An Evaluation of Per-Chip Nonuniform Frequency Scaling on Multicores *

Xiao Zhang Kai Shen Sandhya Dwarkadas Rongrong Zhong
Department of Computer Science, University of Rochester
{xiao, kshen, sandhya, rzhong}@cs.rochester.edu

Abstract

Concurrently running applications on multiprocessors may desire different CPU frequency/voltage settings in order to achieve performance, power, or thermal objectives. Today's multicores typically require that all sibling cores on a single chip run at the same frequency/voltage level while different CPU chips can have non-uniform settings. This paper targets multicore-based symmetric platforms and demonstrates the benefits of per-chip adaptive frequency scaling on multicores. Specifically, by grouping applications with similar frequency-to-performance effects, we create the opportunity for setting a chip-wide desirable frequency level. We run experiments with 12 SPECCPU2000 benchmarks and two server-style applications on a machine with two dual-core Intel "Woodcrest" processors. Results show that per-chip frequency scaling can save ∼20 watts of CPU power while maintaining performance within a specified bound of the original system.

1 Introduction and Background

Dynamic voltage and frequency scaling (DVFS) is a hardware mechanism on many processors that trades processing speed for power saving. Typically, each CPU frequency level is paired with a minimum operating voltage so that a frequency reduction lowers both power and energy consumption. Frequency scaling-based CPU power/energy saving has been studied for over a decade. Weiser *et al.* [17] first proposed adjusting the CPU speed according to its utilization. The basic principle is that when the CPU is not fully utilized, the processing capability can be lowered to improve the power efficiency. The same principle was also applied to workload concentration with partial system shutdown or dynamic DVFS in server clusters [6,7]. Bianchini and Rajamony [5] provide a thorough survey of energy-saving techniques for servers circa 2004.

When the CPU is already fully utilized, DVFS may be applied to reduce the CPU speed when running memory intensive applications. The rationale is that memory-bound applications do not have sufficient instruction-level parallelism to keep the CPU busy while waiting for memory accesses to complete, and therefore decreasing their CPU frequency will not result in a significant performance penalty. Previous studies along this direction [9, 11, 18] largely focused on exploring power saving opportunities within individual applications. Little evaluation has been done on frequency scaling for multiprogrammed workloads running on today's multicore platforms.

Multicore frequency scaling is subject to an important constraint. Since most current processors use off-chip voltage regulators (or a single on-chip regulator for all cores), they require that all sibling cores be set to the same voltage level. Therefore, a single frequency setting applies to all active cores on Intel mutlicore processors [2,14]. AMD family 10h processors do support per-core frequency selection, but they still maintain the highest voltage level required for all cores [3], which limits power savings. Per-core on-chip voltage regulators add design complexity and die real estate cost and are a subject of ongoing architecture research [10]. Recent work by Merkel and Bellosa [13] recognized this design constraint and showed how to schedule applications on a single chip in order to achieve the best energy-delay product (EDP).

Due to the scalability limitations of today's multicores, multichip, multicore machines are commonplace. Such machines often use a symmetric multiprocessor design, with each of the multiple processor chips containing multiple cores. On these machines, nonuniform frequency scaling can still be achieved on a per-chip basis. The goal of this paper is to evaluate the potential benefits of such per-chip frequency scaling of realistic applications on today's commodity processors. To enable chip-wide frequency scaling opportunities, we group applications with similar frequency-to-performance behavior so that they run on sibling cores of the same processor chip. Using a variable-frequency performance model, we then configure an appropriate frequency setting for each chip.

Experimental Setup Our experimental platform is a 2-chip machine and each processor chip is an Intel "Wood-

*This work was supported in part by NSF grants CNS-0411127, CCF-0448413, CNS-0509270, CNS-0615045, CNS-0615139, CCF-0702505, CNS-0834451, and CCF-0937571; by NIH grants 5 R21 GM079259-02 and 1 R21 HG004648-01; and by several IBM Faculty Partnership Awards.

crest" dual-core (two cores operating at 3 GHz and sharing a 4 MB L2 cache). We modified Linux 2.6.18 to support per-chip DVFS at 2.67, 2.33, and 2 GHz on our platform. Configuring the CPU frequency on a chip requires writing to platform-specific registers, which takes around 300 cycles on our processor. Because the off-chip voltage switching regulators operate at a relatively low speed, it may require some additional delay (typically at tens of microsecond timescales [10]) for a new frequency and voltage configuration to take effect.

Our experiments employ 12 SPECCPU2000 benchmarks (applu, art, bzip, equake, gzip, mcf, mesa, mgrid, parser, swim, twolf, wupwise) and two server-style applications (TPC-H and SPECjbb2005).

2 Prototype System Design

2.1 Multichip Workload Partitioning

To maximize power savings from per-chip frequency scaling while minimizing performance loss, it is essential to group applications with similar frequency-to-performance behavior to sibling cores on a processor chip. A simple metric that indicates such behavior is the application's on-chip cache miss ratio—a higher miss ratio indicates a larger delay due to off-chip resource (typically memory) accesses that are not subject to frequency scaling-based speed reduction. We therefore group applications with similar last level cache miss ratios to run on the same multicore processor chip. We call this approach *similarity grouping*.

A natural question is how our workload partitioning would affect system performance before DVFS is applied. Merkel and Bellosa [13] profiled a set of SPEC-CPU benchmarks and found that memory bus bandwidth (rather than cache space) is the most critical resource on multicores. Based on this observation, they advocated running a mix of memory-bound and CPU-bound applications at any given time on a single multicore platform in order to achieve the best EDP. Although their *complementary mixing* appears to contradict our *similarity grouping*, in reality, they accomplish one of the same goals of more uniform memory demand. Their approach focuses on temporal scheduling of applications on a single chip, while similarity grouping focuses on spatial partitioning of applications over multiple chips. Similarity grouping has the additional advantage of being able to individually control the voltage and frequency of the separate chips.

In addition to the ability to save power by slowing down the processor without loss in performance for high miss ratio applications, miss ratio similarity grouping may lead to more efficient sharing of the cache on multicore chips. Applications typically exhibit high miss ratios because their working sets do not fit in the cache.

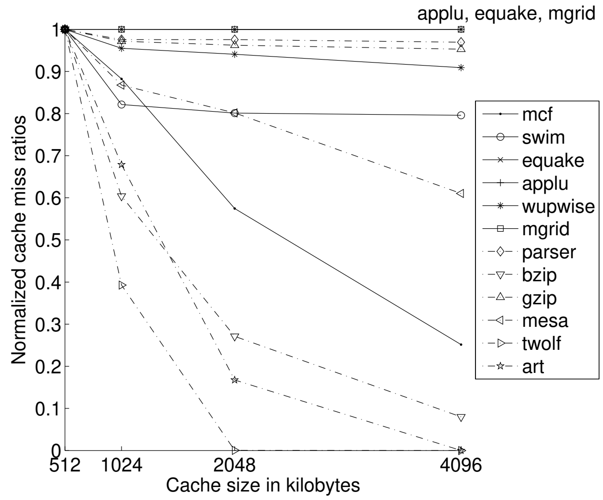

Figure 1: Normalized miss ratios of 12 SPECCPU2000 benchmarks at different cache sizes. The normalization base for each application is its miss ratio at 512 KB cache space. Cache size allocation is enforced using page coloring [20]. Solid lines mark the six applications with the highest miss ratios while dotted lines mark the six applications with the lowest miss ratios.

Increasing available cache space is not likely to improve performance significantly until the cache size exceeds the working set. This can be observed from the L2 cache miss ratio curves of 12 SPECCPU2000 benchmarks, shown in Figure 1. With the exception of mcf, most high miss ratio applications (applu, equake, mgrid, swim, and wupwise) show small or no benefits with additional cache space beyond 512 KB. In fact, the applications will aggressively occupy the cache space, resulting in adverse effects on co-running applications on sibling cores. Similarity grouping helps reduce these adverse effects by separating low miss ratio applications that may be more sensitive to cache pressure so that they run on a different chip.

2.2 Model-Driven Frequency Setting

To realize target performance or power saving objectives, we need an estimation of the target metrics at candidate CPU frequency levels. Several previous studies [9, 18] utilized offline constructed frequency selection lookup tables. Such an approach requires a large amount of offline profiling. Merkel and Bellosa employed a linear model based on memory bus utilization [13] but it only supports a single frequency adjustment level. Kotla *et al.* constructed a performance model for variable CPU frequency levels [11]. Specifically, they assume that all cache and memory stalls are not affected by the CPU frequency scaling while other delays are scaled in a linear fashion. Their model was not evaluated on real frequency scaling platforms.

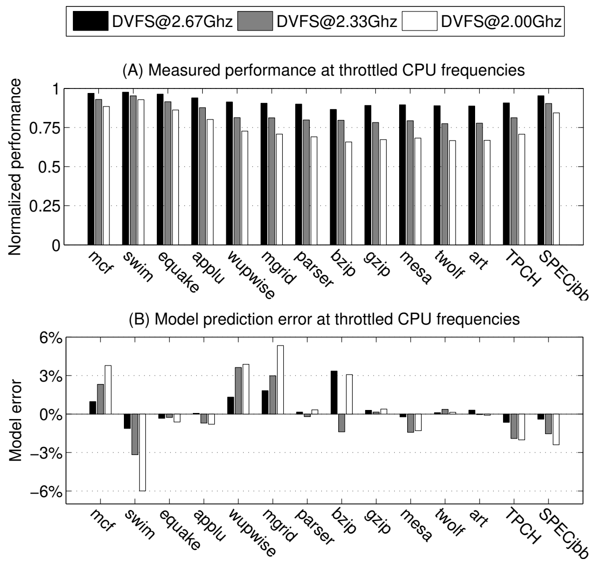

Figure 2: The accuracy of our variable-frequency performance model. Subfigure (A) shows the measured normalized performance (to that of running at the full CPU speed of 3 GHz). Subfigure (B) shows our model's prediction error (defined as $\frac{\text{prediction}-\text{measurement}}{\text{measurement}}$).

In practice, on-chip cache accesses are also affected by frequency scaling, which typically applies to the entire chip. We corrected this aspect of Kotla's model [11]. Specifically, our variable-frequency performance model assumes that the execution time is dominated by memory and cache access latencies, and that the execution of all other instructions can be overlapped with these accesses. Accesses to off-chip memory are not affected by frequency scaling while on-chip cache access latencies are linearly scaled with the CPU frequency. Let $T(f)$ be the average execution time of an application when the CPU runs at frequency f. Then:

$$T(f) \propto \frac{F}{f} \cdot (1 - R_{\text{cachemiss}}) \cdot L_{\text{cache}} + R_{\text{cachemiss}} \cdot L_{\text{memory}},$$

where F is the maximum CPU frequency. L_{cache} and L_{memory} are access latencies to the cache and memory respectively measured at full speed. We assume that these access latencies are platform-specific constants that apply to all applications. Using a micro-benchmark, we measured the average cache and memory access latencies to be around 3 and 121 nanoseconds respectively on our experimental platform. The miss ratio $R_{\text{cachemiss}}$ represents the proportion of data accesses that go to memory. Specifically, it is measured as the ratio between the L2 cache misses (L2_LINES_IN with hardware prefetch also included) and data references (L1D_ALL_REF) performance counters on our processors [8].

The normalized performance (as compared to running at the full CPU speed) at a throttled frequency f is therefore $\frac{T(F)}{T(f)}$. Since $R_{\text{cachemiss}}$ does not change across different CPU frequency settings, we can simply use the online measured cache miss ratio to determine normalized performance online. Figure 2 shows the accuracy of our model when predicting the performance of 12 SPECCPU2000 benchmarks and two server benchmarks at different frequencies. The results show that our model achieves a high prediction accuracy with no more than 6% error for the 14 applications.

Test	Chip	Similarity grouping	Complementary mixing
#1	0	{equake, swim}	{swim, parser}
	1	{parser, bzip}	{equake, bzip}
#2	0	{mcf, applu}	{mcf, art}
	1	{art, twolf}	{applu, twolf}
#3	0	{wupwise, mgrid}	{wupwise, mesa}
	1	{mesa, gzip}	{mgrid, gzip}
#4	0	{mcf, swim, equake, applu, wupwise, mgrid}	{swim, equake, applu, wupwise, gzip, twolf}
	1	{parser, bzip, gzip, mesa, twolf, art}	{mcf, mgrid, parser, bzip, mesa, art}
#5	0	2 SPECjbb threads	1 SPECjbb thread and 1 TPC-H thread
	1	2 TPC-H threads	1 SPECjbb thread and 1 TPC-H thread

Table 1: Benchmark suites and scheduling partitions of 5 tests. Complementary mixing mingles high-low miss-ratio applications such that two chips are equally pressured in memory bandwidth. Similarity grouping separates high and low miss-ratio applications on different chips (Chip-0 hosts high miss-ratio ones in these partitions).

The variable-frequency performance model allows us to set the per-chip CPU frequencies according to specific performance objectives. For instance, we can we can maximize power savings while bounding the slowdown of any application. The online adaptive frequency setting must react to dynamic execution behavior changes. Specifically, we monitor our model parameter $R_{\text{cachemiss}}$ and make changes to the CPU frequency setting when necessary.

3 Evaluation Results

3.1 Scheduling Comparison

First, we compare the overall performance of the default Linux (version 2.6.18) scheduler, complementary mixing (within each chip), and similarity grouping (across chips) scheduling policies. We design five multiprogrammed test scenarios using our suite of applications. Each test includes both memory intensive and non-intensive benchmarks. Benchmarks and scheduling partitions are detailed in Table 1.

Figure 3 compares the performance of the different scheduling policies when both chips are running at full CPU speed. For each test, the geometric mean of the applications' performance normalized to the default scheduler is reported. On average, similarity grouping is about

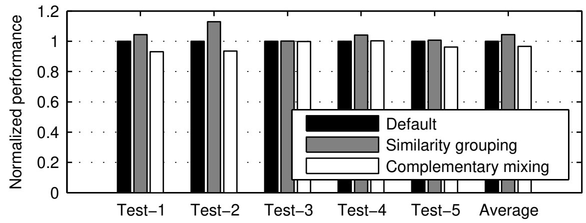

Figure 3: Performance (higher is better) of the different scheduling policies at full CPU speed.

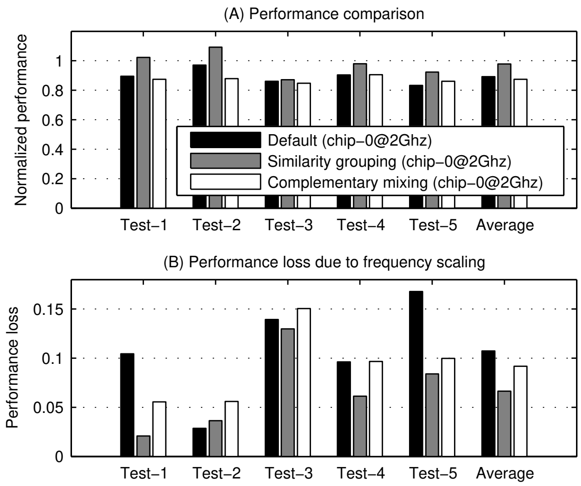

Figure 4: Performance comparisons of different scheduling policies when Chip-0 is scaled to 2 GHz. In subfigure (A), the performance normalization base is the default scheduling without frequency scaling in all cases. In subfigure (B), the performance loss is calculated relative to the same scheduling policy without frequency scaling in each case.

4% and 8% better than default and complementary mixing respectively. As explained in Section 2.1, the performance gains are due to reduced cache space interference when using similarity grouping. We also measure the power consumption of these policies using a WattsUpPro meter [1]. Our test platform consumes 224 watts when idle and 322 watts when running our highest power-consuming workload. We notice that similarity grouping consumes slightly more power, up to 3 watts as compared to the default Linux scheduler. However, the small power increase is offset by its superior performance, leading to improved power efficiency.

Next, we examine how performance degrades when the frequency of one of the two chips is scaled down. Default scheduling does not employ CPU binding and applications have equal chances of running on any chip, so deploying frequency scaling on either Chip-0 or Chip-1 has the same results. We only scale Chip-0 for similarity grouping scheduling since it hosts the high miss-ratio applications. For complementary mixing, scaling Chip-0

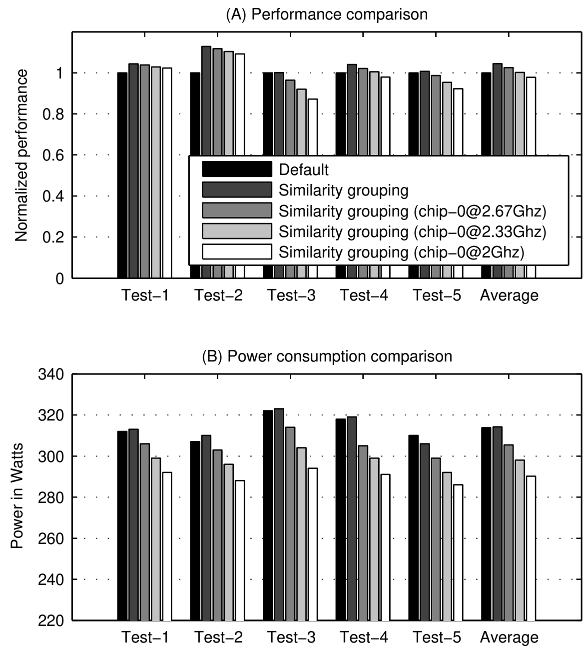

Figure 5: Performance and power consumption for per-chip frequency scaling under the similarity grouping schedule. Subfigure (B) only shows the range of active power (from idle power at around 224 watts), which is mostly consumed by the CPU and memory in our platform.

shows slightly better results than scaling Chip-1. Hence, we report results for all three scheduling policies with Chip-0 scaled to 2 GHz. Figure 4 shows that similarity grouping still achieves the best overall performance and the lowest self-relative performance loss under frequency scaling.

3.2 Nonuniform Frequency Scaling

We then evaluate the performance and power consumption of per-chip nonuniform frequency scaling under similarity grouping. We keep Chip-1 at 3 GHz and only vary the frequency on Chip-0 where high miss-ratio applications are hosted. Figure 5(B) shows significant power saving due to frequency scaling—specifically, 8.4, 15.8, and 23.6 watts power savings on average for throttling Chip-0 to 2.67, 2.33, and 2 GHz respectively. At the same time, Figure 5(A) shows that the performance when throttling Chip-0 is still comparable to that with the default scheduler.

We next evaluate the power efficiency of our system. We use *performance per watt* as our metric of power efficiency. Figure 6(A) shows that, on average, per-chip nonuniform frequency scaling achieves a modest (4–6%) increase in power efficiency over default scheduling. This is because the idle power on our platform is substantial (224 watts). Considering a hypothetical energy-

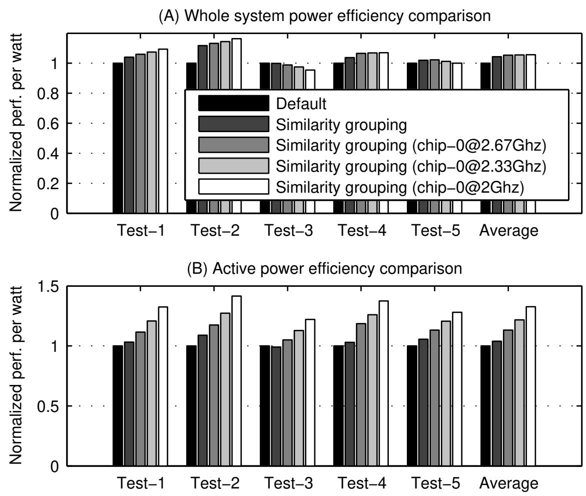

Figure 6: Power efficiency for per-chip frequency scaling under the similarity grouping schedule. Subfigure (A) uses whole system power while (B) uses active power in the efficiency calculation.

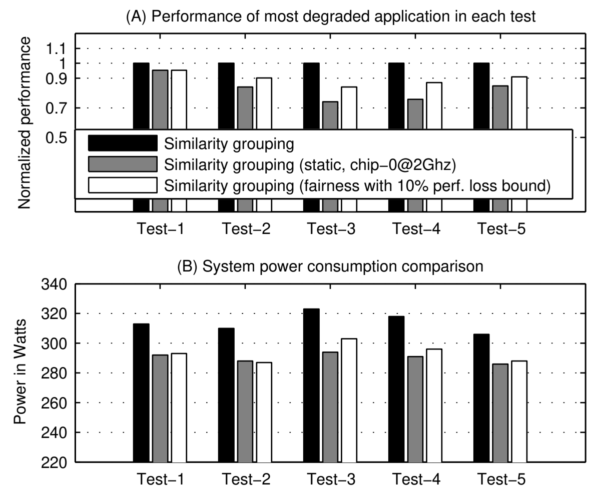

Figure 7: Performance and power consumption for static (2 GHz) and fair per-chip frequency scaling under the similarity grouping scheduling.

proportional computing platform [4] on which the idle power is negligible, we use the active power (full operating power minus idle power) to estimate the power efficiency improvement. In this case, Figure 6(B) shows more sizable gaps. Scaling Chip-0 at 2.67, 2.33, and 2 GHz achieves 13%, 21%, and 32% better active power efficiency respectively.

3.3 Application Fairness

While it shows encouraging overall performance, the per-chip nonuniform frequency scaling even with similarity grouping does not provide any performance guarantee for individual applications. For example, setting Chip-0 to 2 GHz causes a 26% performance loss for mgrid as compared to the same schedule without frequency scaling.

To be fair to all applications, we want to achieve power savings with bounded individual performance loss. Based on the frequency-performance model described in Section 2.2, our system will periodically (every 10 milliseconds) adjust the frequency setting if necessary to bound the performance degradation of running applications (*e.g.*, a target of 10% degradation in this experiment). Note that in this case the system may scale down any processor chip as long as the performance degradation bound is not exceeded.

Figure 7(A) shows the normalized performance of the most degraded application in each test. We observe that fairness-controlled frequency scaling is closer than the static (2 GHz) scaling to the 90% performance target. It completely satisfies the bound for three tests while it exhibits slight violations in test-3 and test-4. The most degraded application in these cases is mgrid, whose performance is 6% and 3% away from the 90% target in test-3 and test-4 respectively. Figure 2 shows that our model over-estimates mgrid's performance by up to 6%. This inaccuracy causes the fairness violation in test-3 and test-4. Figure 7(B) shows power savings for both static (2 GHz) and fairness-controlled frequency scaling. Fairness-controlled frequency scaling provides better quality-of-service while achieving comparable power savings to the static scheme.

4 Discussions

We have seen a slow but stable trend of increasing core numbers on a single chip, which will exacerbate the contention for memory bandwidth. Fortunately, memory technology advancement has significantly mitigated this problem. Measured using the STREAM benchmark [12], our testbed with 3 GHz CPUs (two dual-core chips) and 2GB DDR2 533 MHz memory achieves 2.6 GB/sec memory bandwidth. In comparison, a newer Intel Nehalem machine with 2.27 GHz CPUs (one quad-core chip) and 6GB DDR3 1,066 MHz memory achieves an 8.6 GB/sec memory bandwidth.

The idle power constitutes a substantial part (about 70%) of the full system power consumption on our testbed, which questions the practical benefits of optimizations on active power consumption. However, we are optimistic that future hardware designs will trend toward more energy-proportional platforms [4]. We have already observed this trend—the idle power constitutes a smaller part (about 60%) of the full power on the newer Nehalem machine. In addition, our measurement shows that per-chip nonuniform frequency scaling can reduce the average CPU temperature (by up to 5 degrees Celsius, averaged over four cores), which may lead to additional power savings on cooling.

Per-chip CPU frequency scaling is largely orthogonal to the management of shared resources on multicore processors. In particular, our frequency scaling scheme partitions applications among multiple multicore chips on the machine and mainly targets power consumption while resource management techniques such as cache space partitioning [20] and nonuniform core throttling [19] further regulate resource competition within each multicore chip.

Evaluation in this paper focuses on multiprogrammed workloads. When a single server application (consisting of many concurrent requests) runs on the machine, it may also be beneficial to group requests with similar frequency-to-performance behavior for per-chip adaptive frequency scaling. This would be possible with on-the-fly identification of request execution characteristics [15, 16] for online grouping and control.

5 Conclusion

In this paper, we advocate a simple scheduling policy that groups applications with similar cache miss ratios on the same multicore chip. On one hand, such scheduling improves the performance due to reduced cache interference. On the other hand, it facilitates per-chip frequency scaling to save CPU power and reduce heat dissipation. Guided by a variable-frequency performance model, our CPU frequency scaling can save about 20 watts of CPU power and reduce up to 5 degrees Celsius of CPU temperature on average on our multicore platform. These benefits were realized without exceeding the performance degradation bound for almost all applications. This result demonstrates the strong benefits possible from per-chip adaptive frequency scaling on multichip, multicore platforms.

References

[1] Watts Up Power Meter. https://www.wattsupmeters.com.

[2] Intel turbo boost technology in intel core microarchitecture (nehalem) based processors, Nov. 2008.

[3] AMD BIOS and kernel developer's guide (BKDG) for AMD family 10h processors, Sept. 2009.

[4] L. A. Barroso and U. Hölzle. The case for energy-proportional computing. *IEEE Computer*, 40(12):33–37, Dec. 2007.

[5] R. Bianchini and R. Rajamony. Power and energy management for server systems. In *IEEE Computer*, volume 37, Nov. 2004.

[6] J. Chase, D. Anderson, P. Thakar, and A. Vahdat. Managing energy and server resources in hosting centers. In *Proc. of the 18th ACM Symp. on Operating Systems Principles*, Banff, Canada, Oct. 2001.

[7] E. N. Elnozahy, M. Kistler, and R. Rajamony. Energy-efficient server clusters. In *Proc. of the 2nd Workshop on Power-Aware Computing Systems*, Feb. 2002.

[8] IA-32 Intel architecture software developer's manual, volume 3: System programming guide, Mar. 2006.

[9] C. Isci, G. Contreras, and M. Martonosi. Live, runtime phase monitoring and prediction on real systems with application to dynamic power management. In *International Symposium on Microarchitecture*, Orlando, FL, Dec. 2006.

[10] W. Kim, M. S. Gupta, G.-Y. Wei, and D. Brooks. System level analysis of fast, per-core DVFS using on-chip switching regulators. In *HPCA'08*, Salt Lake City, UT, Feb. 2008.

[11] R. Kotla, A. Devgan, S. Ghiasi, T. Keller, and F. Rawson. Characterizing the impact of different memory-intensity levels. In *IEEE 7th Annual Workshop on Workload Characterization*, Austin, Texas, Oct. 2004.

[12] J. McCalpin. Memory bandwidth and machine balance in current high performance computers. In *IEEE Technical Committee on Computer Architecture newsletter*, 1995.

[13] A. Merkel and F. Bellosa. Memory-aware scheduling for energy efficiency on multicore processors. In *Workshop on Power Aware Computing and Systems, HotPower'08*, San Diego, CA, Dec. 2008.

[14] A. Naveh, E. Rotem, A. Mendelson, S. Gochman, R. Chabukswar, K. Krishnan, and A. Kumar. Power and thermal management in the Intel Core Duo processor. *Intel Technology Journal*, 10(2):109–122, 2006.

[15] K. Shen. Request behavior variations. In *15th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 103–116, Pittsburgh, PA, Mar. 2010.

[16] K. Shen, M. Zhong, S. Dwarkadas, C. Li, C. Stewart, and X. Zhang. Hardware counter driven on-the-fly request signatures. In *13th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 189–200, Seattle, WA, Mar. 2008.

[17] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *First USENIX Symp. on Operating Systems Design and Implementation (OSDI)*, pages 13–23, 1994.

[18] 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*, Grenoble, France, Oct. 2002.

[19] X. Zhang, S. Dwarkadas, and K. Shen. Hardware execution throttling for multi-core resource management. In *USENIX Annual Technical Conference (USENIX)*, Santa Diego, CA, June 2009.

[20] X. Zhang, S. Dwarkadas, and K. Shen. Towards practical page coloring-based multicore cache management. In *4th European Conf. on Computer systems*, Nuremberg, Germany, Apr. 2009.