Brief Contributions

A Counter Architecture for Online DVFS Profitability Estimation

Stijn Eyerman and Lieven Eeckhout, *Member*, *IEEE*

Abstract—Dynamic voltage and frequency scaling (DVFS) is a well known and effective technique for reducing power consumption in modern microprocessors. An important concern though is to estimate its profitability in terms of performance and energy. Current DVFS profitability estimation approaches, however, lack accuracy or incur runtime performance and/or energy overhead. This paper proposes a counter architecture for online DVFS profitability estimation on superscalar out-of-order processors. The counter architecture teases apart the fraction of the execution time that is susceptible to clock frequency versus the fraction that is insusceptible to clock frequency. By doing so, the counter architecture can accurately estimate the performance and energy consumption at different V/f operating points from a single program execution. The DVFS counter architecture estimates performance, energy consumption, and energy-delay-squared-product (ED^2P) within 0.2, 0.5, and 0.8 percent on average, respectively, over a $4\times$ frequency range. Further, the counter architecture incurs a small hardware cost and is an enabler for online DVFS scheduling both at the intracore as well as at the intercore level in a multicore processor.

Index Terms—Computer systems organization, performance of systems, modeling techniques, modeling of computer architecture, power management.

1 Introduction

ENERGY and power consumption are first-class design concerns for contemporary microprocessors, from low-end embedded systems to high-end high-performance microprocessors. For embedded devices, the focus is on low energy consumption to increase battery time. For high-performance microprocessors, the goal is to maximize system performance within a given power budget.

An effective and widely used power reduction technique is *Dynamic Voltage and Frequency Scaling (DVFS)*. DVFS lowers the supply voltage as well as the clock frequency to reduce both dynamic and static power consumption. DVFS is being used in commercial processors across the entire computing range. The applicability of DVFS is not limited to reducing power and energy consumption. It is also effective at addressing timing errors due to process variability; e.g., Razor [7] tunes the supply voltage to minimize error rate and power consumption in the presence of process variability. Other uses of DVFS are dynamic thermal management [2] and dynamic lifetime reliability management [21].

An important delimiter to DVFS though is that there exists no accurate and practical way for estimating its impact on performance and energy consumption. Existing DVFS profitability estimation approaches can be categorized in four classes:

- A simple approach for estimating the performance impact of DVFS is *proportional scaling*, i.e., performance is assumed to scale proportionally with supply voltage and clock frequency (see, for example, [11], [12]). This is a practical approach because the performance under DVFS can be computed as the execution time at a nominal clock frequency multiplied by a scaling factor. Proportional scaling yields accurate estimates for compute-bound applications, but incurs (severe) errors for memory-bound applications because off-chip memory access latencies do not scale with processor clock frequency.
- *Linear scaling* states that performance is a linear function of clock frequency (see, for example, [25]). The slope of this linear function is a result of the application behavior. If the application is compute-bound, the slope will be proportional to clock frequency. If on the other hand, the application is memory-bound, the slope is (almost) flat, i.e., performance is barely affected by processor clock frequency. Although linear scaling yields accurate DVFS performance estimates for both compute-bound and memory-bound applications, it introduces runtime performance and/or energy overhead because it requires two samples at different V/f operating points for computing the linear slope.
- *Estimated linear scaling* eliminates the runtime overhead in linear scaling by estimating the relationship between performance and clock frequency (see, for example, [4], [13], [18], [20], [23], [24]). Estimated linear scaling uses existing hardware performance counters to count the number of off-chip memory accesses, and derives an empirical model to estimate the linear slope as a function of the number of off-chip memory accesses. The limitation of estimated linear scaling is that it does not account for the impact of memory-level parallelism (MLP, or multiple memory accesses overlapping in time) on the nonpipelined fraction of the execution time, and therefore, leads to inaccurate DVFS profitability estimates.
- *Stall cycle counting* counts the number of stall cycles due to off-chip memory accesses as an estimate for the nonpipelined execution time. To the best of our knowledge, stall cycle counting has not been proposed previously for DVFS profitability estimation, although it may be possible to deploy on contemporary microprocessors using existing hardware performance counters. However, stall cycle counting still leads to inaccurate performance predictions because it underestimates the nonpipelined execution time—it disregards the fraction of the nonpipelined off-chip load miss penalty where useful work gets done.

This paper proposes a novel and practical counter architecture that accurately estimates DVFS profitability on superscalar out-of-order processors. The counter architecture divides total execution time into a *pipelined* fraction subject to clock frequency and a *nonpipelined* fraction due to off-chip memory accesses. DVFS profitability is then estimated from these two fractions: the pipelined fraction scales proportionally with clock frequency whereas the nonpipelined fraction does not scale at all; this yields an estimate for the execution time under DVFS, which in its turn is used to estimate energy consumption. The DVFS counter architecture is implemented in hardware and computes the pipelined and nonpipelined fractions online during program execution. The hardware cost is small (70 bits of storage plus a 64-bit incrementer). Using detailed simulations, we show that the proposed counter architecture is accurate: it estimates performance within 0.2 percent on average (and 2.2 percent max error) while scaling clock frequency over a $4\times$ range; energy is estimated with an average error of 0.5 and 0.7 percent for a clock-gated and non-clock-gated processor, respectively. Combining the performance and energy estimates, the counter architecture estimates energy-delay product (EDP) and

• *The authors are with the ELIS Department, Ghent University, Sint-Pietersnieuwstraat 41, B-9000 Gent, Belgium.*
E-mail: {seyerman, leeckhou}@elis.ugent.be.

Manuscript received 16 Dec. 2008; revised 10 Sept. 2009; accepted 21 Dec. 2009; published online 2 Mar. 2010.
Recommended for acceptance by J. Lach.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2008-12-0621.
Digital Object Identifier no. 10.1109/TC.2010.65.

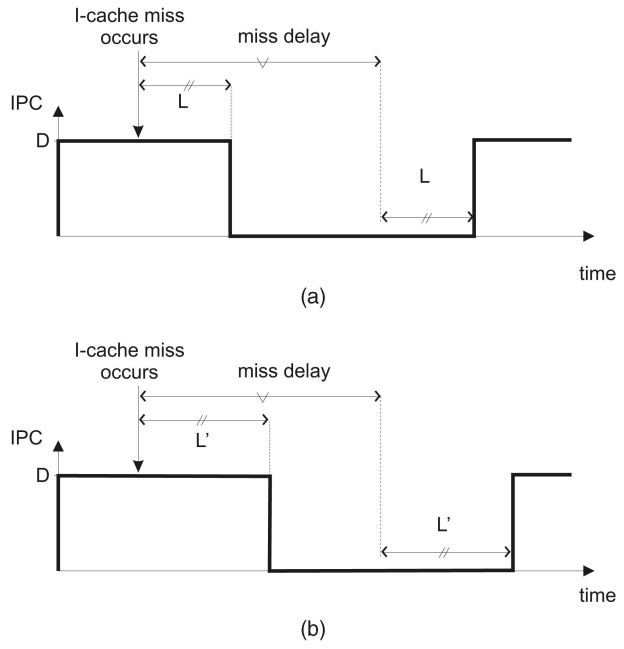

Fig. 1. Off-chip I-cache miss timing: (a) at frequency f, and (b) at frequency $f' < f$.

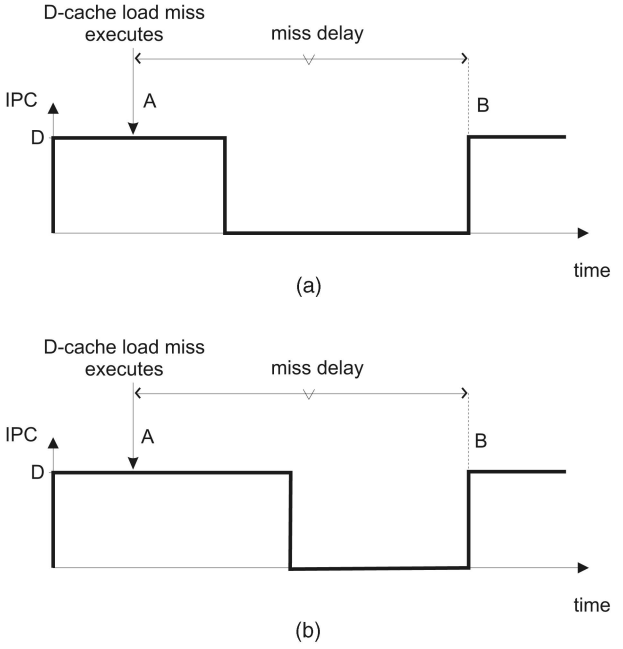

Fig. 2. Timing of an isolated off-chip load miss: (a) at frequency f, and (b) at frequency $f' < f$.

energy-delay-square product (ED^2P) within 0.7 and 0.8 percent on average, respectively, and at most five percent. The counter architecture is substantially more accurate than proportional scaling, estimated linear scaling, and stall cycle counting.

The counter architecture is a key enabler for a number of DVFS-based optimizations. In a single processor core, the counter architecture can be used to estimate whether scaling clock frequency and supply voltage will violate deadlines in soft real-time media applications (e.g., video or audio decoders). In a high-end multicore processor with multiple per-core voltage/clock domains, the counter architecture can be used to trade off performance and power consumption at the chip level by scaling clock frequency and supply voltage at the core level, e.g., maximize chip-level throughput within a given power budget by adjusting per-core clock frequency and supply voltage. The counter architecture does not cause any power budget overshoots—in contrast to proportional scaling—and achieves higher system throughput and shorter turnaround times than estimated linear scaling and stall cycle counting.

2 Counter Architecture

2.1 Performance Model

The key idea behind the proposed DVFS counter architecture is to split up the total execution time into a *pipelined* fraction and a *nonpipelined* fraction. Total execution time T at the nominal clock frequency f_n and supply voltage V_n then equals

$$T(V_n, f_n) = T_{pipelined}(V_n, f_n) + T_{nonpipelined}. \quad (1)$$

At voltage V and frequency f, the execution time can then be estimated as $\tilde{T}$

$$\tilde{T}(V, f) = \frac{T_{pipelined}(V_n, f_n)}{f/f_n} + T_{nonpipelined}, \quad (2)$$

i.e., the pipelined fraction scales proportionally with clock frequency whereas the nonpipelined fraction does not.

The key challenge now is to determine the pipelined and nonpipelined fractions of the total execution time. In this work, we compute the nonpipelined fraction due to off-chip memory accesses; the pipelined fraction then is the complement of the total execution time.

2.2 Estimating the Nonpipelined Fraction

For estimating the nonpipelined fraction of the total execution time ($T_{nonpipelined}$), we focus on the two major off-chip contributors to the total execution time, namely off-chip I-cache misses and off-chip load misses. (TLB misses incur similar overheads, and we therefore treat them collectively with off-chip cache misses.) The reasoning that we develop in the following sections is based on a recently proposed mechanistic performance model, namely interval analysis [9].

2.2.1 Off-chip I-Cache Misses

Fig. 1 shows a schematic drawing of the timing behavior for an off-chip (L2) I-cache miss. The vertical axis shows the processor's dispatch behavior as a function of time on the horizontal axis. Initially, the processor dispatches D instructions every cycle from the front-end pipeline into the reorder buffer and issue queues. At some point, an L2 I-cache miss occurs. Then, it takes L cycles before dispatch stops, with L the number of front-end pipeline stages (i.e., the front-end pipeline depth from instruction fetch to dispatch). In the meanwhile, the L2 I-cache miss is handled (through an off-chip memory access). When the off-chip memory access gets back, the processor will resume fetching instructions, and L cycles later, dispatch will resume. The penalty for an off-chip I-cache miss thus equals the off-chip memory access time, i.e., draining and filling the front-end pipeline offset each other.

In other words, the nonpipelined fraction of the total execution time due to an off-chip I-cache miss can be computed by simply counting the number of cycles the off-chip memory access takes divided by clock frequency.

2.2.2 Off-chip Load Misses

Off-chip (L2) D-cache load misses are more complicated, and we make a distinction between an isolated long-latency load miss and overlapping long-latency load misses. Fig. 2 shows the timing behavior for an isolated off-chip load miss. The L2 cache miss load instruction gets executed and the memory access gets

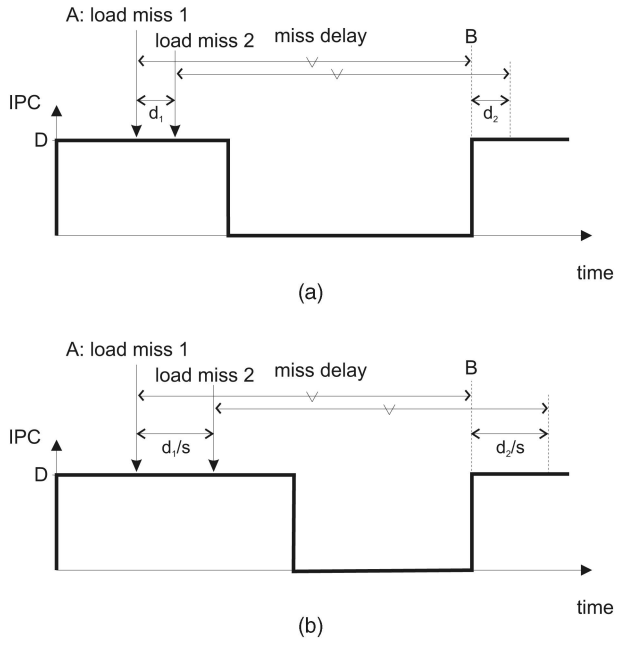

Fig. 3. Timing of overlapping off-chip load misses: (a) at frequency f, and (b) at frequency $f' < f$.

initiated (see point "A" in Fig. 2). Underneath the handling of the off-chip memory access, the processor will continue dispatching instructions until either 1) the reorder buffer completely fills up and the long-latency load blocks the head of the reorder buffer, 2) the issue queues fill up because of instructions that are dependent on the long-latency load, or 3) the number of rename registers gets exhausted. (Karkhanis and Smith [15] found 1) to be the most common case.) Eventually, dispatch ceases for a long period of time. When the data gets back from memory, dispatch resumes (see point "B" in Fig. 2).

The time period where useful work gets done underneath the memory access scales when scaling frequency (see Figs. 2a and 2b), however, it does not affect the nonpipelined fraction due to the memory access. The nonpipelined fraction of the execution time thus is the time between the long-latency load initiating its memory access (point "A" in Fig. 2), and the data getting back from memory (point "B").

Fig. 3 shows the timing behavior for two independent long-latency load misses. At some point while the first long-latency load miss is being serviced, the second miss will occur. Both miss penalties overlap, i.e., MLP [5], [10], [16] gets exposed. To derive the nonpipelined fraction of the execution time, assume now there are d_1 and d_2 time units between both memory accesses being initiated and returning from memory, respectively (see Fig. 3a). Scaling the clock frequency by a factor s, these time deltas d_1 and d_2 will scale proportionally, i.e., the second memory access will be initiated d_1/s time units past the first memory access, and the time delta between both memory accesses returning from memory will be d_2/s time units (see Fig. 3b). The nonpipelined fraction of the execution time for two independent long-latency load misses thus is the time period between the first miss going to memory (point "A") and the first miss returning from memory (point "B"); the time period between the two misses going to memory (i.e., d_1) gets hidden under the nonpipelined miss delay, and the time period between the first miss and second returning from memory (i.e., d_2) is pipelined.

This observation generalizes to an arbitrary number of independent long-latency loads. And thus, the nonpipelined fraction of the execution time due to long-latency load misses is the memory access time for the *first* long-latency load miss in a burst of long-latency load misses.

2.3 A Practical Counter Architecture

Based on these insights, we propose a counter architecture that counts

- the number of cycles that an off-chip I-cache miss accesses memory, and
- the number of cycles that an off-chip load miss accesses memory and this load miss is the first in a burst of load misses. To do so, the counter architecture uses an *Active Counting (AC)* bit that denotes whether the counter is actively counting, and a *First Pending Miss (FPM)* register which denotes the ID of the MSHR entry for the first pending miss in a burst of misses. Initially, the AC bit is set to zero. Upon a load miss, the AC bit is set, an MSHR entry is allocated, the allocated MSHR entry ID is stored in the FPM register, and the counter starts counting. In the meanwhile, other requests may be issued, i.e., MLP is exposed. When the first request (the one stored in the FPM register) returns from memory, the AC bit is reset, the FPM content is cleared, and the counter stops counting. A counting epoch thus starts and stops with the first load miss in a burst of load misses, and can only start when the AC bit is zero.

The counter architecture does not double count cycles where an off-chip I-cache miss is overlapped by off-chip load misses, and vice versa.

The hardware cost for the counter architecture is small. It only requires a counter and an associated incrementer for counting the number of off-chip cycles, an AC bit, and an FPM register (requiring $\lceil log_2(N_{MSHR}) \rceil$ bits with N_{MSHR} the number of MSHR entries). The total hardware cost is limited to 70 bits (assuming 32 MSHRs) plus a 64-bit incrementer.

Note that the counter architecture captures variable memory access latencies naturally, i.e., it does not approximate off-chip memory access latencies by constants but instead counts the number of off-chip memory access cycles. This enables computing variable memory access latencies due to open page hits and misses, memory bank conflicts, memory reference reordering by the memory controller, etc. In addition, it also applies to remote cache accesses in multiprocessor and/or multicore processor architectures due to coherence actions.

2.4 Estimating Energy Consumption

We make a distinction between a fully clock-gated processor versus a processor without clock gating for estimating the impact of DVFS on energy consumption. One can derive the formula for a microprocessor with partial clock gating by making a distinction between the parts of the processor that are fully clock gated versus the parts that are not. In what follows, we assume that the processor features the capability for online measuring the (total) energy consumption $E(V_n, f_n)$ and execution time $T(V_n, f_n)$ at a nominal clock frequency f_n and supply voltage V_n. Execution time can be measured using hardware performance counters that are typically available on modern microprocessors. Energy consumption can be estimated online using the techniques as described by Isci and Martonosi [14], and implemented in the Intel Foxton technology [17]. We also assume that the static power consumption of the microprocessor $P_s(V)$ is known. Static power consumption can be considered as a relatively slowly varying constant and could be measured periodically by halting the processor for a period of time and measure its power consumption—power consumption during halting approximates static power consumption.

2.4.1 Without Clock Gating

In a non-clock-gated processor, the processor consumes dynamic and static power during its entire operation. Energy consumption at frequency f and supply voltage V can thus be estimated as

$$\tilde{E}(V,f) = \left(\frac{E_d(V_n,f_n)}{T(V_n,f_n)} \cdot \frac{V^2 f}{V_n^2 f_n} + P_s(V)\right) \cdot \tilde{T}(V,f), \qquad (3)$$

TABLE 1
Processor Model Assumed in Our Experimental Setup

ROB	128 entries
LSQ	64 entries
processor width	decode, dispatch, issue and commit 4 wide fetch 8 wide
latencies	load (2), mul (3), div (20)
L1 I-cache	32KB 4-way set-assoc, 1 cycle
L1 D-cache	32KB 4-way set-assoc, 1 cycle
L2 cache	unified, 2MB 8-way set-assoc, 9 cycles
main memory	250 cycle access time
branch predictor	hybrid bimodal/gshare predictor
frontend pipeline	5 stages

with $\tilde{T}(V,f)$ the estimated execution time at frequency f and supply voltage V. The first term between brackets in the above formula estimates dynamic power consumption whereas the second term estimates static power consumption.

The nominal dynamic energy consumption in the above formula can be computed as

$$E_d(V_n,f_n) = E(V_n,f_n) - P_s(V_n)T(V_n,f_n), \quad (4)$$

i.e., the total energy consumption minus the static energy consumption at the nominal V/f operating point.

2.4.2 With Clock Gating

In a clock-gated processor, energy consumption can be estimated as

$$\tilde{E}(V,f) = E_d(V_n,f_n)\cdot\frac{V^2}{V_n^2} + P_s(V)\tilde{T}(V,f). \quad (5)$$

The first and second term estimate dynamic and static energy consumption, respectively. The static energy consumption term is easy to understand: static power is consumed during the entire (estimated) execution time $\tilde{T}(V,f)$. The dynamic energy consumption term is slightly more complicated: the intuition is that a clock-gated processor only consumes dynamic power when there is work to be done, and the fraction of time where work is done scales proportionally with frequency under DVFS. In other words, if the processor consumes dynamic power for U time units at the nominal V_n/f_n operating point, the processor will consume dynamic power for $U\cdot f_n/f$ time units at a V/f operating point. Dynamic power consumption is a factor $V^2f/V_n^2f_n$ of the nominal dynamic power consumption. Hence, the dynamic energy consumption at the V/f operating point under clock gating equals $E_d(V_n,f_n)\cdot\frac{U\cdot f_n/f}{U}\cdot\frac{V^2f}{V_n^2f_n} = E_d(V_n,f_n)\cdot\frac{V^2}{V_n^2}$.

2.5 Assumptions

The counter architecture makes a number of simplifying assumptions. This is done on purpose, in order to limit the design complexity of the counter architecture with limited impact on accuracy.

First, the performance model assumes that the amount of work done underneath a nonpipelined memory access does not exceed the memory access time when scaling clock frequency. This is a reasonable assumption because memory access time is typically on the order of several hundreds of processor cycles whereas the amount of work done by the processor underneath a memory access typically takes no longer than a few tens of cycles. In other words, frequency can be scaled by one order of magnitude without violating this assumption.

Second, the performance model assumes that the nonpipelined fraction of the total execution time does not change with clock frequency scaling. This may not be a valid assumption because of (slightly) different bus contention patterns and memory access reorderings due to the different points in time at which off-chip memory requests are being submitted by the processor. We believe this assumption is reasonable in practice though, especially for First-Come-First-Served-based bus and memory controllers.

Third, off-chip store misses may also incur an additional contribution to the nonpipelined fraction of the total execution time. This occurs when retirement blocks on a store miss at the head of a full store buffer *and* dispatch blocks because of a full reorder buffer, issue queue, etc., *and* the store miss does not overlap with other off-chip memory accesses. This occurs on a rare occasion though for most workloads, which is why the counter architecture simply ignores off-chip store misses.

Finally, for the energy predictions, it is assumed that the effective capacitance does not change when scaling clock frequency. Or, in other words, it is assumed that the per-cycle device activity level remains constant across a range of clock frequencies, which we found to be a reasonable assumption.

TABLE 2
The V/f Settings Considered in this Paper

f	V
3.6GHz	1V
2.7GHz	0.88V
1.8GHz	0.76V
0.9GHz	0.64V

3 Experimental Setup

In this paper, we use the SPEC CPU2000 benchmarks; the binaries are highly optimized Alpha binaries (taken from the SimpleScalar website). To limit the simulation time in our experiments, we use representative 100 M-instruction simulation points provided by SimPoint [19].[1] We use the SimpleScalar/Alpha v3.0 out-of-order simulator for all of our experiments. We assume a contemporary 4-wide superscalar out-of-order processor configuration (see Table 1).

The dynamic power model is taken from Wattch v1.02 [3]. We consider two power modes: 1) `cc0` which assumes that all processor structures consume dynamic power every clock cycle, i.e., there is no clock gating and 2) `cc1` which assumes that unused processor structures consume no dynamic power consumption, i.e., they are clock gated. The static power model is taken from HotLeakage [26]; HotLeakage models subthreshold and gate leakage. Further, we assume a 70 nm CMOS chip technology, a nominal $V_{dd,n} = 1\,V$ supply voltage and $f_n = 3.6$ GHz clock frequency. The other V/f settings are shown in Table 2.

4 Evaluation

We now evaluate the proposed counter architecture. We first run each benchmark at the nominal V_n/f_n operating point, and the counter architecture measures the pipelined and nonpipelined fractions of the total execution time. The execution time at another V/f operating point is then estimated using Formula (2); energy consumption is estimated using Formula (3) and (5), for a non-clock-gated and clock-gated processor, respectively. We then compare these estimates against measurements through benchmark simulation at the V/f operating points. We take a similar approach for the other DVFS profitability estimation techniques. Employing DVFS profitability estimation in practice may require profiling the execution during the current interval to predict the next one within a single benchmark run. Because this is the case for all DVFS profitability estimation techniques, we consider a multiple-run setup and focus on evaluating the intrinsic accuracy of the proposed counter architecture.

1. SimPoint selects simulation points based on code signatures, which capture program behavior in a hardware-independent way. As such, the simulation points can be used across different V/f settings.

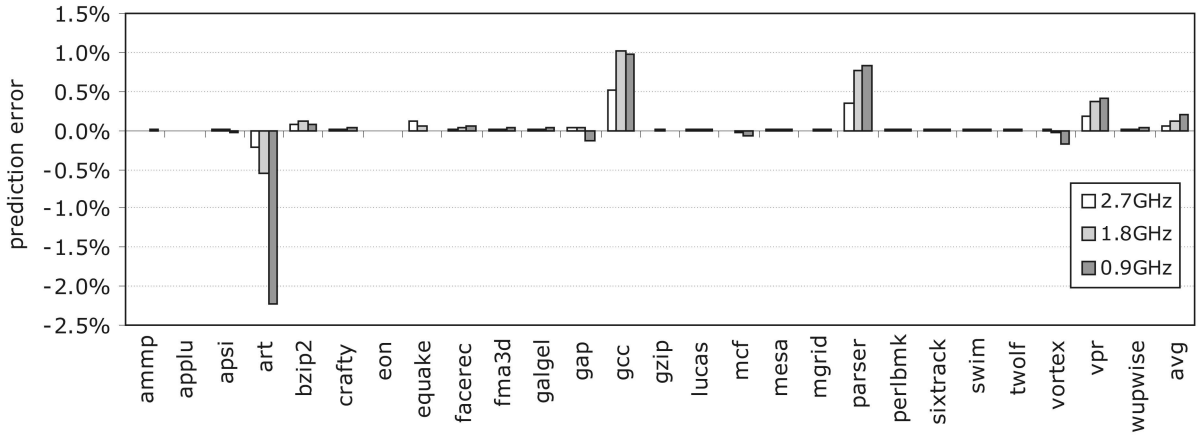

Fig. 4. Prediction error for predicting execution time at three V/f operating points based on a run at the nominal 3.6 GHz operating point.

4.1 Execution Time

Fig. 4 shows the prediction error for the counter architecture for predicting execution time at the 2.7, 1.8, and 0.9 GHz operating points based on a run at the nominal 3.6 GHz operating point. The errors are small and increase with operating points further away from the nominal operating point. The average (absolute) error is 0.2 percent and the maximum error is 2.2 percent at the 0.9 GHz operating point for art. The underestimation for art is due to a second-order effect: front-end miss events (e.g., branch mispredictions and L1 I-cache misses) between two independent load misses increase the time between the independent load misses going to memory at low clock frequencies, and as a result, the processor may stall on the second load miss which is not the case at higher clock frequencies. This appears to be a minor effect only though.

4.2 Comparison to Proportional Scaling, Estimated Linear Scaling, and Stall Cycle Counting

The proposed counter architecture is substantially more accurate than proportional scaling, estimated linear scaling, and stall cycle counting (see Fig. 5). Proportional scaling is accurate for compute-bound applications, however, it incurs large errors for memory-bound and compound compute/memory-intensive applications; the average error equals 33 percent and goes up to 174 percent at the 0.9 GHz operating point.

Estimated linear scaling learns the relationship between performance and clock frequency by building an empirical model that correlates performance with the number of last-level cache misses (or off-chip memory accesses). The empirical model considered here is a linear regression model similar to the work presented in [4], [18], [23]: the independent variable is the number of cache misses per instruction and the dependent variable is the (estimated) execution time. Estimated linear scaling is more accurate than proportional scaling with an average error of 9.8 percent, however, for several benchmarks, estimated linear scaling incurs errors larger than 20 percent and up to 50 percent. The reason for this high inaccuracy is that estimated linear scaling does not accurately account for overlapping off-chip memory accesses due to memory-level parallelism.

Stall cycle counting estimates the nonpipelined fraction of the execution time as the time where no instructions are dispatched. Stall cycle counting is fairly accurate in estimating execution time with an average error of two percent and errors of at most 9.5 percent. However, it does not accurately account for the amount of work that gets done underneath the off-chip load miss penalty and thus underestimates the nonpipelined penalty for off-chip load misses. The proposed counter architecture is more accurate (average error of 0.2 percent and max error of 2.2 percent) at a minor additional hardware cost (6 more bits compared to stall cycle counting).

4.3 Energy Consumption

Fig. 6 shows the error in predicting energy consumption assuming a clock-gated microprocessor. The errors are small with an average error around 0.5 percent and a max error of 1.2 percent. For a

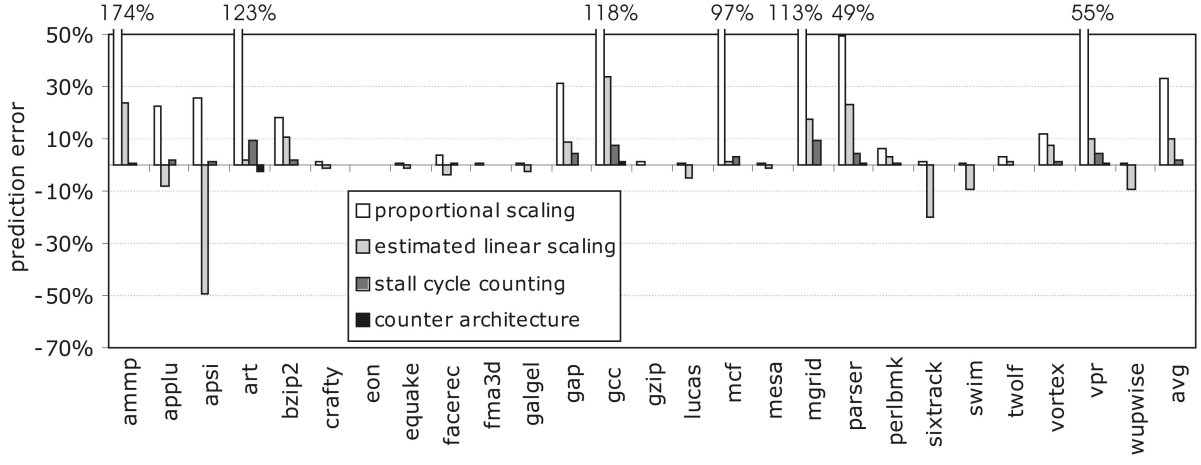

Fig. 5. Prediction error for predicting execution time at the 0.9 GHz operating point for proportional scaling, estimated linear scaling, stall cycle counting, and the proposed counter architecture.

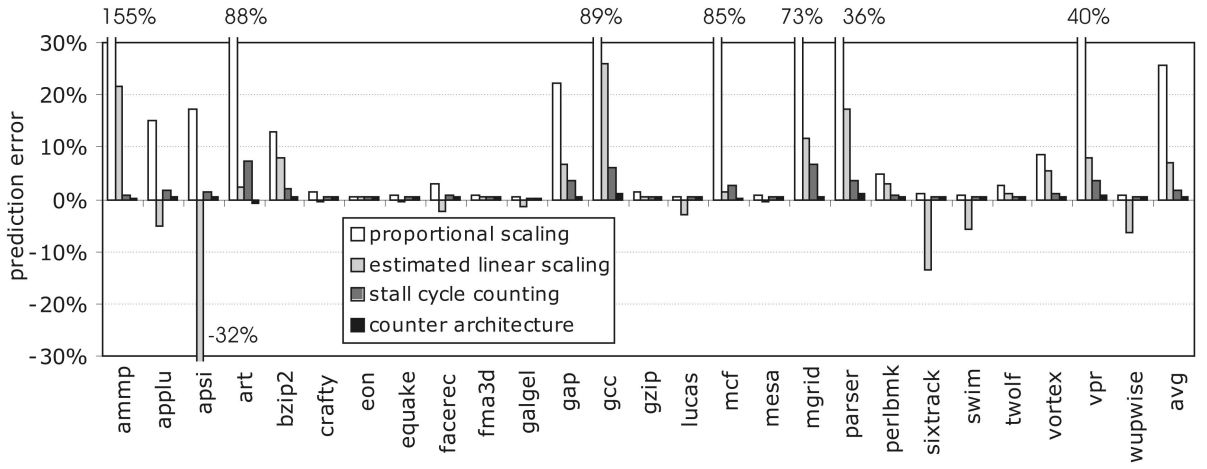

Fig. 6. Prediction error for predicting energy consumption at the 0.9 GHz operating point for proportional scaling, estimated linear scaling, stall cycle counting, and the proposed counter architecture, assuming a clock-gated microprocessor.

non-clock-gated processor, the errors are comparable: a 0.7 percent average error and a 1.7 percent maximum error for art. The counter architecture is more accurate than estimated linear scaling which incurs an average error of 7.1 percent and up to 31.8 percent for a clock-gated processor, and an average error of 10 percent and up to 50 percent for a non-clock-gated processor. Proportional scaling is even more inaccurate. Although it may be accurate over small ranges of V/f settings, as found by others [12], it is highly inaccurate across larger ranges. We found it to generate an average prediction error of 155 and 176 percent for a clock-gated and non-clock-gated processor, respectively, at the 0.9 GHz operating point. Stall cycle counting is relatively accurate compared to proportional scaling and estimated linear scaling (average error of 1.8 percent and max error of 7.4 percent for the clock-gated processor, and 2.5 percent average error and 10.1 percent max error for the non-clock-gated processor), however, it is less accurate than the proposed counter architecture.

4.4 Energy-Efficiency

The EDP and ED^2P metrics are well-known metrics to quantify the energy efficiency of a microprocessor [1]; EDP and ED^2P quantify the energy consumed per unit of performance and are lower-is-better metrics. Fig. 7 quantifies the error in predicting ED^2P at the 0.9 GHz operating point for proportional scaling, estimated linear scaling, stall cycle counting, and the proposed counter architecture, assuming a clock-gated processor. We obtain similar results for EDP and a non-clock-gated processor. The errors for estimating EDP and ED^2P are higher than for estimating energy and execution time, because EDP and ED^2P are composed metrics, i.e., the energy and performance prediction errors magnify each other in the combined energy efficiency metric. Proportional scaling and estimated linear scaling clearly fall short with average errors of 216 and 27 percent, respectively. Stall cycle counting is more accurate with an average error of 6.1 percent and max errors up to 28 percent. The proposed counter architecture is even more accurate with an average error of 0.8 percent and at most five percent (art).

5 Application: Multicore DVFS Power Management

An important objective in multicore processors is to maximize system performance while staying within a predetermined power and temperature budget. Commercial server processors, such as the Intel Itanium Montecito processor [17], feature on-chip circuitry for this purpose: if power consumption is less than its

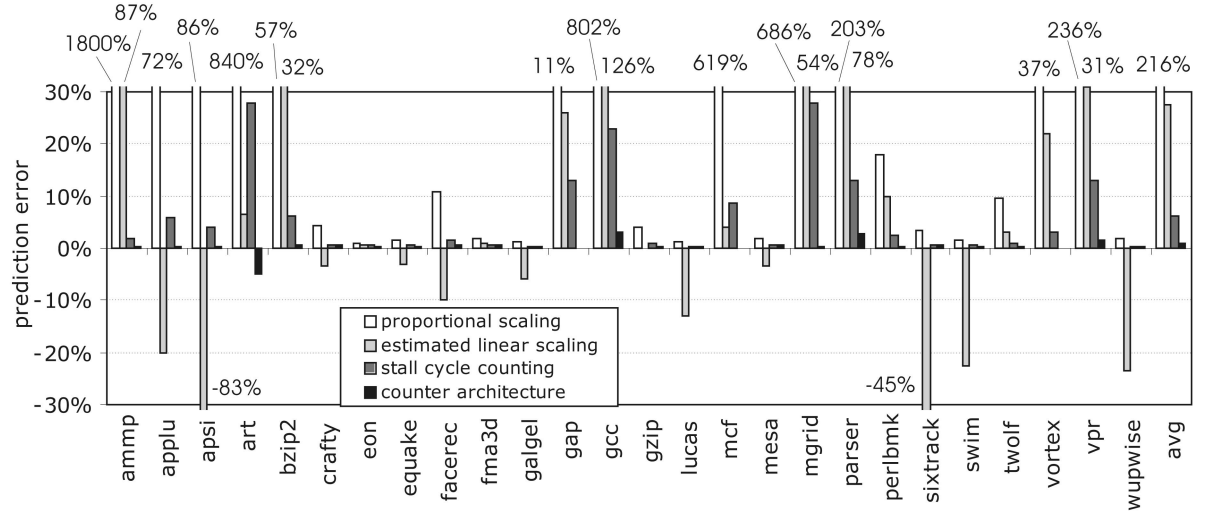

Fig. 7. Prediction error for predicting ED^2P at the 0.9 GHz operating point for proportional scaling, estimated linear scaling, stall cycle counting, and the proposed counter architecture, assuming a clock-gated microprocessor.

budget, voltage and frequency are scaled up, and if power consumption exceeds its budget, voltage and frequency are scaled down. Similarly, the AMD Opteron Quad-Core processor [6] employs core-level frequency scaling to maximize performance within a given power budget.

The proposed counter architecture could be a valuable instrument toward core-level DVFS scheduling in a multicore processor (which may become an important technology for addressing processor variability issues in future multicore processors [22]). Isci et al. [12] propose a core-level DVFS policy: they adjust per-core clock frequency and supply voltage to maximize system throughput while not exceeding the maximum chip-level power budget. They assume proportional scaling which they found to be accurate within four percent for general-purpose applications while scaling frequency over at most 15 percent of its nominal operating point. When scaling frequency over larger ranges, proportional scaling falls short while the proposed counter architecture yields more accurate performance and energy projections, as we have shown earlier in this paper. We now demonstrate that this leads to much better per-core DVFS schedules in a multicore processor.

The input to the setup is the power consumption and execution time at the nominal V_n/f_n operating point for each benchmark. We then estimate the power consumption and execution time at the other V/f operating points through proportional scaling, estimated linear scaling, stall cycle counting, and the counter architecture as proposed in this paper.

We also compute the power consumption and execution time at each V/f operating point through simulation for comparison. (We assume a clock-gated processor in these experiments.) We then consider 20 randomly chosen 4-benchmark combinations assuming that each benchmark runs on a separate core of a 4-core CMP, and determine the best possible per-core V/f settings that maximize system throughput for a given power budget. The power budget is set to 30 W, and performance is quantified in terms of system throughput (STP) and average normalized turnaround time (ANTT) [8]. STP and ANTT quantify the performance of a multiprogram environment in terms of both system-perceived and user-perceived performance, respectively.

Our results (not shown here because of space constraints) report that the counter architecture closely tracks ideal simulation-based DVFS scheduling and does not cause power budget overshoots for any of the workloads. Proportional scaling, on the other hand, results in a power budget overshoot for 4 out of 20 workloads, and this power overshoot goes up to 12.2 percent above the power consumption by the proposed counter architecture. Estimated linear scaling also leads to a power overshoot for a single workload, leading to 10.9 percent more power consumption than the proposed counter architecture. For the other workloads for which none of the approaches lead to a power budget overshoot, the counter architecture achieves similar or significantly higher performance. Compared to estimated linear scaling, the proposed counter architecture leads to up to eight percent higher system throughput and up to 17 percent shorter job turnaround time. Compared to stall cycle counting, the proposed counter architecture leads to up to 15 percent shorter job turnaround time. In summary, the proposed DVFS counter architecture leads to higher system throughput and shorter job turnaround time while not exceeding the target power budget.

6 Conclusion

This paper introduces a counter architecture for online DVFS profitability estimation on superscalar out-of-order processors. The counter architecture computes the pipelined and nonpipelined fractions of the total execution time, and estimates performance and energy by scaling the pipelined fraction under DVFS. Our experimental results using the SPEC CPU2000 benchmarks show that the counter architecture is accurate (performance, energy, and ED^2P prediction errors are within 0.2, 0.5, and 0.8 percent on average, respectively) while incurring limited hardware cost. We have shown the proposed counter architecture to be substantially more accurate than proportional scaling, estimated linear scaling, and stall cycle counting. The online DVFS profitability estimates provided by the proposed counter architecture can be leveraged by various power optimizations both in hardware and software, such as online EDP or ED^2P optimization, single-core DVFS scaling, and per-core DVFS scheduling in multicore processors.

Acknowledgments

We thank the reviewers for their constructive and insightful feedback. Stijn Eyerman is supported through a postdoctoral fellowship by the Research Foundation Flanders (FWO). Additional support is provided by the FWO projects G.0232.06, G.0255.08, and G.0179.10, and the UGent-BOF projects 01J14407 and 01Z04109

References

[1] D. Brooks, P. Bose, S.E. Schuster, H. Jacobson, P.N. Kudva, A. Buyuktosunoglu, J.-D. Wellman, V. Zyuban, M. Gupta, and P.W. Cook, "Power-Aware Microarchitecture: Design and Modeling Challenges for Next-Generation Microprocessors," *IEEE Micro,* vol. 20, no. 6, pp. 26-44, Nov./Dec. 2000.

[2] D. Brooks and M. Martonosi, "Dynamic Thermal Management for High-Performance Microprocessors," *Proc. Seventh Int'l Symp. High-Performance Computer Architecture (HPCA),* pp. 171-182, Jan. 2001.

[3] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations," *Proc. 27th Ann. Int'l Symp. Computer Architecture (ISCA),* pp. 83-94, June 2000.

[4] K. Choi, R. Soma, and M. Pedram, "Fine-Grained Dynamic Voltage and Frequency Scaling for Precise Energy and Performance Trade Off Based on the Ratio of Off-Chip Access to On-Chip Computation Times," *Proc. Symp. Design Automation and Test in Europe (DATE),* pp. 10004-10009, Apr. 2004.

[5] Y. Chou, B. Fahs, and S. Abraham, "Microarchitecture Optimizations for Exploiting Memory-Level Parallelism," *Proc. 31st Ann. Int'l Symp. Computer Architecture (ISCA),* pp. 76-87, June 2004.

[6] J. Dorsey, S. Searles, M. Ciraula, S. Johnson, N. Bujanos, D. Wu, M. Braganza, S. Meyers, E. Fang, and R. Kumar, "An Integrated Quad-Core Opteron Processor," *Proc. IEEE Int'l Solid State Circuits Conf. (ISSCC),* pp. 102-103, Feb. 2007.

[7] D. Ernst, N.S. Kim, S. Das, S. Lee, D. Blaauw, T. Austin, T. Mudge, and K. Flautner, "Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation," *Proc. 36th Ann. IEEE/ACM Int'l Symp. Microarchitecture (MICRO),* pp. 7-18, Dec. 2003.

[8] S. Eyerman and L. Eeckhout, "System-Level Performance Metrics for Multi-Program Workloads," *IEEE Micro,* vol. 28, no. 3, pp. 42-53, May/June 2008.

[9] S. Eyerman, L. Eeckhout, T. Karkhanis, and J.E. Smith, "A Mechanistic Performance Model for Superscalar Out-of-Order Processors," *ACM Trans. Computer Systems,* vol. 27, no. 2, pp. 37, May 2009.

[10] A. Glew, "MLP Yes! ILP No!," *Proc. Eighth Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS) Wild and Crazy Idea Session,* Oct. 1998.

[11] C.J. Hughes, J. Srinivasan, and S.V. Adve, "Saving Energy with Architectural and Frequency Adaptations for Multimedia Applications," *Proc. 34th Ann. IEEE/ACM Int'l Symp. Microarchitecture (MICRO),* pp. 250-261, Dec. 2001.

[12] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi, "An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget," *Proc. 39th Ann. IEEE/ACM Int'l Symp. Microarchitecture (MICRO),* pp. 347-358, Dec. 2006.

[13] C. Isci, A. Buyuktosunoglu, and M. Martonosi, "Long-Term Workload Phases: Duration Predictions and Applications to DVFS," *IEEE Micro,* vol. 25, no. 5, pp. 39-51, Sept. 2005.

[14] C. Isci and M. Martonosi, "Runtime Power Monitoring in High-End Processors: Methodology and Empirical Data," *Proc. 36th Ann. IEEE/ACM Int'l Symp. Microarchitecture (MICRO),* pp. 93-104, Dec. 2003.

[15] T. Karkhanis and J.E. Smith, "A Day in the Life of a Data Cache Miss," *Proc. Second Ann. Workshop Memory Performance Issues (WMPI) Held in Conjunction with Int'l Symp. Computer Architecture (ISCA),* May 2002.

[16] T. Karkhanis and J.E. Smith, "A First-Order Superscalar Processor Model," *Proc. 31st Ann. Int'l Symp. Computer Architecture (ISCA),* pp. 338-349, June 2004.

[17] R. McGowen, C.A. Poirier, C. Bostak, J. Ignowski, M. Millican, W.H. Parks, and S. Naffziger, "Power and Temperature Control on a 90-nm Itanium Family Processor," *IEEE J. Solid-State Circuits,* vol. 41, no. 1, pp. 229-237, Jan. 2006.

[18] C. Poellabauer, L. Singleton, and K. Schwan, "Feedback-Based Dynamic Voltage and Frequency Scaling for Memory-Bound Real-Time Applications," *Proc. IEEE Real-Time Embedded Technology and Applications Symp. (RTAS)*, pp. 234-243, Mar. 2005.

[19] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically Characterizing Large Scale Program Behavior," *Proc. Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 45-57, Oct. 2002.

[20] D.C. Snowdon, S.M. Petters, and G. Heiser, "Accurate On-Line Prediction of Processor and Memory Energy Usage under Voltage Scaling," *Proc. Seventh ACM/IEEE Int'l Conf. Embedded Software (EMSOFT)*, pp. 84-93, Oct. 2007.

[21] J. Srinivasan, S.V. Adve, P. Bose, and J. Rivers, "The Case for Lifetime Reliability-Aware Microprocessors," *Proc. 31st Ann. Int'l Symp. Computer Architecture (ISCA)*, pp. 276-287, June 2004.

[22] R. Teodorescu and J. Torrellas, "Variation-Aware Application Scheduling and Power Management for Chip Multiprocessors," *Proc. 35th Ann. Int'l Symp. Computer Architecture (ISCA)*, pp. 363-374, June 2008.

[23] A. Weissel and F. Bellosa, "Process Cruise Control: Event-Driven Clock Scaling for Dynamic Power Management," *Proc. Int'l Conf. Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, pp. 238-246, Oct. 2002.

[24] Q. Wu, V.J. Reddi, Y. Wu, J. Lee, D. Connors, D. Brooks, M. Martonosi, and D.W. Clark, "A Dynamic Compilation Framework for Controlling Microprocessor Energy and Performance," *Proc. 38th Ann. IEEE/ACM Int'l Symp. Microarchitecture (MICRO)*, pp. 271-282, Nov. 2005.

[25] F. Xie, M. Martonosi, and S. Malik, "Efficient Behavior-Driven Runtime Dynamic Voltage Scaling Policies," *Proc. Third IEEE/ACM/IFIP Int'l Conf. Hardware Software Codesign and System Synthesis (CODES+ISSS '05)*, pp. 105-110, Sept. 2005.

[26] Y. Zhang, D. Parikh, K. Sankaranarayanan, K. Skadron, and M. Stan, "Hotleakage: A Temperature-Aware Model of Subthreshold and Gate Leakage for Architects," technical report, Univ. of Virginia, Mar. 2003.