Implementing a Leading Loads Performance Predictor on Commodity Processors

Bo Su[†][*], Joseph L. Greathouse[‡], Junli Gu[‡], Michael Boyer[‡], Li Shen[†], Zhiying Wang[†]

[†]State Key Lab of HPC, College of Computer, National University of Defense Technology
[‡]AMD Research

Abstract

Modern CPUs employ Dynamic Voltage and Frequency Scaling (DVFS) to boost performance, lower power, and improve energy efficiency. Good DVFS decisions require accurate performance predictions across frequencies. A new hardware structure for measuring *leading load* cycles was recently proposed and demonstrated promising performance prediction abilities in simulation.

This paper proposes a method of leveraging existing hardware performance monitors to emulate a leading loads predictor. Our proposal, LL-MAB, uses existing miss status handling register occupancy information to estimate leading load cycles. We implement and validate LL-MAB on a collection of commercial AMD CPUs. Experiments demonstrate that it can accurately predict performance with an average error of 2.7% using an AMD Opteron$^{\text{TM}}$4386 processor over a 2.2x change in frequency. LL-MAB requires no hardware- or application-specific training, and it is more accurate and requires fewer counters than similar approaches.

1 Introduction

Dynamic voltage and frequency scaling (DVFS) is used to optimize performance under power and energy constraints, typically under the control of the OS or firmware. One of the key challenges of utilizing DVFS effectively is dynamically predicting the performance impact of frequency changes for arbitrary applications. This can be difficult because program execution time does not depend solely on core frequency. While some sections of a program will run faster at higher frequencies, others are limited by non-core components, such as DRAM latency. Because of this, simple linear scaling models (where performance is directly proportional to frequency) often yield inadequate estimates [5].

Unfortunately, it can be difficult to predict the effect of memory accesses on performance. Not all memory accesses cause stalls to the core because of caches in the core clock domain. In addition, processors can exploit memory-level parallelism (MLP) by overlapping multiple cache misses. As such, not all cache misses directly affect the performance. Finally, DRAM access latency varies with access patterns, making it difficult to predict time spent waiting on memory from access counts alone.

One promising approach for predicting DVFS performance is the recently proposed *leading loads* model [5, 7, 10], which splits the execution time of an application into time spent in the core (which changes along with frequency) and in the memory (which does not). It then uses new leading load performance counters to accurately estimate this memory time. Simulations demonstrated that this approach could predict application performance under DVFS with an order of magnitude higher accuracy than previously proposed models, while requiring no training (unlike regression-based models). These results led the authors to suggest that hardware support for leading loads should be added to future processors.

This paper demonstrates how to leverage *existing* hardware performance counters that measure miss status handling register (MSHR) activity on commodity AMD processors in order to approximate a leading loads performance predictor. This predictor can accurately estimate the performance impact of DVFS changes on arbitrary applications running on real hardware. It requires no hardware- or application-specific training, and uses only a small number of performance counters.

We validate our method on three different AMD processors across multiple hardware generations. We compare our technique with previously proposed predictors and explain how it is different from an ideal leading load predictor. We show that our model provides a more accurate prediction with less variance in error rate than other predictors that work on existing hardware. To the best of our knowledge, this is the first time the leading loads model has been demonstrated on real hardware.

[*]This work took place while Bo Su interned with AMD Research.

2 Related Work

There is a large body of work on performance modeling under frequency variation. Rountree et al. [10] and Eeckhout [4] describe many of the previous techniques. Eeckhout categorizes analytic performance estimation models into *empirical* models, which use black-box approaches, and *mechanistic* models which are designed from the underlying machine principals. Some of the most popular empirical models are based on regression [11]. They can have good accuracy, but need many input variables and long training runs. Their accuracy is a function of the quality of the training set, and they must be retrained whenever the underlying machine changes.

Mechanistic models often include simplifications and abstractions to make the problem tractable. Proportional scaling models (or "linear scaling") are the simplest and assume that performance scales linearly with frequency. These are simple to implement, but only work well when the application spends little time accessing memory.

Recent mechanistic DVFS estimation models are built from the underlying concept that program execution time is split into *core time* (time doing work) and *memory time* (time stalled waiting for memory). Core time is inversely proportional to frequency. However, because the latency to memory does not change when the core's frequency changes, memory time is not affected by DVFS.

The difficulty of this performance estimation mechanism lies in appropriately characterizing these times. Modern processors can execute instructions out of order, with multiple loads accessing memory in parallel. Simple memory models do not capture this, which leads to incorrect estimates. For example, stall models monitor the amount of time that a core is not committing instructions and assume that this is due to time spent in the memory system [5]. We will show later that this is often an inaccurate assumption, since processors can stall for numerous reasons besides memory latency.

2.1 Leading Loads (LL) Model

Leading loads were simultaneously defined by three groups attempting to solve the problems of these linear and stall models [5, 7, 10]. They utilized the insight that, while many memory accesses may be outstanding, only one can stall the pipeline. As such, the first non-speculative load that misses in the last level of the core's cache is considered a *leading load*. The time between the miss and when it returns is assumed to be memory time. This time is counted even if core work continues under the miss. All misses until this load returns are not leading loads – they represent MLP . A simplifying assumption of this model is that these MLP accesses will not eventually stall the pipeline.

Once a leading load returns, the next miss becomes the leading load. All time when there is no leading load is counted as core time. This is illustrated in Figure 1, where there are 3 load misses: A, B and C. The misses begin at t1, t2, and t6, then finish at t4, t5, and t8, respectively. A and C are leading loads; their delay is memory time. B is not a leading load because a leading load (A) already exists; it is MLP and does not stall the core.

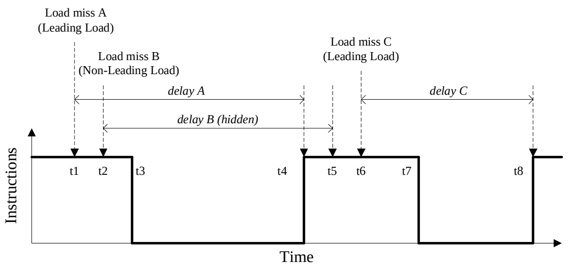


Figure 1: Leading loads example.

There are limitations to this model, such as the assumption that MLP loads will not stall the pipeline, and its inability to deal with bandwidth-bound applications [9]. However, the simulated results for these counters appear promising (with estimation errors of 0.2%), and the hardware is simple, requiring only a single counter per core. The major impediment was the apparent lack of leading load hardware performance events, which prevented testing this mechanism outside of simulation.

2.2 Green Governor (GG) Model

Because their newly proposed leading load counter did not yet exist in hardware, Spiliopoulos et al. also devised a simpler model in their Green Governors work that used existing counters [12]. They monitor the number of last level cache (LLC) misses and number of cycles without a retired instruction. They then characterize the average miss latency using a tool such as lmbench [8] and multiply this delay by the number of LLC misses to estimate memory time. If the amount of time not retiring instructions is less than this, the smaller time is used instead.

When predicting the program's performance from frequency f to f', this model can be described by Equation 1. The memory time M_t is calculated using stall cycles S, the number of LLC misses N, the per-miss delay time D, and the original frequency f. The new execution time T' is then calculated from M_t, the original execution time T, and the ratio of frequency change.

$$T' = (T - M_t) \times \frac{f}{f'} + M_t; M_t = min(\frac{S}{f}, N \times D) \qquad (1)$$

This model makes a number of simplifying assumptions, since it is constrained by existing hardware. First, it assumes that LLC misses have a constant latency, since it only measures miss count. Second, it ignores MLP. Nonetheless, with careful tuning, it can yield reasonably accurate performance estimates.

3 Implementing Leading Loads

On AMD CPUs, a Miss Address Buffer (MAB)[1] is a structure that tracks a single outstanding cache miss. Misses are assigned to available MABs based on a fixed priority; a miss that occurs when the highest priority MAB is available will always be assigned to that MAB. The first miss will be assigned to the highest priority MAB, as will the next miss after that one returns. Thus, the amount of time that the highest priority MAB is occupied represents the aggregate latency of all of the leading loads. This MAB's occupancy time can be measured directly (in clock cycles) using a hardware performance counter. To the best of our knowledge, no such mechanism exists in processors from other vendors.

We thus describe our LL-MAB model, which assumes that the wall-clock occupancy time of the highest priority MAB will remain unchanged across different core frequencies, since its occupancy relies on values returning from memory. The remaining execution time, when nothing is in the highest priority MAB, is core time, which is inversely proportional to frequency. More formally, if an application's execution time is T at frequency f and the highest priority MAB is occupied for M clock cycles, then the predicted execution time T' of the same application at a frequency f' is given by Equation 2.

$$T' = \frac{M}{f} + (T - \frac{M}{f}) \times \frac{f}{f'} \tag{2}$$

While Family 15h cores measure MAB occupancy time for L2 cache misses, Family 10h cores measure *L1* cache misses. For the purposes of implementing the LL model, this introduces two inaccuracies. First, a leading load from the L1 may still hit in the L2, which is also in the core clock domain. Second, for leading loads that miss in both the L1 and L2 caches, the MAB occupancy time includes the latency of the request from the L1 to L2. Neither should be counted as leading load time, since they will change as the core frequency changes.

In addition, the MABs hold prefetch misses, which should not be counted as leading loads because they will not cause the core to stall. We will show in Section 4 that these inaccuracies are small enough that LL-MAB model is still more accurate than existing predictors.

We implemented LL-MAB on three different AMD processors with two different microarchitectures, described in Table 1. These cores assign MABs in slightly different orders: Family 10h and Family 15h processors give highest priority to MAB1 and MAB0, respectively. Our LL-MAB implementation uses these counters to estimate the leading load time.

Table 1: Processor configurations and hardware events.

	AMD Phenom^{TM}II X6 1090T	**AMD Opteron^{TM} 4386**	**AMD A10-5800K**
Family	10h	15h	15h
Core Freq.	1.6/3.2 GHz	1.4/3.1 GHz	1.4/3.8 GHz
DRAM	DDR3-800	DDR3-1600	DDR3-1066
MAB Counter	MAB1	MAB0	MAB0
L3 Latency	13.0ns	32.2ns	n/a
L2 Latency	24.7ns	12.8ns	32.3ns
H/W Event		**Event Select Code**	
E1 Exe. Cyc.	0x00410076	0x00410076	0x00410076
E2 MAB Cyc.	0x00410169	0x00410069	0x00410069
E3 Stall Cyc.	0x014100c0	0x014100c0	0x014100c0
E4 L3 Misses	0x4004107e0	0x40040f7e1	No L3 Cache
		0x40040ffe1	
E5 L2 Misses	0x0001077e	0x00410043	0x00410043

We also implemented an enhanced version of the Green Governor (GG) performance estimation model for comparison. Because the L3 cache in AMD processors is in a separate clock domain from the cores, its access time will remain constant at different core frequencies. Spiliopoulos et al. measured *last level cache* misses, which means that they did not measure L2 (core domain) misses that hit in the L3 (memory domain). To compare both of these designs, we build two GG models: one that counts L2 misses, and one that counts L3 misses.

Different memory access patterns cause different DRAM delays. As such, rather than using a single memory latency chosen arbitrarily from a microbenchmark, we instead search the space of possible latencies to find the value that yields the lowest estimation error. This means that we are testing the algorithm on its training data, which may yield optimistic results from the GG model. However, this allows us to operate under the assumption that the GG model's latency has been chosen well (which may not always be the case).

Finally, where possible, our GG models did not use cache misses caused by prefetchers. On Family 15h processors, specifying the appropriate selection of L3 performance events allows us to ignore prefetch misses. However, this is not possible on Family 10h processors.

Table 1 shows the configurations of the systems we measured and the specific performance events we used to collect the data required by the predictors in our experiments. All of the predictors use the Program Cycles counter, though this could potentially be replaced by the hard-coded timestamp counter to reduce counter requirements. The linear estimation method uses only this, while the stall model also uses Stall Cycles. LL-MAB model needs only one additional counter: MAB Wait Cycles. The GG models need 2 or 3 more. GG-L3 uses Stall Cycles and L3 Misses (Family 15h uses 2 hardware counters for this in order to remove prefetch misses). In GG-L2, L3 Misses are replaced by L2 Misses.

[1]Commonly known as a Miss Status Handling Register (MSHR).

4 Evaluation Results and Analysis

4.1 Experimental Methodology

We validated our LL-MAB model using 66 single-threaded benchmarks from the NAS Parallel Benchmarks [1][2], PARSEC [2][3], Rodinia [3][4], and SPEC CPU2006 [6][5]. For comparison, we also tested the linear, stall, GG-L2 and GG-L3 models.

The AMD Phenom^{TM}II 1090T processor ran Canonical Ubuntu Desktop 12.04 (kernel 3.2.0-24), the AMD Opteron^{TM}4386 processor ran Fedora^{TM}19 Desktop (kernel 3.10.6-200), and the AMD A10-5800K processor ran CentOS release 6.4 (kernel 2.6.32-358.23.2). We used msr-tools to set and read the performance counters, cpufreq to change DVFS states, and numactl to lock each benchmark to a single core. For each processor, we measured two frequencies, the higher at least twice the lower. We then estimated the change in user-level unhalted clock cycles between the two frequencies.

As mentioned, we chose the GG miss latency that provided the best average prediction accuracy across all of the benchmarks, based on an exhaustive search of all possible latency values from 0 to 200ns with a step size of 0.1ns. These values are also listed in Table 1. Because the L3 misses on the AMD Phenom^{TM}II 1090T processor include prefetch misses, the ideal L3 miss latency was lower than the L2 miss latency. These values are up to $2\times$ different than those measured with microbenchmarks, which implies that our GG results are optimistic.

4.2 Experimental Evaluation

Figures 2, 3, and 4 show the average and standard deviation of the absolute value of the prediction errors. In this case, error is the difference between the estimated and the actual cycles at one frequency when gathering statistics at the other. Unless noted, all results are listed in the order: AMD Phenom^{TM}II 1090T processor, AMD Opteron^{TM}4386 processor, AMD A10-5800K processor. The stall-based model had much higher average error than the others and is not shown. Its average errors were 21.7%, 31.3% and 32.0%, respectively.

As shown in the figures, our LL-MAB predictor had the lowest average error and standard deviation for all three processors. The GG-L2 and GG-L3 models are slightly worse, but have similar accuracy to one another. It is worth reiterating that their latency values were carefully tuned to reduce error rate, so these numbers do not necessarily mean that one is better than the other. Nonetheless, LL-MAB model's average error was 5.27%, 2.71%, and 4.80%, 1.22, 1.30, and 1.71 percentage points lower than the most accurate GG model. Linear estimation had the worst average prediction error among the models shown.

While a lower prediction error is preferred, the second graph in each of these figures demonstrates the standard deviation of these error rates. In this case, the LL-MAB model had a much smaller variance in its errors, implying a more consistent error rate. This is especially important for performance estimation, since an outlier can lead to a severe loss in performance or energy efficiency.

Figures 5 and 6 plot the absolute prediction error versus memory boundedness for each benchmark. As described by Rountree et al. [10], memory boundedness is the ratio of measured execution cycles at the two frequencies. For compute-bound applications, the number of execution cycles should be (approximately) fixed regardless of the frequency, so the memory boundedness should be (approximately) one. Larger values indicate applications that spend more time in the memory system.

By definition, the linear model's error is proportional to the memory boundedness, so its error rate was highest for memory-bound programs. The stall-based model, on the other hand, exhibited large errors for compute-bound programs, because it incorrectly assumed that compute-bound applications with many pipeline stalls (due to, for example, mispredicted branches) were memory bound.

The GG models use cache miss counts to reduce the prediction error when the memory boundedness is low. In this way, the GG models overcame the high error rate of the stall-based model for more compute-bound applications, while keeping the relatively low error rate of the stall-based model for more memory-bound applications.

Our LL-MAB model was accurate across a range of benchmarks and hardware, because it more directly measures the time spent in the memory system. However, as the memory boundedness increases, the limitations discussed in Section 3 cause some errors. For the programs whose memory boundedness is greater than 1.1, the average absolute error of LL-MAB is still the lowest.

4.3 LL-MAB Model Discussion

LL-MAB demonstrates three primary advantages:

1. LL-MAB provides better average prediction accuracy. This was true despite the fact that we gave our comparison point, GG, as many advantages as possible. The accuracy of the GG models would be even worse using directly measured miss latencies.

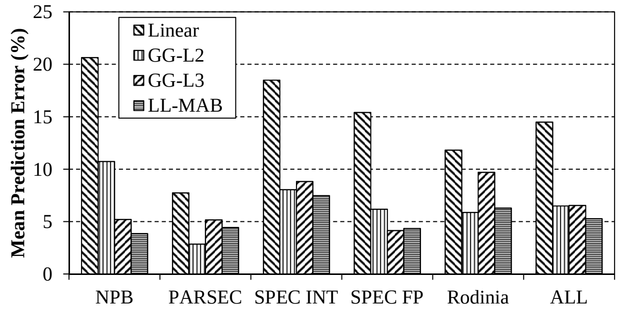
2. LL-MAB is easier to implement. It only requires two performance counters, while the GG models need three or four. The LL-MAB model also requires no hardware- or application-specific training, unlike Green Governor or regression models.

[2]NPB: All 10 *SER* programs; size "B" for DC and "C" for others.

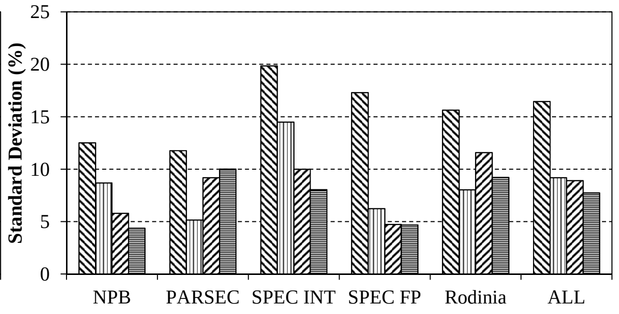
[3]PARSEC: All 13 *gcc-serial* benchmarks with *native* inputs.

[4]Rodinia: bfs, b+tree, heartwall, hotspot, kmeans, lavaMD, leukocyte, lud, particlefilter, pathfinder, srad, cfd, nw, streamcluster.

[5]SPEC CPU2006: All 29 benchmarks with *ref* inputs.

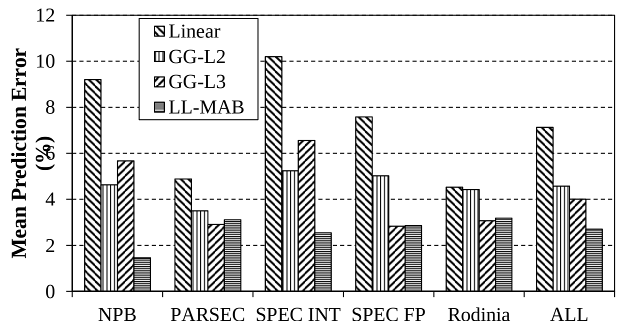


(a) Average absolute prediction error.

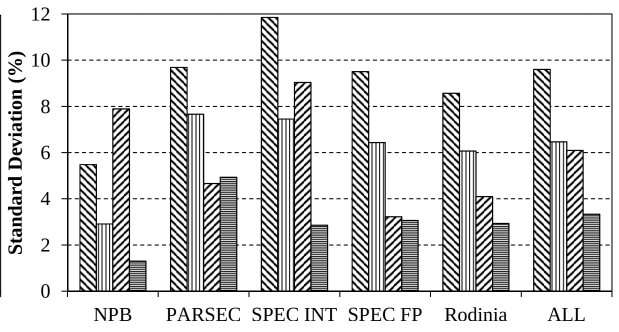


(b) Standard deviation of the absolute prediction error.

Figure 2: Average and standard deviation of prediction errors on the AMD Phenom$^{\text{TM}}$II 1090T processor.

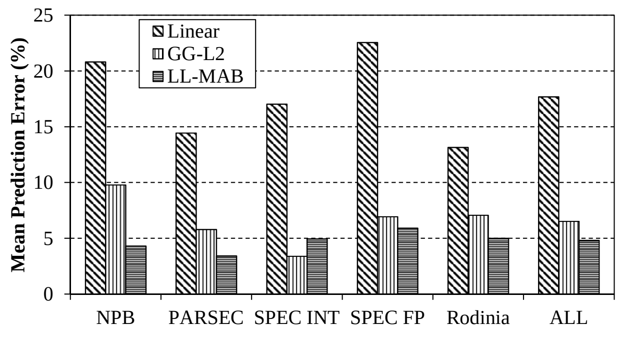


(a) Average absolute prediction error.

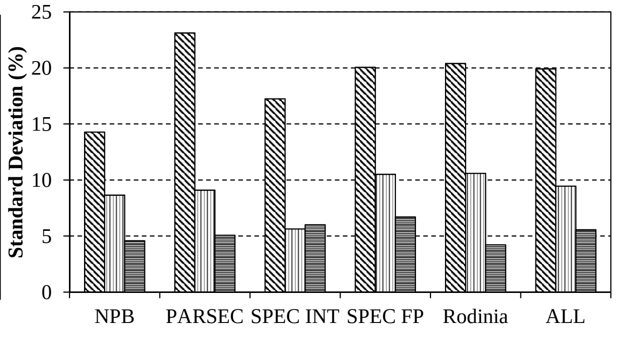


(b) Standard deviation of the absolute prediction error.

Figure 3: Average and standard deviation of prediction errors on the AMD Opteron$^{\text{TM}}$4386 processor.

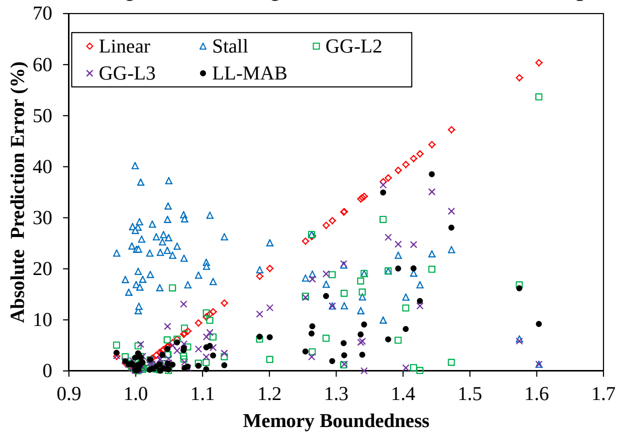


(a) Average absolute prediction error.

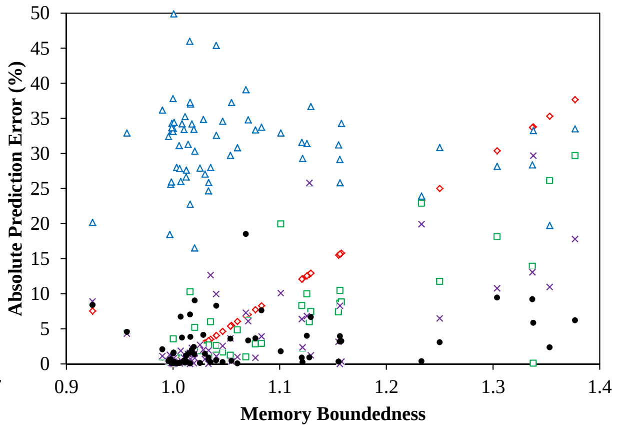


(b) Standard deviation of the absolute prediction error.

Figure 4: Average and standard deviation of prediction errors on the AMD A10-5800K processor.



(a) AMD Phenom$^{\text{TM}}$II 1090T processor



(b) AMD Opteron$^{\text{TM}}$4386 processor

Figure 5: Prediction error vs. measured memory boundedness (higher means more time in the memory system).

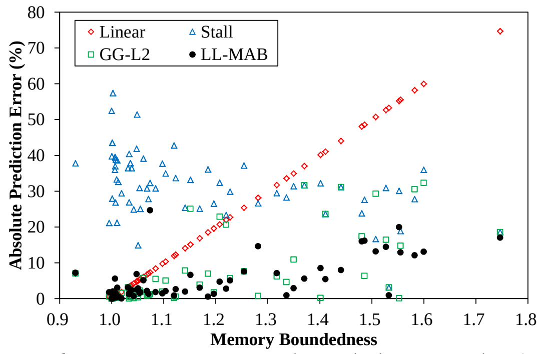


Figure 6: Error vs. memory boundedness on the AMD A10-5800K processor.

3. LL-MAB is more flexible to system configuration changes. For example, changing the DRAM frequency of a machine would not impact LL-MAB. Other models would require retraining.

Unlike the leading loads results shown in the literature, LL-MAB has a higher average error rate, between 2.5% and 5%. All three initial papers that modeled LL showed average error rates of 0.2%, though Miftakhutdinov et al. ran simulations with a more complex memory system that showed worse LL results [9]. Regardless, these results used LL counters that did not have the limitations of our MAB counter. For instance, they only counted misses to the LLC, they had no hardware prefetchers, and (often) assumed a constant delay to memory.

Others (such as those detailed by Rountree et al. [10]) demonstrate regression models on real hardware with better accuracy than LL-MAB. We did not study these in detail, because they have the disadvantage of requiring offline training and more hardware counters.

5 Conclusion and Future Work

In this paper, we presented LL-MAB, the first DVFS performance prediction model based on leading loads implemented on existing hardware. Experiments show it has better prediction accuracy than other state-of-the-art models. Moreover, it requires fewer hardware counters, is easier to use, and has less error variance. Because it is built using existing hardware, it can easily be used by software to enable online DVFS performance prediction with no further hardware changes.

Future work could include using the LL-MAB predictor over short periods for fine-grained DVFS decisions. Similarly, a regression model with this counter may show even better performance than previous regression models. Because LL-MAB requires so few hardware counters, it may also be possible to do online power estimation by monitoring other energy-hungry events.

There are also simple modifications that could increase the accuracy of the MAB event, such as filtering prefetches. Unlike the scheme described by Miftakhutdinov et al. [9], which would require at least an adder for every MAB, these approaches may yield better results with little added hardware.

Acknowledgements

We would like to sincerely thank Barry Rountree for sharing his leading loads expertise and for his help discovering the relationship between leading loads and MAB occupancy. We would also like to thank Vasileios Spiliopoulos, for his help implementing our GG model, and the anonymous reviewers. This work is partially supported by 863 Program of China (2012AA010905), NSFC (61272144, 61272143) and NUDT/Hunan Innov. Fund. For PostGrad. (B120604, CX2012B029).

References

[1] Bailey, D., Barszcz, E., Barton, J., Browning, D., Carter, R., Dagum, L., Fatoohi, R., Finebert, S., Frederickson, P., Lasinski, T., Schreiber, R., Simon, H., Venkatakrishnan, V., and Weeratunga, S. The NAS Parallel Benchmarks. Tech. Rep. RNR-94-007, March 1994.

[2] Bienia, C., Kumar, S., Singh, J. P., and Li, K. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In *Int'l Conf. on Parallel Architectures and Compilation Techniques* (2008).

[3] Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J. W., Lee, S.-H., and Skadron, K. Rodinia: A Benchmark Suite for Heterogeneous Computing. In *IEEE Int'l Symp. on Workload Characterization* (2009).

[4] Eeckhout, L. *Computer Architecture Performance Evaluation Methods*. Morgan & Claypool Publishers, 2010.

[5] Eyerman, S., and Eeckhout, L. A Counter Architecture for Online DVFS Profitability Estimation. *IEEE Trans. on Computers 59*, 11 (2010), 1576–1583.

[6] Henning, J. L. SPEC CPU2006 Benchmark Descriptions. *ACM SIGARCH Computer Architecture News 34*, 4 (2006), 1–17.

[7] Keramidas, G., Spiliopoulos, V., and Kaxiras, S. Interval-Based Models for Run-Time DVFS Orchestration in Superscalar Processors. In *Int'l Conf. on Computing Frontiers* (2010).

[8] McVoy, L., and Staelin, C. lmbench: Portable tools for performance analysis. In *USENIX Annual Technical Conf.* (1996).

[9] Miftakhutdinov, R., Ebrahimi, E., and Patt, Y. N. Predicting Performance Impact of DVFS for Realistic Memory Systems. In *Int'l Symp. on Microarchitecture* (2012).

[10] Rountree, B., Lowenthal, D. K., Schulz, M., and de Supinski, B. R. Practical Performance Prediction Under Dynamic Voltage Frequency Scaling. In *Int'l Green Computing Conf. and Workshops* (2011).

[11] Snowdon, D. C., Linden, G. V. D., Petters, S. M., and Heiser, G. Accurate Run-Time Prediction of Performance Degradation Under Frequency Scaling. In *Workshop on Operating System Platforms for Embedded Real-Time Applications* (2007).

[12] Spiliopoulos, V., Kaxiras, S., and Keramidas, G. Green Governors: A Framework for Continuously Adaptive DVFS. In *Int'l Green Computing Conf. and Workshops* (2011).