Practical Performance Prediction Under Dynamic Voltage Frequency Scaling

Barry Rountree[1], David K. Lowenthal[2], Martin Schulz[1], Bronis R. de Supinski[1]

[1]Lawrence Livermore National Laboratory, Computation Directorate,
Livermore, CA 94550, {rountree, schulzm, bronis}@llnl.gov

[2]University of Arizona, Department of Computer Science
Tucson, AZ, 85721, dkl@cs.arizona.edu

Predicting performance under Dynamic Voltage Frequency Scaling (DVFS) remains an open problem. Current best practice explores available performance counters to serve as input to linear regression models that predict performance. However, the inaccuracies of these models require that large-scale DVFS runtime algorithms predict performance conservatively in order to avoid significant consequences of mispredictions. Recent theoretical work based on interval analysis advocates a more accurate and reliable solution based on a single new performance counter, *Leading Loads*.

In this paper, we evaluate a processor-independent analytic framework for existing performance counters based on this interval analysis model. We begin with an analysis of the counters used in many published models. We then briefly describe the *Leading Loads* architectural model and describe how we can use `Leading Loads Cycles` to predict performance under DVFS. We validate this approach for the NAS Parallel Benchmarks and SPEC CPU 2006 benchmarks, demonstrating an order of magnitude improvement in both error and standard deviation compared to the best existing approaches.

I. Introduction

Predicting execution time under Dynamic Voltage Frequency Scaling (DVFS) remains an open problem for domains such as power savings [1], processor thermal regulation [2, 3] and processor reliability [4]. While researchers agree that varying sensitivity to changes in CPU clock frequency arise from memory accesses, our survey of 15 papers spanning nine processor families (and two processor simulators) found 17 unique hardware performance monitor (HPM) combinations used to predict performance. No consensus yet exists for this important, common problem.

A related line of research models the effects of DVFS using interval analysis. This approach provides a clearer understanding of the interactions between processor clock and memory subsystem. Using cycle-accurate simulations, three teams [5–7] independently discovered that a single counter that monitors *Leading Loads* [5] can predict execution time under DVFS with an order of magnitude less error than the best regression-based models. Unfortunately, no existing processors support this significant theoretical result.

Accurate performance prediction under DVFS is particularly vital to green supercomputing. Adaptive runtime systems may adjust the clock frequency to reduce power when this will not impact overall execution time. If a DVFS runtime system mispredicts execution on the critical path and chooses a faster clock frequency than necessary, a local opportunity for energy savings will be lost. On the other hand, choosing a slower frequency than necessary may result in longer execution time, thus multiplying the energy penalty across all processors.

Consider this motivating example: 100 million cores running at 20 megawatts (MW), or 20 megajoules (MJ) of energy per second [8]. If a runtime system correctly predicts that during one particular second of program execution, 10% of cores can be slowed to 50% power consumption, then we realize a 5% savings in energy (1 MJ) for that second. However, if the runtime system underpredicts execution time by 1%, causing the slowed nodes to remain in that state for 1.01 seconds, the system would use an additional 0.19 MJ in energy. At slightly over 5% prediction error, the system consumes more energy than it saved by slowing the cores.

Measuring average error is insufficient for evaluating prediction strategies. A predictor with a relatively low average error but a relatively high standard deviation may

 This work was partially performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.
IGCC July 2011, Orlando, Florida, USA.

significantly increase both execution time and energy consumption. Strategies such as adding buffer time can mitigate the chances of using more energy than was saved, but only at the cost of forgoing even more potential savings. In contrast,we evaluate an HPM that not only reduces median error of predicted execution time across the SpecCPU2006 [9] and NAS Parallel Benchmarks [10] from 2.455% to 0.123%, but also reduces the standard deviation from 2.868 to 0.340.

Current best practice for predictors in runtime systems [1] first minimizes execution time overhead and only then attempts to save energy. This conservative approach masks significant potential energy savings. We target techniques that identify and realize these savings while simultaneously reducing mispredictions that increase time and energy costs.

This paper makes the following contributions:

1) We present a thorough review of the extensive related literature to determine what HPMs have most often been used and why;
2) We perform a comprehensive overview of the effectiveness of these counters when applied to a recent Intel processor;
3) We repeat these experiments through simulation to demonstrate that our proposed counter, leading loads, provides an order of magnitude improvement in both median error and standard deviation.

The structure of the paper follows. Section II provides background in DVFS. Section III presents our results for existing HPMs while Section IV reviews the *Leading Loads* approach. Section V compares the new approach to the approaches based on existing HPMs.

II. Overview

Power and frequency are quadratically related. Increases in CPU clock frequency require a quadratic increase in power, but by the same token reductions provide an opportunity for saving quadratically more power. The domain of high-performance computing (HPC) can leverage this relationship in two different ways. For the same power budget, a dense and highly integrated architecture such as Blue Gene may use orders of magnitude more low-frequency processors than a similar cluster with high-frequency processors. For highly parallel workloads, the tradeoff of increased processor count for decreased processor performance has led to a substantial increase in overall performance.

Dynamic Voltage/Frequency Scaling (DVFS) is orthogonal to the low power processor approach. Instead of always using the same frequency, DVFS allows runtime CPU clock frequency selections that best meet power consumption and performance requirements. This capability is particularly effective with parallel programs. One can select the highest available frequency for overloaded processors while using slower frequencies so that the other processors complete "just in time." Recent work has shown this technique can realize the combined goal for DVFS in HPC: significant energy savings with negligible performance loss [1, 11–13].

Reliable performance predictions at a particular CPU clock frequency support greater energy savings—otherwise processors may complete work early and waste energy idling, or inadvertently introduce delays by taking too long to complete. For *CPU-bound* computation, which requires no main memory accesses, we can easily calculate the change in performance for different frequencies. Given an execution time t_0 at CPU clock frequency f_0 and a target CPU clock frequency f_n, the execution time t_n at the target CPU clock frequency is:

$$t_n = \frac{f_0}{f_n} t_0$$

As the number of memory accesses increases, the program becomes increasingly *memory bound*, with a greater fraction of overall performance dependent on memory latency. Theoretically, an entirely memory-bound program that spends all execution time accessing main memory experiences no slowdown when we reduce CPU clock frequency since the change does not impact main memory access times. Although entirely memory-bound programs do not exist, we must consider the degree of memory boundedness [14] in order to predict performance under DVFS accurately. Time spent waiting for memory at f_0 does not incur performance loss at f_n, and out-of-order processors may overlap those stall cycles with computation at the lower frequency.

III. Evaluation of Existing Models

In this section we explore state-of-the-art techniques to predict performance under DVFS. Models based on instruction rates and miss rates provide reasonably accurate average predictions. However, models that use HPMs that more closely correlate with bus timecan provide greater accuracy. We use these results from real hardware to validate the effectiveness of our model that we present in Section IV.

A. Existing Models and Implementations

Models: Nearly all models to predict performance under changes in CPU clock frequency divide program execution into CPU time and bus time. CPU time scales proportionally to the change in frequency while bus time remains constant. For example, Snowdon et al. attribute execution time to CPU, bus, memory or I/O activity [18]; Lee et al. distinguish between memory instructions and non-memory instructions [16]; and Ge et al. distinguish between on-chip and off-chip time [20].

TABLE I
EVALUATION OF EXISTING MODELS ON THE CORE2 ARCHITECTURE

Model	R-squared	Mean Absolute Error (%)	Median Absolute Error (%)	Standard Deviation of Error
I/C [11, 15, 16]	0.147	5.582	4.867	6.986
I/M [17]	0.445	5.161	4.912	5.986
M/C [11, 15, 18]	0.520	4.856	5.153	5.754
X/C [15, 16]	0.909	1.897	1.060	2.546
R/C [19]	0.945	1.432	1.088	1.943
(I,M)/C [11]	0.505	4.845	5.092	5.772
(X,C)/I [16]	0.923	1.533	0.682	2.278
(I,M,X)/C [15]	0.941	1.278	0.567	1.987

I = Instructions, C = Cycles, M = L2 Cache Misses,
X = Bus Accesses, R = Cycle with 1+ Outstanding Reads.

Implementations: These models capture the intuitive relationship between performance and memory boundedness. They are accurate for single-issue architectures: memory access and computation do not overlap so we can determine memory access time and scale any remaining time proportionally to the change in CPU clock frequency. However, computation may overlap bus execution time on out-of-order issue architectures. Researchers have compensated this overlap through linear regression models that capture correlations between program characteristics and performance. We generally expect that programs with a lower instructions-per-cycle (IPC) rate are more memory bound than ones with a higher IPC. We also expect that last level cache misses per cycle (MPC) are higher in memory-bound programs than CPU-bound ones. A typical regression has the form:

$$\frac{T_{f'}}{T_f} = \alpha_0 + \alpha_1 IPC_f + \alpha_2 MPC_f$$

where f is the reference CPU clock frequency, f' is the target CPU clock frequency, T is execution time at a particular CPU clock frequency, and α_0, α_1, and α_2, are constants that minimize error for the training set. To predict performance of a new application at frequency f', the cycles, misses, instructions and execution time are measured at f. The regression then computes the ratio of the measured time to the predicted time.

Experimental Results: We apply several existing regression-based models [11, 15–19] to *real* executions on a quad-core Xeon 5440 processor. Table I summarizes results for the serial versions of the NAS Parallel Benchmark suite classes A through E, modified to limit total iteration counts. We compile all benchmarks using *gcc* at optimization level $-O2$. Unless otherwise noted, we obtain results by using a linear regression over HPM values per cycle at 2.833GHz (e.g., L2 data cache misses per cycle). We predict the ratio of cycles at 2.000GHz to cycles at 2.833GHz.

R^2 is the standard goodness of fit measure. We calculate error for each benchmark as:

$$\text{error} = 100 \times \frac{\text{predicted} - \text{actual}}{\text{actual}}$$

Table I shows the standard deviation of the error values as well as the mean and median of the absolute error values for the benchmarks as a group. We list counters as well as models with the HPMs treated as a single-HPM model in order to assess their contribution. Our summary lists only principle HPMs of the models although we use the complete models that the references describe.

B. Discussion

Snowdon et al. created an effective model based on MPC for the PXA255 processor [18]. This single-issue architecture creates a direct relationship between the number of cache misses and the amount of time spent on the bus. In out-of-order issue architectures, this link in not nearly as strong, and consequently this HPM is not nearly as useful. Likewise, papers published by Lee, Freeh and Ge examine performance of their models primarily on combinations of cache miss rates and instruction rates [11, 16, 17]. Despite their wide use, these two metrics provide little additional accuracy on out-of-order issue architectures. For example, we report a mean absolute error of 1.897% for a regression using only bus transactions per cycle. Adding both I/C and M/C terms to the regression only reduces the error to 1.278%, but comes at the cost of multiple runs in order to collect the necessary HPM data.

More recent processors provide HPMs that more directly measure bus time. Both Curtis-Maury et al. and Lee et al. exploit these HPMs. Curits-Maury et al. dynamically predict IPC in order to select near-optimal thread granularity [15] while Lee et al. predict execution time [16]. The Curtis-Maury model targeted dynamic concurrency throttling. Here, we reduce the model to the three HPMs that are relevant for predicting performance

across changes in CPU clock frequency: bus transactions, cache misses and IPC rates. This modification outperforms any regression model in the literature.

Finally, Intel processor documentation suggests using the Outstanding Bus Requests HPM with a CMASK=1 in order to count the number of cycles with one or more outstanding reads [19]. We abbreviate this as *1+ Reads/Cycle* or simply *R/C*. This metric is strongly correlated with bus transactions and provides a slightly improved predictor of performance (mean absolute error 1.432%) when compared to the improved Curtis-Maury model. To our knowledge, this HPM has not been used in any existing work in this area, despite its performance equivalence to models requiring additional HPMs.

C. Limitations of Current Models

Regression-based approaches target good *average* prediction accuracy. Error will be split between underprediction, which can lead to unnecessary delays and large energy penalties, as well as overprediction, which results in lost power savings. Further, the quality of the prediction with regression-based models depends on the quality of the training data. If a new program is sufficiently different, the error could be much greater than the regression would lead us to expect. If the phenomenon underlying the regression is understood in sufficient detail, we should be able to predict which programs will be the outliers. Current models do not provide this level of understanding. In the next section, we address these weaknesses by presenting a novel model that explicitly accounts for the out-of-order nature of modern server-class processors. We then use this model to create an improved technique to predict performance.

IV. Architectural Model

The previous section showed that regression-based models only provide good median performance predictions. Outliers have significant error and these models do not suggest *why* this error occurs or how to reduce it. We now consider a *Leading Loads* [5–7], HPM that predicts more accurately and more consistently.

A. Definitions and Assumptions

We define a *load* as a non-speculative read that results in a last-level cache miss. Execution of each load begins with its *issue* on the cycle when the read begins and terminates some constant time later (the *latency*) with its *completion* on the cycle when the data arrives from main memory. We assume the results of the load are required by one or more instructions further downstream, and that these instructions cannot begin execution until the load completes. We define the first load to be a *leading load*. We ignore any other loads that occur between the issue and completion of the leading load, and define the load that occurs subsequent to the completion of the previous leading load as the next leading load.

We classify load issue and completion as *load events*. We divide the instruction stream into *intervals* that intuitively are the instructions that occur between two load events. However, we use a slightly more complicated definition to obtain invariant interval boundaries across CPU clock frequency changes. We define the start of the interval as the first instruction in a set of instructions that cannot begin execution until the associated load event occurs. An interval ends when the next interval begins. We assume deterministic instruction scheduling independent of latency, which still allows load events to occur at different times and in different orders.

Most modern processors use write buffers to hide store miss penalties, making the effective latency zero. We could extend our model to handle write-bound applications for which write latency is a performance limiting factor. However, none of the NPB or SpecCPU benchmarks exhibit this behavior so we do not model writes. Our experimental results in Section V use cycle-accurate simulation and demonstrate that we achieve significant accuracy despite this simplification.

B. Leading Load Implementation

We set the leading load bit to high when a leading load occurs. We reset the bit when the load completes. The `Leading Load Cycles` HPM, which is our estimated bus time, increments every cycle that the bit is high. To implement the leading load bit, we add an extra bit to every load/store queue entry. Insertions into this queue check for any entry with this bit set. If none is found then the bit on the new entry is set. When the load completes, the entry is removed from the queue and the corresponding bit is reset. Thus, we expect an overhead of one bit per queue entry (along with the associated control logic), which is about ten bits on recent Xeons, a negligible increase in area and power.

V. Evaluation of Leading Loads

A thorough exploration of available HPMs failed to yield a combination that approximates *Leading Loads*. We now show how we validated this new HPM approach through the PTLSim cycle-accurate simulator [21].

A. Experimental Setup

PTLSim: Validation of the Leading Loads technique requires a cycle-accurate processor and cache simulator. We made the necessary changes to PTLSim to identify and to track leading loads. This simulator uses a compile-time setting to specify the memory latency: the number of CPU cycles taken by any last-level cache miss. Assuming that changes to CPU clock frequency do not affect the memory bus frequency, we can simulate

$$\text{Predicted Execution Time at } f' = \text{Observed Bus Time } + (f'/f) \times \text{Observed CPU Time}$$
$$\text{Observed Bus Time at } f = \frac{\texttt{LeadingLoadCycles at } f}{f}$$
$$\text{Observed CPU Time at } f = \text{Observed Execution Time at } f - \text{Observed Bus Time at } f$$

Fig. 1. Leading Loads Prediction

changes in CPU clock frequency simply by changing the number of cycles of memory latency (faster CPUs fill the same time with more cycles). We use a 2:1 ratio of high to low CPU clock frequency, predicting performance for a processor running half as fast as the observed processor. This ratio matches the greatest range that we have observed in HPC clusters, and this maximum tends to maximize the error of any predictive model.

Benchmarks: We used three benchmark sets for the simulations: 170 unique instances of a synthetic benchmark that we created to increase test coverage, single iterations of 20 benchmarks taken from the NAS Parallel Benchmark suite classes S, W, and A-C, and complete runs of 58 benchmarks taken from the SpecCPU 2006 benchmark suite classes *test* and *train*.

Leading Loads Models: We use leading loads to measure the number of total effective bus cycles at frequency $2n$ and use this to predict execution time at frequency n. Table II summarizes the results. *Because we require no training data, leading load error is independent of the results of other experiments.*

Regression Models: For each of the existing models, we calculate linear regression predictions for each benchmark set. The model-wide results use standard error, and the error of individual results is calculated as in Section III. *Because we effectively use the same data for both testing and training, we expect the error of the regression models to be higher in practice.* Thus, these regression results (as opposed to the leading load results) are a *lower bound* of potential error. Table II summarizes three single-variable regressions (read cycles, L2 cache misses and instructions) as well as a linear regression that combines all three variables (RMI).

CPU and Memory Boundedness: Predicting behavior of applications with essentially no cache misses (CPU-bound) is trivial, as is predicting behavior of applications that are entirely limited by bus speed and capacity (memory-bound). The execution time of a CPU-bound application changes proportionally to the change in CPU clock frequency while a memory-bound application, at least in theory, shows no change in execution time regardless of the change in CPU clock frequency. In Figure 2 we measure CPU boundedness by dividing the number of *cycles* at the higher frequency by the number of cycles at the lower frequency. Thus, CPU-bound benchmarks, which use roughly the same number of cycles spread over proportionally more time, will be closer to 1.0 and the more memory-bound benchmarks, which use proportionally fewer cycles to take the same amount of time to in the extreme case, will be closer to the ratio of the two frequencies. We focus on where prediction is the most difficult: realistic benchmarks in between these trivial extremes.

B. Discussion

Figure 2 shows results for our 78 benchmarks. The dark points are predictions that use leading loads. The white points use regression-based techniques. Clearly, leading loads consistently generate significantly more accurate predictions for the range of CPU-boundedness.

Table II shows the entire results. The first third shows results obtained from a regression over each HPM individually, which serve as a reference point for judging the remaining results. As in the results obtained on the Xeon processor, we observe that models based only on cache misses or instruction counts perform poorly compared to a model based solely on bus cycles. Combining all three HPMs gives a slight improvement over the bus-cycle-only model. These results form the middle third of Table II. We show the performance of the three-HPM regression across each set of benchmarks (NAS Parallel Benchmarks, SpecCPU 2006 and our synthetic benchmarks) individually and the combined NAS and SPEC benchmarks. The synthetic benchmarks share a similar prediction error with the NAS benchmarks.

The final third of Table II shows the effectiveness of leading loads in performance prediction when compared to the three-HPM regression. Median absolute error decreases across the SpecCPU suite from 2.217% to 0.412%, and across the NPB suite from 0.888% to 0.122%. Across all 218 experiments, median absolute error decreases from 1.465% to 0.199%.

For the combination of NAS and Spec (NPB+Spec) we achieve an order-of-magnitude reduction in median absolute error, from 2.455% to 0.123%. This combination causes particular difficulty for regression-based techniques, as these two suites display different behavior in terms of CPU-boundedness. This result shows leading loads provide a general solution to this problem independent of the characteristics of the underlying applications.

TABLE II
EVALUATION OF *leading loads* USING PTLSIM

Model	Benchmarks	R^2	Mean Absolute Error (%)	Median Absolute Error (%)	Standard Deviation of Error
(R) Read Cycles	all	0.940	2.296	1.837	2.938
(M) L2 Misses	all	0.019	9.193	6.323	12.742
(I) Instructions	all	0.629	6.070	4.893	8.941
RMI	all	0.955	1.944	1.465	2.800
RMI	NPB+Spec	0.976	2.379	2.455	2.868
RMI	NPB	0.993	0.888	0.760	1.277
RMI	Spec	0.974	2.115	2.217	2.700
RMI	synth	0.974	0.992	0.744	1.348
Leading Loads	all	n/a	0.276	0.199	0.356
Leading Loads	NPB+Spec	n/a	0.248	0.123	0.340
Leading Loads	NPB	n/a	0.122	0.095	0.108
Leading Loads	Spec	n/a	0.292	0.142	0.382
Leading Loads	synth	n/a	0.288	0.225	0.353

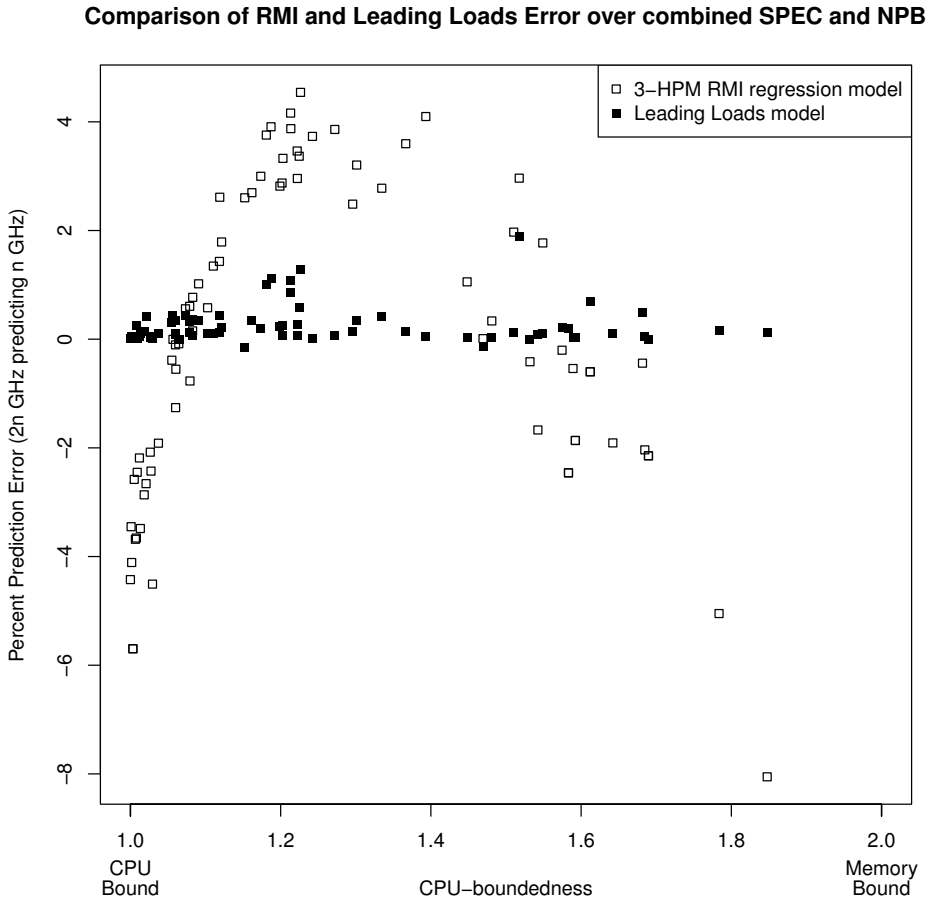

Fig. 2. Comparative Performance of RMI and Leading Loads models.

TABLE III
CLASSIFICATION OF HARDWARE PERFORMANCE COUNTERS USED IN EXISTING MODELS.

Counter Type	Counter Name
Instruction Counts	Total Instructions [11, 15–17, 22–28] Branch Instructions [15, 26, 27, 29] Load and Store Instructions [29] Single and Double Precision SIMD ops [26]
Cycle Counts	Total Stall Cycles [26] Data Depedence Stall Cycles [25] Decoder/Dispatch Stalls [30] Branch Misprediction Stalls [30] I/O Stalls [30] Reorder Buffer Full Stalls [28, 30, 31] Idle Cycles [30] Active Cycles [26]
Cache Accesses	Last Level Cache Miss [11, 23, 27, 30, 32] L1 Data Cache Access [11, 22] L1 Data Cache Miss [24] Trace Cache Deliver Mode [15] L2 Cache Evictions [31]
Memory Accesses	TLB Miss [32, 33] Bus Access [15, 16] DRAM Access due to Page Conflict [28, 31] Burst Transactions [31] Quadword Write Transfers [31] Remote memory accesses [32] Cancelled Memory Accesses [26]
Other	Mispredicted Branches [15] Machine Clears [26] TC Deliver Mode [26] UOP Queue Writes [26]

VI. RELATED WORK

This research intersects several disparate fields. The problem of energy as a design constraint in high-performance computing has led to low-power processors in high-density supercomputers [34,35]. DVFS has been applied to determine the most efficient per-program allocation of processors and frequencies [36] as well as to application-oblivious runtime algorithms [1, 11, 37]. These applications have been limited by the absence of a reliable predictive model of program performance under changing CPU clock frequencies. Work in bounding potential energy savings required complete program execution at every frequency [38], and runtime algorithms commonly assume all slowdown is scaled slowdown until demonstrated otherwise [1].

We refer the reader to table III for the counters used by existing predictive models.

VII. CONCLUSIONS: IMPLICATIONS FOR GREEN SUPERCOMPUTING

Prior HPM-based models can be classified as either high-level (with significant error) or exhaustive-search (which provides little explanation as to why the counters work or when they will fail). We have taken a new architecture model of the relationship between CPU clock frequency, memory usage, and performance and from this evaluated a proposed HPM, *leading loads*. We have compared this approach to performance prediction under DVFS to the best existing performance counter approaches and found order-of-magnitude reductions in error and standard deviation.

This model and predictor will help to create DVFS-based runtime systems that aggressively and reliably save energy. Given that underprediction on the critical path can waste energy across the entire system, we lose significant opportunities for energy savings when prediction error is high. Reducing prediction error means that runtime systems can be less conservative when scheduling CPU frequencies to save energy. Reducing the standard deviation reduces the impact when errors do occur. By relying on an arithmetic model rather than a linear regression, we have removed sources of error resulting from incomplete training sets and thus can be more confident that these results will be reproducible across other applications.

Much work remains. We have explored the single-core case; multicore will introduce complications due to bus contention across cores. Newer architectures such as Nehalem and Sandy Bridge have multiple memory latencies and will "overclock" a subset of cores if the remainder are idle. Accounting for these architectural features will require additions to this model.

REFERENCES

[1] B. Rountree, D. K. Lowenthal, B. de Supinski, M. Schulz, and V. W. Freeh, "Adagio: Making DVS Practial for Complex HPC Applications," in *International Conference on Supercompuing (ICS)*, Yorktown Heights, New York, Jun. 2009.

[2] H. Hanson, S. W. Keckler, S. Ghiasi, K. Rajamani, F. Rawson, and J. Rubio, "Thermal Response to DVFS: Analysis with an Intel Pentium M," in *International Symposium on Low Power Electronics and Design (ISLPED)*, Portland, OR, 2007.

[3] J. S. Lee, K. Skadron, and S. W. Chung, "Predictive Temperature-Aware DVFS," *IEEE Transactions on Computers (ToC)*, vol. 59, no. 1, pp. 127–133, 2010.

[4] M. Basoglu, M. Orshansky, and M. Erez, "NBTI-aware DVFS: A New Approach to Saving Energy and Increasing Processor Lifetime," in *International Symposium on Low Power Electronics and Design (ISLPED)*, Austin, TX, 2010.

[5] B. Rountree, "Theory and practice of dynamic voltage/frequency scaling in the high-performance computing environment," Ph.D. dissertation, University of Arizona, 2010.

[6] S. Eyerman and L. Eeckhout, "A Counter Architecture for Online DVFS Profitability Estimation," *IEEE Transactions on Computers (ToC)*, 2010, preprint.

[7] G. Keramidas, V. Spiliopoulos, and S. Kaxiras, "Interval-Based Models for Run-Time DVFS Orchestration in Superscalar Processors," in *Proceedings of the 2010 International Conference on Computing Frontiers (CF)*, Bertinoro, Italy, 2010.

[8] Exascale Initiative Steering Committee, "A decadal DOE plan for providing exascale applications and technologies for DOE mission needs," 2010.

[9] Standard Performance Evaluation Corp., *SpecCPU 2006*. http://www.spec.org, 2006.

[10] NASA Advanced Supercomputing Division, *NAS Parallel Benchmark Suite*. http://www.nas.nasa.gov/Resources/Software/npb.html, 2006, version 3.3.

[11] R. Ge, X. Feng, W. Feng, and K. W. Cameron, "CPU Miser: A performance-Directed, Run-Time System for Power-aware Clusters," in *Proceedings of the 2007 International Conference on Parallel Processing (ICPP)*, Xi'An, China, 2007.

[12] C.-H. Hsu, W.-C. Feng, and J. S. Archuleta, "Towards Efficient Supercomputing: A Quest for the Right Metric," in *High-Performance Power-Aware Computing (HPPAC)*, Denver, Colorado, 2005.

[13] N. Kappiah, V. W. Freeh, D. K. Lowenthal, and F. Pan, "Exploiting Slack Time in Power-Aware, High-Performance Programs," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, Seattle, Washington, Nov. 2005.

[14] C.-H. Hsu and W.-C. Feng, "A Power-Aware Run-Time System for High-Performance Computing," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, Seattle, Washington, Nov. 2005.

[15] M. Curtis-Maury, J. Dzierwa, C. D. Antonopoulos, and D. S. Nikolopoulos, "Online Power-Performance Adaptation of Multithreaded Programs using Hardware Event-Based Prediction," in *International Conference on Supercomputing (ICS)*, Queensland, Australia, Jun. 2006.

[16] S.-J. Lee, H.-K. Lee, and P.-C. Yew, "Runtime Performance Projection Model for Dynamic Power Management," in *12th Asia-Pacific Conference on Advances in Computer Systems Architecture (ACSAC)*, Seoul, Korea, Aug. 2007.

[17] V. W. Freeh, D. K. Lowenthal, F. Pan, N. Kappiah, and R. Springer., "Exploring the Energy-Time Tradeoff in MPI Programs on a Power-Scalable Cluster," in *19th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Denver, Colorado, Apr. 2005.

[18] D. C. Snowdon, G. van der Linden, S. M. Petters, and G. Heiser, "Accurate Run-Time Prediction of Performance Degradation Under Frequency Scaling," in *Operating System Platforms for Embedded Real-Time Applications (OSPERTA)*, 2007.

[19] Intel, *Intel 64 and IA-32 Architectures Optimization Reference Guide*. Intel Corporation, Nov 2007, no. 248966-016.

[20] R. Ge and K. W. Cameron, "Power-Aware Speedup," in *21st IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Long Beach, California, Mar. 2007.

[21] M. T. Yourst, "PTLsim: A Cycle Accurate Full System x86-64 Microarchitectrual Simulator," in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, San Jose, CA, 2007.

[22] M. Curtis-Maury, A. Shah, F. Blagojevic, D. S. Nikolopoulos, B. R. de Supinski, and M. Schulz, "Prediction Models for Multi-dimensional Power-Performance Optimization on Many Cores," in *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, Toronto, Canada, Oct. 2008.

[23] G. Dhiman, T. Rosing, V. Kontorinis, E. Saxe, D. Tullsen, and J. Chew, "Dynamic Workload Characterization for Power Efficient Scheduling on CMP Systems," in *International Symposium on Low Power Electronics and Design (ISLPED)*, Austin, TX, USA, 2010.

[24] M. Moeng and R. Melhem, "Applying Statistical Machine Learning to Multicore Voltage and Frequency Scaling," in *Proceedings of the 2010 International Conference on Computing Frontiers (CF)*, Bertinoro, Italy, 2010.

[25] K. Choi, R. Soma, and M. Pedram, "Dynamic Voltage and Frequency Scaling Based on Workload Decomposition," in *International Symposium on Low Power Electronics and Design (ISLPED)*, Newport Beach, CA, USA, 2004.

[26] M. Curtis-Maury, F. Blagojevic, C. D. Antonopoluos, and D. S. Nikolopoulos, "Prediction-Based Power-Performance Adaptation of Multithreaded Scientific Codes," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 10, pp. 1396–1410, 2008.

[27] A. Weissel and F. Bellosa, "Process Cruise Control: Event-Driven Clock Scaling for Dynamic Power Management," in *International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES)*, Grenoble, France, 2002.

[28] G. Dhiman and T. S. Rosing, "Dynamic Voltage Frequency Scaling for Multi-Tasking Systems Using Online Learning," in *International Symposium on Low Power Electronics and Design (ISLPED)*, Portland, Oregon, Aug. 2007.

[29] D. Sasikala, M. E. Ravichandran, and C. S. Ravichandran, "Processor Performance Enhancement Using Self-Adaptive Clock Frequency," *International Journal of Computer Applications (IJCA)*, vol. 3, no. 11, pp. 19–26, 2010.

[30] S. Huang and W. Feng, "Energy-Efficient Cluster Computing via Accurate Workload Characterization," in *Proceedings of the 9th International Symposium on Cluster Computing and the Grid (CCGRID)*, Shanghai, China, 2009.

[31] D. C. Snowdon, E. L. Sueur, S. M. Petters, and G. Heiser, "Koala: A Platform for OS-Level Power Management," in *Proceedings of the 4th Eurosys Conference*, Nuremberg, Germany, 2009.

[32] V. Bui, L. C. McInnes, B. Norris, L. Li, K. Huck, O. Hernandez, and B. Chapman, "A Component Infrastructure for Performance and Power Modeling of Parallel Scientific Applications," in *Proceedings of the 2008 CompFrame/HPC-GECO workshop on Component-based High Performance Computing (CBHPC)*, Karlsruhe, Germany, 2008.

[33] D. C. Snowdon, S. M. Petters, and G. Heiser, "Accurate On-line Prediction of Processor and Memory Energy Usage Under Voltage Scaling," in *Conference on Embedded Software (EMSOFT)*, Salzburg, Austria, Sep. 2007, pp. 84–93.

[34] A. Gara, M. Blumrich, D. Chen, G. L.-T. Chiu, P. Coteus, M. E. Giampapa, R. Haring, P. Heidelberger, D. Hoenicke, G. Kopcsay, T. Liebsch, M. Ohmacht, B. Steinmacher-Burow, T. Takken, and P. Vranas, "Overview of the Blue Gene/L System Architecture," *IBM Journal of Research and Development*, vol. 49, no. 2-3, pp. 195–212, 2005.

[35] K. J. Barker, K. Davis, A. Hoisie, D. J. Kerbyson, M. Lang, S. Pakin, and J. C. Sancho, "Entering the Petaflop Era: The Architecture and Performance of Roadrunner," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, Austin, Texas., Nov. 2008.

[36] R. Springer, D. K. Lowenthal, B. Rountree, and V. W. Freeh, "Minimizing Execution Time in MPI Programs on an Energy-Constrained, Power-Scalable Cluster," in *Principles and Practice of Parallel Programming (PPoPP)*, New York City, New York, Mar. 2006.

[37] V. W. Freeh, N. Kappiah, D. K. Lowenthal, and T. K. Bletsch, "Just-In-Time Dynamic Voltage Scaling: Exploiting Inter-Node Slack to Save Energy in MPI Programs," *Journal of Parallel and Distributed Computing*, vol. 68, no. 9, pp. 1175–1185, 2008.

[38] B. Rountree, D. K. Lowenthal, S. H. Funk, V. W. Freeh, B. R. de Supinski, and M. Schulz, "Bounding Energy Consumption in Large-Scale MPI Programs," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, Reno, Nevada, Nov. 2007.