the optimal DVFS settings according to the characteristics of the current window but our predictor fails to capture differences in the next window. However, the maximum EDP increase is less than 5%.

In terms of ED^{2}P (Figure 8, bottom graph), the miss-based model manages to further increase its distance from the stall-based model. This is because in ED^{2}P, the performance penalty incurred by the less accurate stall-based model is now more pronounced. Both models show substantial ED^{2}P savings: 5% on average (up to 34%) for the stall-based model and 7% on average (up to 36%) for the miss-based model. Considering the three cases mentioned above (*lucas*, *vpr*, and *facerec*), the stall-based model hardly produces ED^{2}P savings, while the miss-based model offers a 12% reduction in *lucas,* 12% in *vpr,* and 5% in *facerec.*

Finally, the real strength of the miss-based model gets exposed when the last policy we consider in this paper is employed. This policy aims to maximize the resulting energy savings in terms of ED^2P under specific performance constraints (maximum performance penalty of d). These constraints can be set either by the OS or by the application itself. We assume d=10% which means "try to minimize the ED^2P without increasing execution time by more than 10% (compared to an unmanaged processor)." Figure 9 depicts the results for this policy in terms of ED^2P (top graph) and execution time penalty (bottom graph). As the bottom graph shows, our models successfully manage to drive DVFS management in such a way so that all applications obey the 10% execution-time increase restriction (less than 1% error). In cases, where the execution time increase is less than 10%, the minimum ED^2P simply appear at lower performance penalties. Under this policy, the miss-based model manages to lower the resulting ED^2P by 7% on average (up to 34%), while the stall-based model 5% on average (up to 27%).

7. CONCLUSIONS

This paper introduces two simple analytical models that are able to drive run-time DVFS decisions for superscalar out of order processors. The proposed models are able to predict with great success the performance and energy impact of DVFS across a wide range of *V/f* points. Both models require minimal input. The first model called stall-based model is fed by an approximation of the stall cycles experienced by the processor due to the performance-critical off-chip load accesses. The second model, called miss-based model, improves on this approximation using as input the occupancy of the L2's miss-handling registers (MSHRs). Our experimental results using the SPEC2K suite show that both models are very accurate in predicting the performance (and energy) for any target *V/f* setting yielding average errors of 2.1% (stall-based model) and 0.2% (miss-based model). The highly accurate predictions provided by our models can be used in various run-time power optimizations. In this paper, we showcase three case studies: operate at optimal EDP and ED^2P and minimize ED^2P within specific performance constrains. Our simulations show that in all case studies we consider the proposed models efficiently manage to orchestrate the necessary DVFS adjustments resulting in significant power/energy benefits.

8. ACKNOWLEDGEMENTS

This work is supported by the EU-FP6 Integrated Project, Scalable computer ARChitecture, Contract No. 27648 and the EU-FP7 ICT Projects, "A Highly Efficient Adaptive multi-Processor framework," Contract No. 247615, and "Embedded Reconfigurable Architecture," Contract No. 249059.

9. REFERENCES

[1] D. Brooks et. al. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 2000.

[2] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. *Proc. of the International Symposium on Computer Architecture*, 2000.

[3] Q. Cai et. al. Meeting Points: Using Thread Criticality to Adapt Multicore Hardware to Parallel Regions. *Proc. of the International Conference on Parallel Architectures and Compilation Techniques*, 2008.

[4] J. Donald and M.Martonosi. Techniques for multicore thermal management: classification and new exploration. *Proc. of the International Symposium on Computer Architecture*, 2006.

[5] S. Eyerman and L. Eeckhout. A Counter Architecture for Online DVFS Profitability Estimation. *IEEE Transactions on Computers*, Mar. 2010.

[6] S. Eyerman, L. Eeckhout, T. Karkhanis, and J.E. Smith. A mechanistic performance model for superscalar out-of-order processors. *ACM Transactions on Computer Systems*, 2010.

[7] C. Isci et. al. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. *Proc. of the International Symposium on Microarchitecture*, 2006.

[8] C. Isci, G. Contreras, and M. Martonosi. Live, runtime phase monitoring and prediction on real systems with application to dynamic power management. *Proc. of the Annual International Symposium on Microarchitecture*, 2006.

[9] C. Isci and M.Martonosi. Identifying program power phase behavior using power vectors. *Proc of the IEEE International Workshop on Workload Characterization*, 2003.

[10] T. Karkhanis and J.E. Smith. A first-order superscalar processor model. *Proc. of the Annual International Symposium on Computer Architecture*, 2004.

[11] S. Kaxiras and M. Martonosi. Computer Architecture Techniques for Power-Efficiency. *Morgan & Claypool Publishers*, 2008.

[12] W. Kim, M.S. Gupta, G.Y. Wei, and D. Brooks. System level analysis of fast, per-core DVFS using on-chip switching regulators. *Proc. of the International Symposium on High-Performance Computer Architecture*, 2008.

[13] H. Li, C.Y. Cher, T.N. Vijaykumar, and K. Roy. VSV: L2-miss-driven variable supply-voltage scaling for low power. *Proc. of the Annual International Symposium on Microarchitecture*, 2003.

[14] G. Maglis, et al. Profile-based dynamic power voltage and frequency scaling for a multiple clock domain processor. *Proc. of the International Conference on Computer Architecture*, 2003.

[15] V. Spiliopoulos. DVFS for processor energy optimization. *Diploma Thesis, ECE Dept., University of Patras,* Sept. 2009.

[16] J. Srinivasan, S. V. Adve, P. Bose, and J. Rivers. The case for lifetime reliability-aware microprocessors. *Proc. of the International Symposium on Computer Architecture*, 2004.

[17] R. Teodorescu and J. Torrellas. Variation-aware application scheduling and power management for chip multiprocessors. *Proc. of the International Symposium on Computer Architecture*, 2008.

[18] Q. Wu et. al. A dynamic compilation framework for controlling microprocessor energy and performance. *Proc. of the IEEE/ACM International Symposium on Microarchitecture*, 2005.

[19] F. Xie, M. Martonosi, and S. Malik. Compile-time dynamic voltage scaling settings: opportunities and limits. *Proc of the Conference on Programming Language Design and Implementation*, 2003.